NORDSEC 2002

Proceedings of the 7th Nordic Workshop on Secure IT Systems

"Towards Secure and Privacy-Enhanced Systems"

7- 8 November 2002, Karlstad, Sweden

Edited by

Simone Fischer-Hübner
Department of Computer Science
Karlstad University
Sweden

Erland Jonsson
Department of Computer Engineering
Chalmers University of Technology
Sweden

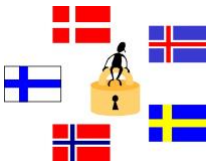

Table of Contents

Foreword

Welcome to the 7th Nordic Workshop on Secure IT Systems (NordSec 2002) - "Towards Secure and Privacy-Enhanced IT Systems" in Karlstad!

The NordSec workshops were started in 1996 with the aim of bringing together researchers and practitioners within computer security in the Nordic countries. The theme of the workshops has been applied security, i.e., all kinds of security issues that could encourage interchange and cooperation between the research community and the industrial/consumer community. The annual NordSec workshops have been held in Gothenburg, Helsinki, Trondheim, Stockholm, Reykjavik and Copenhagen.

Since 1996, the area of IT security has even gained more importance for academia, industry and government. Computer malware, distributed denial of service attacks and threats of cyber crime and cyber terrorism have recently again demonstrated the vulnerability of our Network Society and the need of secure IT systems. Also the user's privacy is increasingly at risk, especially in the mobile Internet, where the user's location, preferences, behavior could be monitored and recorded in detailed user profiles. Hence, research on and development of secure and privacy-enhanced systems will be of key importance for the success of future information and communication infrastructures.

NordSec 2002 with the theme "Towards Secure and Privacy-Enhanced IT Systems" is organised by the Department of Computer Science at Karlstad University in Sweden in cooperation with SIG Security, IEEE Sweden - Computer/Software Engineering Chapter, the Swedish Centre for Internet Technology (SCINT) and Ericsson AB. It is addressing IT security research topics on networks, public-key infrastructures (PKI), cryptography, formal methods, access controls models, human and educational aspects and security management. Also, within the main theme of NordSec 2002 a special track is devoted to privacy enhancing technologies.

We were fortunate to have the choice of a large number of international contributions. Out of 45 submitted full papers, 17 full papers have been selected by the programme committee. Each of them was reviewed by at least three referees. Besides, 5 short "work in progress" presentations were selected for the workshop programme. Workshop speakers come from eleven different countries from Europe, Asia and the United States. Our invited speakers address security and privacy aspects of mobility: Prof. David Sands (Chalmers University of Technology, Gothenburg / Sweden) talks on "Mobile Code Security: Some Challenges and Possibilities" and Prof. Kai Rannenberg (Goethe University Frankfurt / Germany) presents "Mobile Applications and Mulitlateral Security".

It is our pleasure to thank the members of the programme committee, the additional reviewers and the members of the organising committee. Without their work and dedication, NordSec 2002 would not have been possible. Besides, we owe special thanks to Karlstad University, SIG Security, IEEE Sweden Section Computer/Software Engineering Section, the Swedish Centre for Internet Technology (SCINT) and Ericsson AB for financial and organization support. Finally, we also want to thank Niklas Nikitin and Anna Brunström for providing the picture for the cover of the proceedings.

Simone Fischer-Hübner Erland Jonsson

Programme Committee:

Simone Fischer-Hübner (Karlstad University, Sweden, Co-Chair)
Erland Jonsson (Chalmers University of Technology, Sweden, Co-Chair)
Tønnes Brekne (Telenor R&D Agder, Norway)
Anna Brunström (Karlstad University, Sweden)
Mads Dam (SICS & Royal Institute of Technology, Sweden)
Úlfar Erlingsson (Green Border Technologies, USA)
Alberto Escudero Pascual (Royal Institute of Technology, Sweden)
Viiveke Fåk (Linköping University, Sweden)
Christian Gehrmann (Ericsson Mobile Platforms AB, Sweden)
Dieter Gollmann (Microsoft Research, UK)
Ulf Gustafson (Chalmers University of Technology, Sweden)
Marit Hansen (Independent Centre for Privacy Protection / Kiel, Germany)
Audun Jøsang (Distributed Systems Technology Centre, Australia)
Svein J. Knapskog (Norwegian University of Science and Technology, Norway)
Ulf Lindqvist (SRI International, USA)
Stefan Lindskog (Chalmers University of Technology &Karlstad University, Sweden)
Hanne Riis Nielson (Technical University of Denmark, Denmark)
Kaisa Nyberg (Nokia Research Center, Finland)
Kai Rannenberg (Microsoft Research, UK)
Nahid Shahmehri (Linköping University, Sweden)
Teemupekka Virtanen (Helsinki University of Technology, Finland)
Michael Waidner (IBM Research Zurich, Switzerland)
Louise Yngström (Stockholm University/RIT, Sweden)

Additional Reviewers:

Ronja Addams-Moring, Fredrik Björck, Daniel Bradley, Christian Damsgaard Jensen, Tyrone Grandison, Hans Hedbom, René Rydhof Hansen, Morton Swimmer, Douglas Wikström, Albin Zuccato

Organising Committee:

Stefan Lindskog (Chair)
Anna Brunström
Simone Fischer-Hübner
Albin Zuccato

Mobile Code Security: Some Challenges and Possibilities

David Sands

Department of Computing Science,
Chalmers University of Technology and Göteborg University, Sweden.
www.cs.chalmers.se

Abstract. The extensible functionality of modern systems invites security problems. How can we protect ourselves against malicious or faulty mobile code without loosing the evolving functionality that mobile code provides? The *language-based security* approach attempts to enforce security policies at application level by analysing and transforming the semantic behaviours of programs. In this talk I will review some of the promising approaches in the area of language-based security, and outline our work on enforcing confidentiality via covert channel elimination.

Virtual Hierarchies - An Architecture for Building and Maintaining Efficient and Resilient Trust Chains

John Marchesini and Sean Smith *
{carlo,sws}@cs.dartmouth.edu

Department of Computer Science - Dartmouth College, Hanover, NH USA
www.cs.dartmouth.edu/~pkilab

Abstract. In *Public Key Infrastructure (PKI)*, the simple, monopolistic organizational model of a single certificate issuing entity works fine until we consider real-world issues. Then, issues such as scalability and mutually suspicious organizations create the need for a multiplicity of certificate issuing entities, which introduces the problem of how to organize them to balance resilience to compromise against efficiency of path discovery. Many solutions involve organizing the infrastructure to follow a natural organizational hierarchy, but in many cases, such a natural organizational hierarchy may not exist. In this paper, we use tools such as *secure coprocessing*, *threshold cryptography*, and *peer-to-peer networking* to address the former problem by overlaying a *virtual hierarchy* on a mesh architecture of peer certificate issuing entities, and achieving both resilience and efficiency.

1 Introduction

1.1 The Problem

Background By separating the privilege to decrypt or sign a message from the privilege to encrypt or verify, *public-key cryptography* enables forms of trusted communication between parties who do not share secrets *a priori*. Eliminating the need for shared secrets has multiple advantages. On a global level, it potentially enables extending trusted communication across organizational boundaries, between parties who have never met. But it can also reduce overhead in managing communication between parties even on a local level, within one organization: the number of needed keys goes from $\Omega(n^2)$ to $O(n)$, where n is the number of entities.

PKI has many definitions; the most commonly accepted definition refers to how one participating party learns what the public key is for another party. Typically, approaches to PKI begin by condensing trust: rather than *a priori* knowing the public key of each party in the population, the relying party instead knows the public key of a designated special party, who in turn issues signed statements (e.g., certificates and CRLs) about members of the population.

* This work was supported in part by Internet2/AT&T, by IBM Research, by the Mellon Foundation, and by the U.S. Department of Justice, contract 2000-DT-CX-K001. However, the views and conclusions do not necessarily represent those of the sponsors. A preliminary version appears as TR2002-416.

This designated party is typically called the *certificate authority (CA)*. Some approaches separate the process of issuing certificates from the process of identifying and authorizing keyholders to receive the certificates; in these approaches, the latter tasks become the responsibility of the *registration authority (RA)*. Since the CA must hold and wield a private key of considerable value, implementations apply various protections to that private key, such as housing it in a hardened cryptographic module that is kept offline. Some ambiguity thus results regarding what the term "CA" refers to: the entity (typically online) that issues and manages certificates; or this entity's specific machine that houses the private key. In this paper, we use the first implication.

There are numerous models which describe how a CA and RA should be related, and who should operate them. One notion is to have an independent third party operate the CA, and let the organization with actual end-users only operate the RA. Another popular model is to have the CA and RA run by the same organization and have this enterprise PKI blessed by some higher level third party organization (e.g. Verisign).

We make the latter assumption that the CA and RA are managed by the same domain (i.e. as an *enterprise PKI*). This arrangement is discussed in RFC 2459 and in the current literature (e.g. [1–5]). Real-world deployers of this methodology include: the U.S. Treasury and State Departments, the Federal Deposit Insurance Corporation, Verisign, and many colleges and universities, including Dartmouth College.

More than One This simple PKI model of one CA servicing a user population suffers from some inherent limitations. A certificate is the CA's assertion that a keyholder possesses certain properties, such as a particular identity. In order to accept a certificate, a relying party needs to trust the CA to do two tasks: (1) to ensure that its currently valid certificates only assert bindings the CA judges to be true; and (2) to judge a binding is true only when the keyholder really has those properties. If the relying party does not know the CA—or if the CA is not in a position to verify the identity of the keyholders according to some uniform policy—then the relying party cannot reasonably accept the certificate.

These tasks can lead to conflicting constraints. Task (2) requires a relationship between the CA and the keyholders—which can lead to the enterprise PKI model [1] of multiple CA/keyholder sets. However, task (1) requires a relationship between the CA and the relying party, which creates the need to organize these PKIs so that public key operations can take place across these sets.

PKIs within an organization are becoming a common occurrence. Such systems have been well studied, and are often built from commercially available components. Within an organization which has a PKI, the certificates generated by the organization's CA are meaningful. Outside of such an organization, however, those same certificates are meaningless unless some agreement between a number of organizations is in place.

The question thus arises of how to organize multiple CAs. The basic literature (e.g. [2]) gives a serious examination of this question. Readers unfamiliar with this literature may be tempted to assert that the only natural solution here is to use a *name-constraint*

[1] Some older approaches to PKI attempt to address this conflict by isolating the CA—task (1)—but delegating registration to local RAs—task (2). However, this creates its own trust complications [5].

hierarchy: group CAs into sets that have some natural relationship; for each group, establish a new CA that certifies the CAs in that group; and continue this process upward, so that the we result in a tree with a single trust root. For example, one might follow the DNS hierarchy, and assume that a global root certifies a `edu` CA, which certifies a `dartmouth.edu` CA, which certifies a `cs.dartmouth.edu` CA, which certifies each member of our department.

Although apparently natural, this hierarchical approach has many drawbacks.

- Hierarchies are not always *scalable*, in that they cannot permit the participating fraction of the population to grow gradually. Suppose the natural social hierarchy has four levels, and two unrelated leaves want to establish a trust relationship. They can only do this if all the interior CAs—from the first leaf, up to the root, then down to the second leaf—are already participating in the PKI.
- Hierarchies are not always *usable*. The globally unique names determined by the natural hierarchy may not necessarily be usable by the humans who need to use them to make trust judgments. A colleague reports that his `foo.com` domain has 100K machines whose names are of the form `bar.foo.com`. Not only is this namespace crowded—a typo will likely give the user the wrong machine instead of an error—but it also changes dynamically: a sysadmin the user never meets changes machine names without notification. Thus, using the natural hierarchy as a basis for trust (via a hierarchical PKI) is problematic.
- Hierarchies do not always *exist.* The relationship between users and their immediate CAs is usually (but not always) natural. However, the upper regions get murky. With mobile devices [6, 7] or collections of universities or government departments, one typically encounters federations of peers with no clear natural organization.[2] Indeed, except for DNS and perhaps the Roman Catholic church, it is hard to find a natural hierarchy whose upper regions are well-defined. Which U.S. military service[3] should be the root? Which DOE laboratory?[4] Why should CREN or USPS or NIH be the root over academia or U.S. citizens at large?

PKI as a system Consequently, we are going to take the unorthodox view of looking at "PKI" as a *system* that has various properties, instead of an automatic mirror of a social arrangement that may not necessarily be appropriate, even if it exists.

We might start by thinking of conventional CAs (from the simple model above) as nodes, and trying to decide how to link them together—perhaps by creating new CAs—in a directed graph, where edges go from a CA to each entity that it certifies.

Two desirable properties of any PKI are *resilience* and *efficiency*.

- By *resilience*, we mean the ability of the system to tolerate the discovery that any given key pair has been compromised. What trust judgments become impossible? How many key pairs must be revoked and reissued?

[2] The fact that "everyone gets to be King" has been cited as motivation for some U.S. government bridge projects.

[3] The first author would assert that it's the branch to which he belonged.

[4] The second author would assert that it's the DOE laboratory for which he used to work.

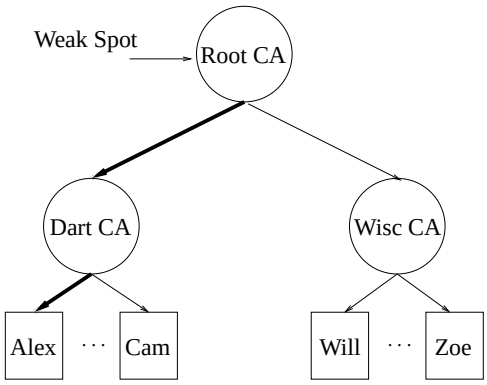

Fig. 1. In a hierarchy, when Zoe receives a certificate from Alex, she verifies that the Dartmouth CA certificate is signed by the Root CA. She then verifies that Alex's certificate is signed by the Dartmouth CA. As the number of CAs grows, it takes $O(logV)$ time to verify the path from the Root CA to the user which needs to be verified. If the Root CA is compromised in a hierarchy, the system goes down until all certificates are revoked and new certificates are issued.

– To verify a certificate, a relying party needs to find a path from a *trust root* (a node it *a priori* trusts) to the certificate in question. By *efficiency*, we mean the running time of the algorithm to discover this path.

Resilience and efficiency are typically competing goals.

Structured Centralization. Many current architectures impose a rigid structure on the organization of CAs, which means that path construction and validation can be deterministic and efficient. Although this structure permits path algorithms to traverse the topology within some efficient time constraints, it also results in a large amount of authority residing in a single place (e.g. the root CA). This centralization of authority directly decreases resilience: if the root CA is compromised, the entire PKI is unusable until it can recover.

Hierarchies are the canonical example of this structured approach. Traditionally, hierarchies achieve $O(logV)$ (where V is the number of CAs) verification time, because paths in a tree are well-defined and easy to find (Figure 1). We noted many drawbacks above; another drawback is that hierarchies place increasing amounts of value on the private keys of interior nodes. If an adversary were to compromise an upper-level CA or even the root CA, the entire PKI must suspend operation until a recovery can occur (i.e. all certificates issued by that CA are revoked, and new ones are reissued with the CA's new private key) (Figure 1).

Thus, hierarchies obtain efficiency at the cost of resilience.

Unstructured Decentralization. An opposing view involves organizing CAs in a more decentralized way in an effort to increase resilience by not placing so much authority in one centralized place. However, decentralization implies that path validation algorithms must now do more work and must often use non-determinism to decide if a

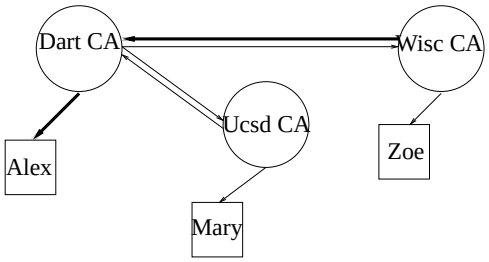

Fig. 2. Meshes offer increased resilience. If the Wisconsin CA's private key is disclosed, the other CAs can continue to operate. Only Wisconsin is affected. When Wisconsin gets back online, it may rejoin the CA network. Meshes also offer decreased efficiency. When Zoe receives a certificate from Alex, she verifies that that a trust path exists from Alex's CA to a point which she trusts by examining certificates stored in the CA directories. In this example, the Dartmouth CA and Wisconsin CA are cross-certified, so she believes Alex's certificate is good. As the number of CAs grows, it takes $O(V)$ time to verify a certificate chain.

received trust chain is valid. These properties translate into a decrease in efficiency and an increase in complexity on the part of the verifier.

Meshes are the canonical example of the unstructured approaches. Mesh PKI architectures have been developed in part to avoid this single point of failure (Figure 2). However, the non-deterministic nature of peer-to-peer organization increases the complexity of the path verification algorithm significantly (Figure 2). Due to the fact that not all possible choices lead to a trusted CA, coupled with trial-and-error construction of the trust path (a path to a trusted CA), verification time in these schemes is usually high. Further, mesh architectures make no guarantee to avoid cycles, leading to the existence of choices in the path construction algorithm which may never terminate.

Thus, meshes obtain resilience at the cost of efficiency.

Other Approaches Finding algorithms which increase the efficiency of path construction in decentralized organizations is an emerging area of research. Algorithms which use *certificate extensions* (such as name constraints and policy extensions), as well as loop elimination techniques have been developed to enhance efficiency [1]. Yet another possibility could be to use machine addressing and routing techniques, as the problems appear to be potentially isomorphic. Our concern however, is the underlying organization of CAs, and how they may be arranged to achieve efficiency and resilience.

Other common architecture schemes are more hybrid.

Extended Trust Lists are used to give users the ability to maintain lists of CAs which they choose to trust. Each entry in this list may represent a single CA or an entire PKI, which itself could be a Hierarchy or a Mesh. This scheme poses new challenges for validation algorithms, as the starting point for these algorithms could be any node in the list. The implication is that a path may have be to be constructed using every entry in the list as a starting point.

Bridge CAs provide another alternative to the common approach of cross-certifying enterprise PKIs through peer-to-peer relationships. Cross-certification without a Bridge

CA results in $(n^2 - n)/2$ relationships for n enterprises (in graph theory, this graph is known as a complete graph on n vertices, and is named K_n [8]).

The Bridge allows each of the enterprise PKIs to cross-certify to it, resulting in a star topology between enterprises and reducing the number of relationships to n. Bridges are also used for translating between different policy and legal domains. In these situations, the PKIs themselves are usually complex.

While this is an attractive solution if all of the enterprises are Hierarchies, the Bridge architecture does not solve path validation issues in general, as each of the enterprises themselves may be Meshes [2, 3].

The *Bottom Up With Name Constraints* model [4] is one which shares our goal of allowing organizations to construct their own PKI and then connect it to other organizations' PKIs. The model assumes a hierarchical namespace and that CAs are certified in both directions, down (from parent to child) and up (from child to parent). The model also allows for CAs to cross certify directly.

Path validation in this model is quite efficient due to the presence of certification in both directions. The validation algorithm begins by starting at a trust anchor and looking first for a cross-certified CA which is either an ancestor of the target or the target itself. If this fails, the algorithm proceeds up to the parent CA and searches through its cross-certified CAs. This terminates when a cross certificate is found or when a common ancestor is found.

This model also has impressive resilience properties. If a key is compromised, the compromised CA can issue new certificates to all of the CAs which are certified (up, down, and cross). The communities belonging to each certified CA will automatically be bound to the new key, without having to make any changes.

The major difference between this model and ours is that we relax the assumption of a hierarchical namespace. As mentioned earlier, hierarchies are not always usable, scalable, or present.

1.2 Our Solution

We believe that both properties—efficiency and resilience—are important to most PKI systems. We thus propose an architecture and are developing a prototype which aims to bridge the gap between these seemingly competing goals. We feel this is novel as most current architectures fail to provide both goals.

Our objective is to devise an architecture which allows for CAs to organize themselves in such a way as to maintain the following two invariants:

1. *Efficiency.* Trust chains produced by any of the entities may be verified in an efficient manner; trust chains are loop-free. This is common in hierarchy schemes.
2. *Resilience.* The secrets (private keys) do not exist in any one place; for upper-level CAs, private-key computations require collaboration. The parties are fairly randomly distributed throughout the topology.

This architecture is useful when the relative authority of the CAs in the real world is not easily represented by a strict hierarchy and when certificates need to be used frequently outside of the issuing namespace.

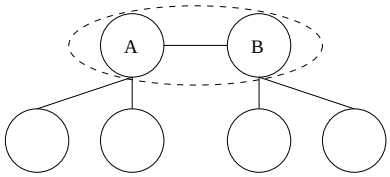

Fig. 3. A single collective. All of the nodes are CAs. The nodes inside of the oval (A and B) are maintaining a portion of the private key privilege (e.g. via threshold cryptography), which is acting as the Root CA key. The other nodes are directly connected to one of the nodes which collaborate in cryptographic computations for the collective and may "use" the key (i.e. make a broadcast to have a message signed by the nodes in the oval). It should be noted that it is possible to put all of the nodes inside the oval, meaning that each member of the collective would maintain a portion of the private key privilege.

Overview The mechanism we propose to accomplish this task is a *virtual hierarchy*—a logical hierarchy formed in a peer-to-peer network. As with a standard hierarchy, we can model a virtual hierarchy as a tree with nodes and directed edges. Leaves can represent bottom-level users; their parents represent their natural CA.

However, the remaining nodes are *virtual* CAs. Although each such node is a logical entity in the virtual hierarchy, it represents the *collective* action of a set of conventional CAs. Consequently, we use the term *collective* for this set. (See Figure 3.) In the virtual hierarchy, such collectives of conventional CAs comprise all the upper-level CAs.

We obtain this collective action via cryptography. There are a number of cryptographic schemes which require collaboration in order to perform cryptographic computations. The broad category for such methods is known as *threshold cryptography*. Some examples include *multi-party signature schemes* [9] and *Multi-Party RSA* [10], which we will discuss in Section 3.4. Even *secret sharing* [11] might qualify as a primitive threshold scheme.

To simplify exposition, we frame our prototype in terms of secret splitting, where the parties in a subset of a collective each hold a fragment of the collective's private key. To wield this key, the parties reconstitute it—leading to the significant drawback that a single party might retain the key. More advanced threshold schemes do not share this weakness; Section 3.2 discusses these issues further.

We have developed (and are prototyping) algorithms that allow natural CAs to form collectives in an ad hoc manner. They then organize collectives into a hierarchy (where a virtual node can itself become certified by another collective) in order to maintain a good tree structure. (See Figure 4.)

This approach thus obtains the goals we desired:

1. *Efficiency.* By maintaining the structure of a hierarchy, we retain an expected $O(logV)$ trust chain verification cost, with no loops.
2. *Resilience.* By distributing the higher-level CA private key privileges among multiple parties, we retain the resilience of decentralized approaches.

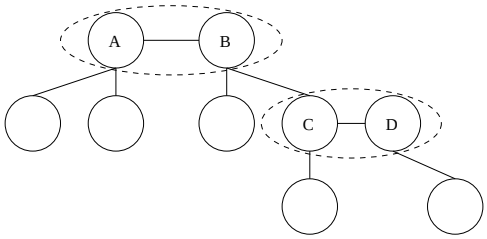

Fig. 4. A hierarchy is emerging. Here there are two collectives, linked by the CA denoted as "C". C is a member of the original collective shown in Figure 3, and is a member collaborating in cryptographic computations (along with D) of the second level collective. The ovals contain nodes which share private key privileges.

Application Layer	CAs reside here.
Virtual Hierarchy Layer (VHL)	Logical hierarchy layer.
Trusted Peer Access Layer (TPA)	Peer-to-peer layer.
TCP/IP Protocol Stack	The TPA uses TCP.

Fig. 5. The protocol stack.

Physical Layer The physical layer is a peer-to-peer network of secure coprocessors [12] (we use the IBM 4758 [5]). The secure coprocessor is not strictly necessary to make the virtual hierarchy layer work. However, since nodes in this layer are CAs, they must all have a cryptographic module; however, tamper-resistance and remote attestation/outbound authentication adds security against insider attack at any one node. Practically speaking, part of our decision to use secure coprocessors came from the fact that we already had some devices, we had some familiarity with the programming environment, and the modules we had are validated to FIPS 140-1 Level 4.

2 Overall Structure

We approached the problem in two stages, the first was to implement a peer access layer which allows secure coprocessor to communicate securely, and the second was to implement the virtual hierarchy algorithms on top of that layer. The resulting protocol stack is depicted in Figure 5.

[5] Recently, security vulnerabilities [13, 14] have been demonstrated in an application (IBM's CCA) which runs on the 4758. It should be noted that these vulnerabilities belong to the application, and not the 4758 platform. At the time of writing, the 4758 has no known vulnerabilities.

Our prototype implements the Virtual Hierarchy Layer (VHL) and the Trusted Peer Access Layer (TPA). The prototype version of the VHL contains a command line interface so that we do not need to integrate with a CA at this stage in development. The peer access layer running inside of the IBM 4758 is depicted as the TPA layer, and the algorithms which construct and maintain the logical hierarchy are shown as the VHL. The two layers are implemented as separate processes, with the output of the VHL being piped into the TPA using standard UNIX pipes.

Before we discuss the layers in detail, a simple example will be useful in understanding the high level operation and what we are trying to achieve. In the example shown in Figures 6 through 10, two machines A and B will connect, negotiate a private key, and store half of the private key privilege. This operation forms the root collective. Four more machines will join the collective, and are able to "use" the private key, whose privilege is maintained by A and B. The result is similar to Figure 3. Then a new collective will be formed by C and D, and a new node will be placed in tree (i.e. the virtual hierarchy, shown in the "VHL" column of the diagrams). This is a simple example, building a virtual hierarchy with only two nodes, but will serve to familiarize the reader with the basic concept. As more CAs make connection requests in the TPA, the tree in the VHL will continue to grow downward.

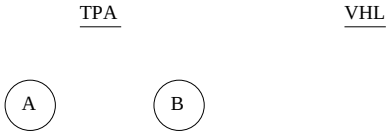

Fig. 6. Step 1: The two machines are not connected, and neither of them share the private key privilege for a virtual CA.

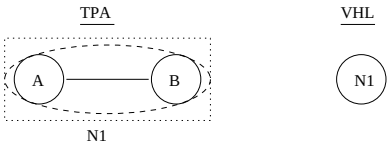

Fig. 7. Step 2: A connects to B and they establish a virtual CA, resulting in the two parties now sharing a portion of the private key privilege (as denoted by the oval). A collective is formed and a node $N1$ is established in the virtual hierarchy.

3 Virtual Hierarchy Layer

From the highest level, the virtual hierarchy (i.e. the logical hierarchy in the peer-to-peer network) is constructed by an algorithm that allows peer CAs to establish a secure

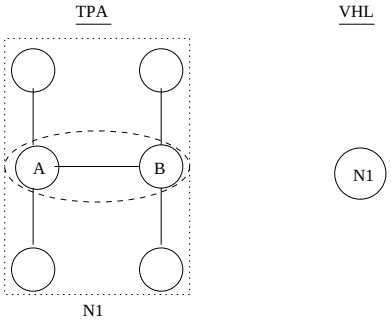

Fig. 8. Step 3: More nodes join the collective by connecting to one of the machines already in the collective (A or B). None of the new nodes are required to negotiate keys, as the maximum size of a collective for this example is six. Since the four new CAs are allowed to sign statements with the key held whose privilege is shared by A and B (as in Figure 3), the virtual hierarchy remains unchanged.

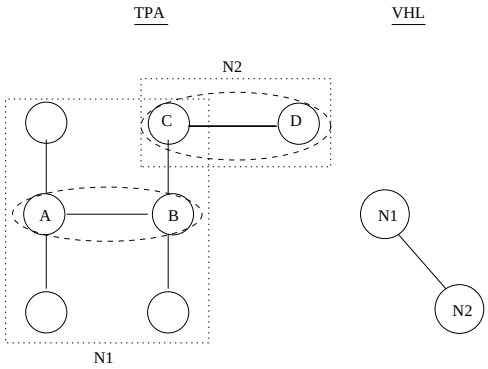

Fig. 9. Step 4: Machine D makes a connection to machine C. Because D is at a distance greater than one from one of machines acting as a virtual CA (machines A and B in this example), C and D must establish a new virtual CA and share a portion of the private key privilege (as in step 2 above). This operation forms a new collective and a new node $N2$ in the virtual hierarchy. Note that CA C is now in both collectives.

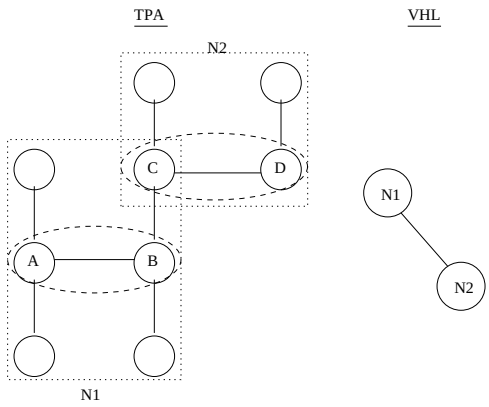

Fig. 10. Step 5: Two CAs make connections to members of the new collective (C or D), and join the new collective. Since a private key has already been established for that collective (whose privilege is shared by C and D), and the maximum size is six, the new members do not need to negotiate a key (as in Step 3). The virtual hierarchy remains unchanged.

connection and negotiate a secret which each of their communities may use as an end-point in their trust chain. Pieces of the trust root's privilege are then stored among the peers who negotiate it.

This leads us to make the following two claims:

Increased Resilience The result of this negotiation produces a root "entity" whose privilege is distributed among the n parties who are at a distance one (i.e. directly connected) to one of the actors in the negotiation. This group of n parties is a collective and all act as though the "entity" is their root CA. The result of requiring collaboration to perform cryptographic computation among a group of peers alleviates the single point of failure problem. While the number of targets increases, the payoff for successfully compromising a target decreases. Minimally, an attacker must successfully compromise at least as many targets as are sharing the private key privilege to get the private key. As with pro-active security [15], this makes the scheme more resilient to compromise.

Increased Efficiency The result of this negotiation produces a root "entity" whose role is to act as a trust point for the n parties who are at a distance one to one of the actors in the negotiation. We argue that this maintains a hierarchical trust structure similar to one which would be found in a physical hierarchy of CAs. Maintaining this hierarchy allows trust calculations to be performed at an average of $O(logV)$ time (again where V is the number of CAs participating in the network).

It is important to realize that our solution, like other current schemes, would require each community to perform some amount of work to reflect the new topology. Time and space complexity analysis of this task is an area for future work.

3.1 Registration

In order for a CA to join a collective, it must offer a statement of its policy and practices to the collective for review. If and only if the collective reaches a consensus to allow the CA to join, will the CA be admitted. The same process holds for collectives which join other collectives.

There are a number of good questions that arise here. How is this policy expressed? How is it evaluated? How does a collective reach a consensus? While these questions are all important, they are all orthogonal to the idea we are presenting. We have provided a framework in which to explore a range of possible solutions. In our prototype, we rely on the presence of a 4758 running our software. As noted earlier, the 4758's outbound authentication facilities [16], or the remote attestation features of Trusted Computing Platform Alliance (TCPA) or Palladium could provide more assurance, in that they could verify things such as code load and CA software.

3.2 Simplifying Invariants

Our algorithm follows several rules that constrain and simplify the problem.

In order to maintain the property that verification may be done in $O(logV)$, we have designed our algorithm to maintain the invariant that there are no cycles in the connection graph produced by the connection network of CAs. These connections are accomplished using the protocol TPA Layer.

We maintain this invariant because if we were to allow cycles at this layer, we would break the hierarchical structure by transforming it from a tree into a less-structured graph. Breaking the hierarchical structure would have the following two implications:

First, to perform an efficient path verification algorithm in this graph, the algorithm would need to locate the shortest correct (i.e. matching the certificate chain) path. This would take longer than $O(logV)$ in the average case.

Second, any such algorithm would require state to be maintained so that the shortest correct path may be calculated without having to account for the time it takes to discover the topology in real time. This could be accomplished by implementing a "smarter" routing algorithm in the TPA Layer (e.g. reverse path forwarding [17]). Because we maintain our no-cycle invariant, we can instead use simple broadcasting. Alternative schemes that relax this invariant are an area for future work.

In addition, our algorithm maintains simplifying restrictions on the communication that occurs between two collectives.

First, at least one collective must be a *root collective* (the root node in some virtual hierarchy). Without this restriction, intra-collective connections would break the tree topology by introducing cycles. This can be seen by envisioning two unconnected trees. If a leaf node in one tree makes a connection to a leaf node in another tree, no cycle is introduced. But if their parents also connect, a cycle is formed. By forcing at least one node in the connection to be the root node, we alleviate the formation of cycles. If neither collective is a root collective, the protocol will not allow the connection.

In addition to breaking the topology, violating this restriction is a potential attack, whereby an attacker tries to join a number of collectives in an attempt to gather pieces of the private key privilege. As noted, one possible defense is to utilize the IBM 4758's

outbound authentication scheme as a basis for authentication. Other secure hardware solutions, such as the TCPA, may have similar solutions, but are perhaps more easily compromised than a FIPS 140-1 Level 4 validated device, such as the IBM 4758.

Second, nodes which collaborate in cryptographic computations for one collective may not collaborate for another. Allowing nodes to collaborate for two collectives simultaneously forces that node to hold two separate trust chains and breaks the hierarchical constraints.

We considered having the algorithm maintain a "balance invariant" on the hierarchy (e.g., each operation would maintain some balance property in the tree). However, this approach could result in large changes in the topology when a single node joins the network. We do make an assumption, however, that the nodes join and leave the network in a random fashion, resulting in a randomly built tree. It can be shown that randomly built trees have a height of $O(logV)$, and a worse case height of V. The possibilities of enforcing a balance invariant is an area for future work.

Cryptography In order to meet our claim of increased resilience, it is necessary for our scheme to require collaboration in order to perform cryptographic computations.

We take the approach that pieces of the private key privilege are scattered throughout the collective, as noted earlier. Many cryptographic techniques can enable this behavior, such as secret sharing and cooperative signature schemes [18]. These schemes are secure, but complex to implement. We discuss implications of some of these techniques in Section 3.4, and discuss Multi-Party RSA, as we plan to eventually implement it.

For simplicity, we considered the naive method of secret splitting [19]. When a request is made to sign something, the key is discovered by the host which received the request by broadcasting to the collective and ordering the pieces of the key. This is a *transient* operation in that the key is not stored at the host. Once the operation has been performed, the host forgets the key. It should be stated that this operation is *dangerous, as it exposes the assembled key*. In reality, assembling the key inside secure hardware—that could be trusted to protect it from malicious insiders—could provide some level of protection, as could using a better threshold scheme which does not require the key to be assembled at all.

3.3 The Algorithms

Our prototype maintains the invariants put forth in Section 1.2. The client and server actions guarantee the first invariant by eliminating cycles in the topology. We argue that the elimination of cycles is key to allow for efficient validation. The algorithms maintain the second invariant by enforcing that parties which negotiate a secret only store a fraction of the privilege. This implies that the secrets are distributed among members of the collective.

The server action is responsible for accepting connection requests, authenticating them, and deciding whether the two parties need to negotiate a secret. This decision is based on whether one of the parties has a portion of the private key privilege. The presence of such an entity in either party implies that at least one of the parties belongs to a collective and the other one is joining. The absence of this portion of the privilege

implies that a new secret must be negotiated, which in turn, means that a new collective is being formed.

The client action is called from an outside entity (i.e. user code), and is essentially making the same decision as above. The added burdens of avoiding cycles and enforcing assumptions about communication between collectives belongs to this action.

The validation procedure's sole responsibility is to determine whether some trust chain it receives is valid. This is done by traversing the list from the front (trust point) to the rear, and validating each node. The validation for any node is done by the Verify call. If Verify is successful for every node in the chain, then the validation procedure will return true.

Pseudocode for the algorithms can be found in the appendix of our preliminary report [20].

3.4 Analysis

In order to meet our claims of increased resilience and efficiency, we need to establish the following:

Structural Correctness The client and server actions maintain the negotiated secrets in a hierarchical, acyclic fashion. This is necessary to get $O(logV)$ average running times for the Validate procedure.

Privilege Distribution The functions maintain the property that the private key privilege and collaborating parties for each collective are distributed throughout the collective.

Structural Correctness The notion of structural correctness is used to show that the client and server actions maintain the private key privileges in a hierarchical, acyclic topology.

The hierarchy is maintained in two ways. First, the original two parties to connect form the root collective. As additional nodes join one of these two nodes, they are integrated into the collective as they are at a distance of one from one of parties maintaining the private key privilege for the collective.

As nodes make connections with collective members which are not maintaining the private key privilege, new collectives are formed. It is worth mentioning that a node which does not belong to a collective is the root of a virtual hierarchy which contains only itself.

Second, if there is a connection established between collectives, at least one of the collectives must be a root collective. If this were not the case, it would be possible for two leaf or internal collectives to join, resulting in every node in both trees to be reachable from two different trust roots, forming a cycle. This is exactly what we are trying to avoid, as this is the type of situation which leads to validation algorithms having to try multiple paths from an end point to a trust point.

The algorithms maintain a topology which avoids cycles. Each node in every collective maintains a *my_root* variable which is set to the root collective. This variable is

managed to always contain the node's root collective. As nodes attempt to make connections, they check this so as ensure that they do not attempt to make connections with nodes which already belong to the same tree.

Secret Distribution Secret distribution is the principle means by which we meet our claim of increased resilience. As noted earlier, threshold cryptography provides many tools. We discuss two.

Secret Splitting The scheme we implemented in our initial prototype is perhaps the simplest, and the weakest. The client and server actions distribute the keys across the collective in such a way that they can be correctly reassembled, and used to sign statements from the collective.

Splitting the private key into x pieces and re-assembling them when the collective needs to sign a statement does not invalidate the key. This technique is referred to as *Secret Splitting* [19], and for our prototype, we let $x = 2$.

One problem with this scheme is that we do not mandate redundancy of the key fragments. If Alice and Bob each hold a fragment and Alice has a power outage, the collective can no longer sign statements, at least until a new key can be established (which invalidates all the outstanding signed statements), or Alice powers up again.

The second problem with this scheme is that the key must be reassembled to be used. The only justification for this (albeit a weak one) is the fact that we have secure hardware. Without such machinery, this would totally expose the private key for a short time.

Secret sharing [11] would eliminate the first problem but not the second. Multi-Party RSA would eliminate both, which is why we plan to implement it in our next prototype.

Multi-Party RSA In opposition to a transient reassembling of the key and letting the result sign some statement, we envision a scheme which sends the statement around to each node holding a key fragment, and a portion of the signature being applied at that node. We plan to eventually use some instance of Multi-Party RSA to employ this technique in our system [18, 10].

Due to the algebraic properties of RSA, the algorithm lends itself to collaborative signature schemes quite naturally (an idea first proposed by Boyd [9]). Since then, the cryptographic community has generated a number of methods and protocols which utilize these properties. Samples of some such protocols and proofs of their security are discussed by Bellare and Sandhu [10] and Tsudik et. al. [21]. (This is by no means a complete list.)

Practically, using Multi-Party RSA in our system would allow a subset of members in the collective to possess key fragments, but would relax the assumption that they key is reassembled. The message is instead broadcast to the collective and comes back signed by the keyholders.

4 The Trusted Peer Access Layer

The TPA implements a protocol for trusted peers which allows them to communicate in a secure fashion. By secure, we mean that all parties mutually authenticate one another,

and that all traffic is encrypted by the secure coprocessor in such a way that an adversary could not discover the plain-text of the message — *not even if the adversary is the host* (i.e., the computer which houses the coprocessor). The protocol need only provide a decentralized means to locate items stored among those participating in the network (e.g. Gnutella) [22].

Loosely, the TPA Layer is a peer access layer running in secure hardware (the IBM 4758 Secure Coprocessor). The protocol is implemented across two communicating programs, one running on the host and the other residing in the card.

The host code is responsible for 1) implementing a command line interface which allows users (or other programs) to issue commands, 2) connection management between nodes over standard sockets, and 3) handing the TCP payloads to the card for processing and putting response packets from the card onto a socket.

The card code is where the protocol's packet processing logic resides, as well as the routing tables and secrets. The idea is that the card manufactures outgoing packets, encrypts them using secrets negotiated by it and another coprocessor in the network, and sends a chunk of ciphertext along with a socket number to the host so that it may place the ciphertext into a TCP payload and fire it to the intended recipient. When a packet arrives, the host program pulls the ciphertext out of the TCP packet and sends it to the card for processing.

The following is a brief discussion of the three major phases of development that drove our prototype implementation.

Peer-to-Peer. Our first task was to evaluate existing true peer-to-peer protocols that allowed for distributed location without the aid of a central server (like Napster). Gnutella was immediately appealing due to its simplicity, community, and availability of documentation and open source implementations.

It is important to understand what exactly Gnutella is and what it is not. Gnutella is a protocol and nothing more. In v0.4 (the base specification), Gnutella defines five packet types (called descriptors), a format for headers, and six rules for routing. Gnutella is only used to locate files across a network, transfers are done out of band (usually over HTTP).

However, Gnutella is not an implementation of this protocol. There are several implementations in existence, some of which add to the basic protocol, but they implement at least the core functionality described above [23].

We chose to use the core protocol as well as it seemed to fit our needs (actually, the "Push" descriptor type exceeds our needs, so we eliminated it), and could help reduce our time to prototype.

Secure Hardware. The next task was to find a fairly mature code base that implemented an open source Gnutella *servent* (SERVer + cliENT). Our constraints was that it should run on Linux, and be command line driven in order that we may pipe commands to it (something GUI based schemes lack).

We chose Gnut v0.4.25 because it met our requirements, was well documented, and professionally coded [24].

We then undertook the task of finding which pieces of Gnut stayed on the host and which went to the 4758. As stated above, the socket management code remained on the

host, and the packet logic and routing tables were ported to CP/Q++ (the native OS of the 4758).

At the end of this phase, we were able to observe 4758-enabled machines store strings and using the command line interface, were able to let other nodes locate them.

Adding Armor. In order to meet our definition of resilience, we had to implement a protocol for authentication and encryption, using the native cryptographic services provided by the 4758.

First, we consider authentication. The first element of our definition of resilience is that nodes must have a way to mutually authenticate one another. Bird et al. [25] explain that nonce based protocols are most secure, and since the 4758 provides a random number generator, we decided to go this way. We ended up implementing FIPS 196, which is essentially the core of most authentication schemes used in practice (e.g. Secure Sockets Layer) [26].

Second, we consider encryption. Once nodes have authenticated, the initiator sends four 3DES keys generated by its 4758 to be used for further encryption of all traffic between the two parties. Two of the keys are for encrypting messages and the other two are used for constructing a keyed Message Authentication Code for each message. We chose DES/3DES because it is fast.

5 Summary and Future Work

We are currently in the process of implementing the prototype, and once complete, we intend to make it available for public download. The TPA is lacking encryption support for all traffic. However, we do currently support authentication and are able to locate strings (which would represent cryptographic keys) across machines in the lab. The VHL is currently being implemented.

As it turns out, the result of this work has led to many more questions. In its current state, we plan to show proof of concept. As future work on this project progresses, we plan to address some of the questions that have been raised in order to evolve the system past being just a proof of concept. We are considering many ways to enhance the architecture.

One direction is to examine data structures other than trees. Balanced trees (e.g. AVL or Red-Black trees), and directed acyclic graphs could possibly lead to better solutions.

Another direction is to examine different routing protocols in the TPA. Specifically, reverse path forwarding or some other protocol which is a little smarter than just broadcasting could be interesting.

Our current architecture uses secret splitting, but as mentioned, we plan to extend the prototype to use Multi-Party RSA, allowing the message to travel around the collective to be operated on instead of the key being reassembled at one machine.

We plan to make much use of the virtual hierarchy technique in our current NSF-funded Marianas project [22], which explores using peer-to-peer techniques and secure hardware to build survivable trusted third parties.

References

1. Elley, Y., Anderson, A., Hanna, S., Mullan, S., Perlman, R., Proctor, S.: Building certification paths: Forward vs. reverse. In: Network and Distributed System Symposium Conference Proceedings. (2001)
2. Housley, R., Polk, T.: Planning for PKI. Wiley (2001)
3. Polk, T., Hastings, N.: Bridge certification authorities: Connecting b2b public key infrastructures. In: PKI Forum Meeting Proceedings. (2000)
4. Kaufman, C., Perlman, R., Speciner, M.: Chapter 15. In: Network Security - Private Communication in a Public World. 2nd edn. Prentice Hall (2002)
5. Ellison, C.: Improvements on conventional pki wisdom. In: 1st Annual PKI Research Workshop. (2002) 165–175
6. Hubaux, J., Buttyan, L., Capkun, S.: The quest for security in mobile ad hoc networks. In: Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing. (2001)
7. Zhou, L., Haas, Z.: Securing ad hoc networks. IEEE Network (1999) 24–30
8. Wilson, R.: Introduction to Graph Theory. Addison Wesley (1997)
9. Boyd, C.: Digital multisignatures. In: Cryptography and Coding. Oxford University Press (1989) 241–246
10. Bellare, M., Sandhu, R.: The security of a family of two-party rsa signature schemes. `citeseer.nj.nec.com/bellare01security.html` (2001)
11. Shamir, A.: How to share a secret. Communications of the ACM **22** (1979)
12. Smith, S., Weingart, S.: Building a high-performance, programmable secure coprocessor. Computer Networks **31** (1999) 831–860 Special Issue on Computer Network Security.
13. Anderson, R., Bond, M.: Api-level attacks on embedded systems. Computer (2001)
14. Clulow: (2002) Personal Communication.
15. Rabin, T.: A simplified approach to threshold and proactive rsa. In: CRYPTO. (1998)
16. Smith, S.: Outbound authentication for programmable secure coprocessors. In: 7th European Symposium on Research in Computer Science. (2002)
17. Tanenbaum, A.: Computer Networks. Third edn. Prentice Hall (1996)
18. Simmons, G.: An introduction to shared secret and/or shared control schemes and their application. Contemporary Cryptology: The Science of Information Integrity (1992) 615–630
19. Feistel, H.: Cryptographic coding for data-bank privacy. Technical Report RC 2827, IBM Research (1970)
20. Marchesini, J., Smith, S.: Virtual hierarchies - an architecture for building and maintaining efficient and resilient trust chains. Technical report, Dartmouth College (2002) Available at `www.cs.dartmouth.edu/tr/ncstrl.dartmouthcs/TR2002-416/`.
21. Boneh, D., Ding, X., Tsudik, G., Wong, C.: A method for fast revocation of public key certificates and security capabilities. In: 10th USENIX Security Symposium, USENIX (2001) 297–308
22. Nicol, D., Smith, S., Hawblitzel, C., Feustel, E., Marchesini, J., Yee, B.: Survivable trust for critical infrastructure. In: Internet2 Collaborative Computing in Higher Education: Peer-to-Peer and Beyond. (2002)
23. Clip2: The gnutella protocol specification v0.4. (`www.clip2.com`)
24. Pieper, J., Munafo, R.: Gnut documentation. (`www.gnutelliums.com`)
25. Bird, R., Gopal, I., Herzberg, A., Janson, P., Kutten, S., Molva, R., Yung, M.: Systematic design of a family of attack-resistant authentication protocols (1992)
26. U.S. Dept. of Commerce / National Institute of Standards and Technology: Entity Authentication Using Public Key Cryptography. (1997) FIPS PUB 196.

Local Networks and PKI

Sanna Kunnari[1], Tiina Seppänen[2]

[1] Ericsson Research
Sanna.Kunnari@ericsson.fi
[2] Ericsson R&D
Tiina.Seppanen@ericsson.fi

Abstract. Experience has shown that creating a global PKI for almost any purpose is a very demanding task. Therefore, there have been some proposals to postpone the extensive implementation of global PKIs. Instead, so called *local PKI* support could and perhaps should be provided first. One proposed method is to bootstrap a local PKI from the cellular infrastructure, leading to a solution where the current authentication and charging framework could be utilized locally.

Local PKI solutions have many advantages over a global one, the main benefit being ease of deployment. On the other hand, it suffers from a number of open issues, including problems in localization of service.

In this paper we describe our experiences with our local PKI prototype, which we have built on Linux platform. During the course of implementation we have noticed a number of practical problems, and eventually made us to question the concept of locality altogether.

1 Introduction

New business models need new security solutions to support them. Public Key Infrastructure (PKI) is suggested to provide a potential cohesive framework, within which business applications can be conducted with the trust they require [8]. However, as we discuss in this paper, the role and extend that PKI should take is far from clear.

A PKI is defined as "the set of hardware, software, people, policies and procedures needed to create, manage, store, distribute, and revoke public key certificates based on public-key cryptography" [12]. The current network security protocols rely on public-key cryptography to provide services such as confidentiality, data integrity, data origin authentication, and non-repudiation. From this point of view, the purpose of a PKI is to provide a framework for trusted and efficient key and certificate management, thereby enabling the use of authentication, non-repudiation, and confidentiality services [12].

The main technical function offered by PKI systems is distribution of public keys and establishing trust in the public keys. The user's public keys are typically stored in a database with associated subject identity and other information, each record is represented as a signed certificate. Certificates however, do not by themselves enhance the trust in the system, as a CA is still required to create the certificates and

thereby provide the necessary trust. A PKI enables certification and supports distribution of certificates.

During its lifetime, a certificate may be compromised either through carelessness by its owner, through system compromise, or via mandatory retirement of the certificate. Therefore, certificate revocation or other means of validity checking must be included in all PKI systems.

Revocation or other validity information must be publicly available, exactly as the certificates themselves are available. This information is typically represented as a Certificate Revocation List (CRL) [2], signed by the CA. In the list, each entry details a revoked certificate. Typical information contained in each list entry includes a certificate serial number, the revoker's identity, time and date of revocation, and a reason for revocation. As new revocations occur, entries are added to the list, and a new version of the list is published.

Other means for validity checking include online validation [8] and short-lived certificates [17]. However, we limit the discussion within this paper to CRL based revocation.

1.1 Market for Authorization and Charging

With current systems, it is technically feasible to offer mobile users many types of new services. In typical wide scale IP-based systems, RADIUS or DIAMETER based Authentication, Authorization and Accounting (AAA) servers and protocols typically take care of entity authentication and authorization [13]. However, there is a lack of a large-scale infrastructure to authorize and charge users for these new services. By combining PKI and charging systems it might be possible to create such an infrastructure.

Some market analyses indicate that the global PKI technology has failed to take off [1]. On the other hand, the business needs of the customers drive for a functional PKI based infrastructure [11]. In practice, companies do use public key technologies in web-based businesses to ensure that vital customer and corporate information is kept integral and confidential. Therefore, well-functioning PKI products are business opportunities for competent manufacturers. Additionally, current situation seems to be that customers have desires for small specific solutions [18].

PKI in a global scale has been a phenomenon that has not yet really appeared [1, 18], although, many applications and protocols expect the services of it. Experience has shown that creation of a global PKI for almost any purpose is a very demanding task. The only widely used example of such a PKI is the one used to secure TLS based Web transactions [8], and that is of relatively low security level. The main reason for the security weaknesses can be found in the client end, where both the multitude of default CA certificates and the arcane user interface impose security threats. As a consequence, there have been some proposals to postpone the extensive implementation of global PKIs.

Instead, so called *local PKI* support could and perhaps should be provided first. A global PKI may be unreachable as the first approach. One proposed solution is to bootstrap a local PKI from the cellular infrastructure, leading to a solution where the current cellular based authentication and charging framework could be utilized [10].

1.2 Paper Organization

In this paper we aim to generalize the originally cellular centralized local PKI ideas into a more generic framework. The original thoughts were developed within 3GPP [10], but we see that the concept is applicable for other networks as well. We have identified several open issues with the concept and these issues are being addressed in this paper. Additionally, we have identified and examined some (key management) scenarios from the concept identified and examined some certificate delivery scenarios.

The rest of this paper is organized as follows. Next, in Section 2, we introduce the concept of local PKI. Section 3 describes some related research. In Section 4 we illustrate the functions of a typical local PKI, and in Section 5 we describe our prototype. In Section 6 we describe our experiences with the prototype, identifying a number of future research topics. Finally, Section 7 contains our conclusions from this research.

2 Local networks and PKI

2.1 Background

As global PKIs have been found to be inapplicable in the (near) future, the research community is going towards local aspects. People are focusing on small, specific cases and network areas, environments that can be easily defined. The concept of local PKI includes the need to define what a local network is. A definition of network boundaries is needed. In the cellular network case the answer can be derived from the operator network boundaries. In the Internet case a local PKI could refer to Intranets, personal area networks (PAN), or application domains. However, the purpose is not to imply a local PKI where a CA is controlled by the same organization as those supplying the majority of PKI users. In principle, the local CA is any available (local) trusted third party that the user and/or his organization trust.

Certificate retrieval is an important issue in PKI systems, as verification of certificates requires public key decryption, which can be a potentially high-cost operation. It is therefore desirable to optimize the certificate retrieval process to reduce waiting times and overheads. Retrieval of a certificate from within the local domain is a straightforward operation, which makes local domains and networks very appealing.

2.2 Local Networks and Domains

Public-key infrastructures have been an important development for addressing the security concerns of network applications. PKIs provide a facility to partition the world into localized security domains [5]. Domains are typically localized within organization boundaries, encompassing the trust region of that organization only.

Domains may contain from one to many hosts and many clients, though the purpose of domains is to localize a manageable number of clients and machines within the scope of a security administrator. The simplest PKI system is comprised of only a single domain within which all entities exist.

Splitting the world into domains alleviates some problems, but creates others. First problem is how does one domain establish trust with other domains. Other problem involves the practical communication problems between the domains, e.g., ones caused by intermittent connectivity. It is difficult ensure that certificates are always passed around between domains as needed.

These problems can be resolved by addressing the trust routing problem; a domain needs to be able to establish trust, via cross-certification, with other domains. Each domain may opt to use its own ad hoc routing method or a global hierarchy can be established within which every domain resides, cross-certified with at least one other domain [5].

Cross-certification performs two essential functions within the PKI domain hierarchy [5]. Firstly, it propagates trust between domains; thereby enabling inter-domain communications secured using PKI services. Secondly, it enables shortcuts through the domain hierarchy, speeding up certificate retrieval between any two domains by reducing hop-distance between the two domains through the hierarchy to a single hop. Cross-certification occurs in much the same manner as client certification, except that there must be higher assurance of authenticity of each party.

The case of inter-domain certificate and CRL retrieval is more complicated and depends heavily on the domain interconnection strategy adopted within the PKI. To retrieve a certificate from a remote domain requires that the CAs of the local and remote domains are cross-certified or there is a path of cross-certified domains linking the local and the CA of the remote domain.

Retrieving a certificate from a remote domain yields a chain of certificates corresponding to the path of cross-certification of CAs and/or domains through the hierarchy and the desired certificate. Validating the certificate chain requires that each certificate's digital signature is verified with the public key of the issuing CA. It is obvious that certificate chains need to be kept as short as possible to ensure timely retrieval.

2.3 Defining Local PKI

The CA is the trusted third party within the system by which all trust is propagated. It is a highly trusted and secured piece of software and must reside within the trusted computing base of the installation to maintain the trust in the local PKI domain and overall PKI hierarchy

The basic idea of local PKI is that it is defined in a specific, rather bordered network. There is a local CA that can assign and delete certificates for local members or visitors. The local CA is a trusted by local service providers and users (and/or the organization behind the user). However, the local CA is not tied to or controlled by the user’s organization. There may not be vital needs for external trusted third parties or other communications. Advantages of the concept are discussed later in Section 4.

The basic concept is presented in Figure 1. Each local network has one or more CAs. There may or may not be some service providers, depending on the network service level. A user may be at his/her home network or then alternatively visiting another network.

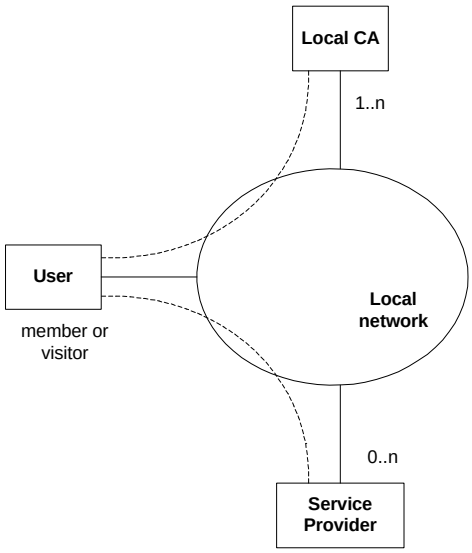

Fig. 1. Local PKI in general view

There can be three kinds of certificates: authorization certificates that can be used for many purposes, e.g. to denote performed purchases of services, attribute certificates, and identity certificates [10]. In most current systems identity certificates are eventually used for access control, even though authorization certificates often provide a better solution.

The lifetime of authorization certificates is expected to match the validity period of the authorization, e.g., to follow the subscription period purchased. The concept could advantage short-life certificates because they will not usually need to be revoked and will not need to be include on the certification revocation lists (CRL).

3 Research on the Area

3.1 Related Techniques and Utilization

There are already activities in the utilization of local network services, such as utilizing local CA services. It appears to be that China is actively utilizing the concept of local PKI, as they distribute the services to local fixed telecommunication networks [4].

The Federal Public Key Infrastructure Technical Working Group (FPKITWG) has developed the Bridge Certification Authority (BCA) concept to provide certificate chains to link enterprise Public Key Infrastructures within the Federal government, and to provide trust chains between the Federal Public Key Infrastructure and those of external organizations [3]. However, the existence of certificate chains among infrastructures does not, by itself, provide the ability for subscribers of different PKIs to communicate securely. In particular, the BCA concept as defined to date does not provide a mechanism for making certificates or revocation information generated in any given public key infrastructure domain available to relying parties in other public key infrastructure domains. The research is trying to overcome these problems with concepts like the Border Directory System Agents (BDSAs).

There is also analysis on short-lived certificates. It suggests that using certificates with shorter validity can simplify the security framework [17]. To free a client from key ownership responsibility, there should be no possible association between the client and the PKC key pairs. For example, any short-lived certificate issued to the client should not be taken from a small pool of key pairs. Short-lived certificates demand minimal key management and protection, since they expire soon after their expected usage. The authors claim that they can develop an infrastructure that can deliver certificates both efficiently and securely. Their idea is based two separate tasks: entity registration and certification.

The credentials used in a PKI typically consist of a public and private key pair, a corresponding certificate or certificate chain and some trust or root CA information [20]. They are usually stored on a desktop or laptop system as part of an application specific store. Currently, users need to get too involved with the mechanics of creating and maintaining their PKI credentials. Application specific stores mean that users can not easily use the same credential in multiple applications or on multiple devices. In effect, today, credentials are not portable. Ideally, users would be able to use a common set of credentials with their Internet-ready devices.

A distributed terminal consists of several components within physical proximity to each other and the user or users. They are interconnected with local communication links like short-range wireless connections, e.g. Bluetooth. This type of personal local network used to be called a Personal Area Network (PAN). There are publications on the area that has a goal to provide security architecture applicable to the PAN reference model [22, 23]. Naturally, authentication and key exchange using a certificate demand the certificate to be signed by a common trusted third party. In a personal environment this can be achieved by letting the owner issue certificates to all his components using a particular *CA device*. The authors call such a device a *personal CA device*. A personal CA device might be a mobile phone, PDA or laptop that fulfils the security requirements for issuing and signing certificates. The personal CA concept requires a *personal PKI* that must satisfy several security requirements. An ordinary user for home or small office deployment uses the personal CA. [22, 23]

3.2 Related Protocol Suggestions

The related work within the local PKI area includes protocol proposals like AKA [6], AAA credentials [19, 21], SPKI [14], EAP AKA [6] and EAP SIM Authentication

[7]. These are mainly tools that can be used when building local PKI solutions but these might be considered as competing approaches as well. There have also been ideas about combining small networks together using (extended) cross-certification [5, 18]. Hybrid PKI model has also some interesting thought [9].

Authentication and Key Agreement (AKA) protocol is a secure protocol developed for authentication and key management in 3G-network [6]. It is a challenge response protocol, which uses Authentication Center from the home network to derive the challenge. In 3G cases, the local PKI uses AKA to gain an authenticated channel for certificate negotiation [16].

In Authentication, Authorization, and Accounting (AAA) protocol authentication involves validating the identity of a user prior to permitting him network access. This process keys on the notion that the end-user possesses a unique piece of information that serves as unambiguous identification credentials. Such information could be a username and password combination, a secret key, or perhaps biometric data. The AAA server compares the user-supplied authentication data with the user-associated data stored in its database, and if the credentials match, the user is granted network access. [21]

Extensible Authentication Protocol (EAP) mechanism has been applied also for authentication and session key generation using the GSM Subscriber Identity Module (SIM) [7]. This approach is called EAP SIM Authentication. Basically, it re-uses the GSM authentication and provides optional support for protecting the privacy of subscriber identity.

Simple Public Key Infrastructure (SPKI) has been developed as a more flexible alternative to X.509 [14]. There can be both identity certificates and authorization certificates. The name certificates bind names to the rights transmitted by the certificates. The authorization certificates bind the cryptographic keys to the transmitted authorities. Basic idea is that anyone with access to a resource can authorize others to use the resource by issuing him or her an authorization certificate. SPKI certificates can be used to implement systems that support anonymity, delegation and dynamic distributed policy management. [15]

Hybrid PKI model can be used for specifying and enforcing permissions in distributed computing environment [9]. The research deals with approaches to specify and to enforce permissions of clients on remote servers, which are based on public key cryptography. In various forms, trust has to be assigned to public keys and to appropriate attributes that are claimed to be true for a public key or the holder of the corresponding key. Management of trust is organized within a PKI. Secure mediation is taken as an example for applications that require security policies for confidentiality and integrity.

3.3 Example: Solution for 3GPP

The 3GPP AKA results in an integrity key between the user and the serving network [6]. This authenticated channel could be used submit the users public signature verification key and obtain a temporary certificate issued by the serving network. To ensure that the correct private key is being certified, the public key submission or certificate request should be signed by the users private signing key. The user can use

the signing key to sign the service requests during access via the local network. A service provider who knows the signature verification key of the local serving network can verify the users certificate and signature, and use it as an authorization for services. The certificate covers single or many purchases during the validity period. For IMSI-based access control, there can be three kinds of certificates: for purchases, attribute certificates and identification certificates. The certificate for purchases is a specialization of an authorization certificate, not a new certificate type. [10]

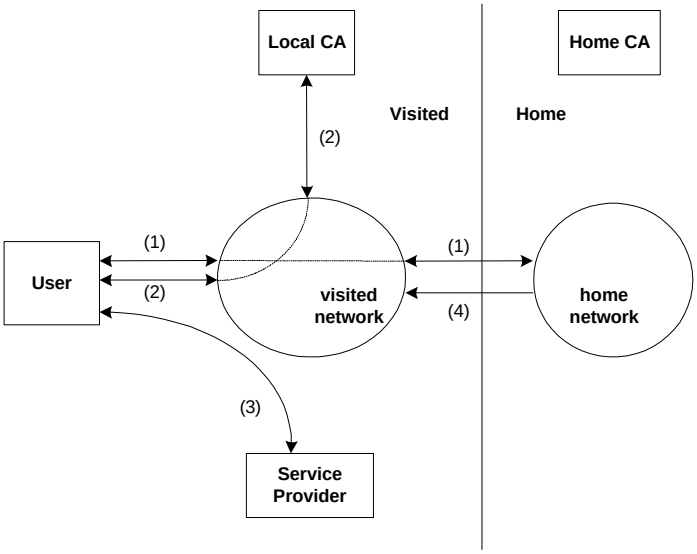

Fig. 2. Solution for local PKI in 3G networks

Figure 2 presents high-level view of the communication lines between the network nodes. Line (1) presents the authentication and charging at home network using 3GPP framework. The result is the integrity key and authenticated channel. Within line (2) user generates a *Certificate request* (new message) with a new public key to visited network. There the user generates a temporary public/private key pair and sends a certificate request. The visited network forwards the request to its local CA. Then the local CA generates and signs a temporary certificate for the user. Next the user receives the certificate in a *Certificate response* (new message).

As illustrated by line (3), the user can now use its temporary private key to sign a service request to a local service provider. Additionally, the service provider can verify the signature, because the local CA signs the new public key of the user. It is assumed that the local service provider possesses visited networks public key (e.g. via local CA). Optionally in the line (4), a hash of user's long-term public key is sent from home to visited network.

4 Local PKI Illustrated

4.1 Certificate Creation and Revocation Processes

The following two scenarios of key management can be used in the definition of a local PKI: user sends keys (pull) or CA generates keys (push). These two scenarios present different requirements for the certificate creation process.

In the first case, user generates keys and delivers the public key within a certificate request. >From the user's side of view, it is a *pull* operation since he is the active participant in the certification process. It is essential that the keys be delivered confidentially to the CA. There are risks of fraud and error during the transfer, so the public key should be at least integrity protected. It is not obligatory to encrypt the key since it is public information. Proper solution for the integrity protection might consist of user signing the public key with the responsive CA's public key. In a successful case, the CA reply contains a granted certificate, which should also be integrity protected. Now the CA may use its private key for the creation of digital signature.

The second case uses CA to generate keys. User may first send a certificate request for the CA, but depending on the local security policies, the request might be optional. The key generation operations and responsibilities are on the CA and the certificate delivery can be seen as a *push* style for the user. This case might be suitable for mobile networks or for less computationally efficient use devices. The problem is how to deliver securely these keys for the user. In some network cases, the user could approach the CA and receive the keys manually. If the case is a visited network, the authentication mechanism needs to be solved first. In mobile networks, it is possible to take advantage of the shared secret between a user and an operator by encrypting the user's keys by this shared secret. Because the shared secret is only known between these two parties, the solution is not applicable to roaming situations without advanced trust relationships between operators.

Certificate revocation may occur by a user request, by a CA or by an expired lifetime of a certificate. If a CA decides to revoke a certificate, it can be considered as a push operation from the CA. If a user requests for certificate revocation, he pulls a revocation operation from the CA. If a user is requesting revocation of its own certificate, he must digitally sign the request.

A simple and suitable solution for certificate revocation within local PKI would be to use short-lived certificates. They may diminish both the verification and revocation burden since the certificates are not valid for very long. Short-lived certificates might be applicable for one-time use and for short sessions.

4.2 Advantages of the Solution

There are many advantages in the local PKI approach and some of these advantages are discussed here yet comprehensive analysis is left for future work.

Local PKI supports re-use of network authentication mechanisms. It is an asset that current mechanisms can be utilized and there is no need to develop own

solutions. That means shorter time-to-market times for local PKI solutions. If local PKI is the only thing that is needed for service request authentication, the solution is also very efficient.

Local PKI does not require any per-user configuration in customer databases before a user is requesting to access new services. This is essential for mobile networks, since the subscriber database operations may be rather complex.

One of the main ideas of the concept is that local PKI does not require or mandate trusting external entities. That makes deployment easier and lessens the trust establishment burden between the networks.

In addition, the concept does not prohibit global relationships and possible future expansions. There are methods for chaining local network and PKI areas together and establishing connections between them. Local PKIs can exist and operate in a closed manner but still maintain their external connections. For example, this enables the possibility to utilize long-term public keys from CA that is located in home network, if required.

4.3 Open Issues

There appear to be many open issues with the proposals and they need to be solved in order to get local PKI applicable one day. Many of these issues involve practical problems and we try to give them some solution proposals.

Users who visit other network face interesting challenges. As their first task, users have to acquire information about the local security policy. In order to require a local certificate, user needs to locate the correct CA. Roaming situations are typical examples of these networking visits.

As user enters a new network, he has to perform authentication with it. In mobile network cases, operators have established roaming agreements with each other, which enables users to trust the visited operator. In the same context, authentication material is most likely retrieved from the user's home network. In general, if user requires mutual authentication with visited networks; he has to have some pre-shared authentication material for the network and vice versa.

Local PKI concept may be tied to a certain local CA but there might be more than one CA within the network. However, this requires establishing CA hierarchies and may lead to inefficient solutions. Therefore, it might be convenient to limit the number of CAs within a local PKI target area. Another open issue is if operators can share a local CA within some geographical area. If a network has *outsourced* its CA, it may be difficult to control the configuration of it.

Service providers are considered here as local, but it is possible that they would serve multiple network areas. There are probably not enough business possibilities for service providers if they limit their supply on one network area. This raises a question if a service provider is truly tied to a specific local CA.

In end-to-end communications, it would be practical to know if the other party is in the same local network. Localization services should be applied to such situations.

Interoperability and migration to global PKI are issues that need further analysis. Local PKI might be considered as a first aid to the lack of global PKI but eventually they should incorporate. There are techniques that can be used to chain separate local

PKI networks together, which is one possible way to develop. However, it also is possible to keep local PKI as it is, as a convenient and efficient solution for small network areas.

5 Prototype

We have implemented a prototype of local PKI on Linux platform. The prototype is a combination of our own code and additional software components. It is not trying to fulfill all the aspects of local PKI. Rather, it is trying to evaluate how difficult it is to implement such solutions.

The network concept is derived from the general case with some interactions to the 3GPP case. Our scenario assumes that a user is entering a visited network and home network is not available. The visited network is of Intranet type and all data traffic goes through firewalls. In the prototype, user generates key material for itself, which is a more practical solution for the communications.

Figure 3 presents the network picture for the prototype. The first connection performs authentication, the second requests services from a service provider. In the third connection, user negotiates a certificate from the local CA. In the fourth connection, the service provider checks the certificate that the user has provided.

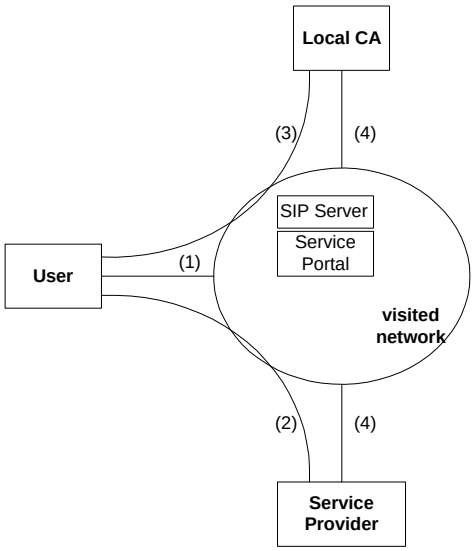

Fig. 3. Prototype view of a local PKI

Figure 4 goes to the next level of details. It presents a sequence diagram of a scenario, where a user connects to a visited network and requests for services. Additionally, it presents architecture blocks of the prototype, because it assorts the main functional elements.

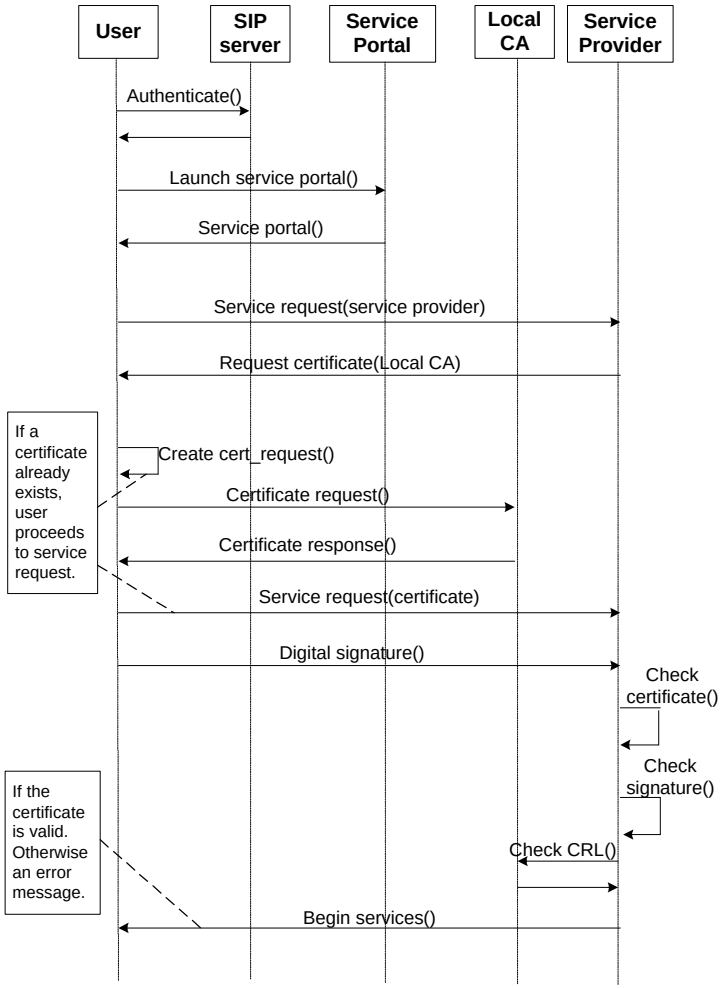

Fig. 4. User is requesting services

As user arrives to a local PKI network, he performs authentication to a local SIP (Session Initiation Protocol) [24] server. We are using SIP here, although AKA (Authentication and Key Agreement) would have been a better choice. Unfortunately, we did not have the AKA implementation available in time. The SIP server is the first contact point to this (visited) network. If the authentication is successful, the SIP server delivers the address of the service portal to the user. This message is *virtual* and only presented to keep the model coherent. The service portal is also a rather virtual node in the network but essential for our prototype. User selects a service from the portal, which results in an initialization message for the service provider. Next, the service provider asks the user for a (local) valid certificate. If the user does not have one, he contacts the local CA. The contact information was in the service provider's

message. Then user may continue by creating a certificate request and sending it to the local CA.

As soon as the certificate is available, user sends it to the service provider in a service request. For identification purposes, the user generates also a digital signature of the service request and sends it as another message.

The service provider must check the digital signature and the certificate before it allows the user to access its services. Because the digital signature was created by user's private key, it can be verified by the user's public key. Basic information, as CA information and validity intervals, are verifiable by the service provider itself but it uses also CRLs.

6 Experiences from the Prototype

Implementation has raised many practical problems and questioned the concept of locality. The problems have focused on roaming, interoperability, authentication and service discovery.

The prototype was not built into mobile network, which made it impossible to test some features. However, for the prototype implementation, it does not matter, if the user is mobile or not. Because the prototype is running on an Intranet network, the prototype reflects better to common network scenarios.

Original idea was to use AKA to create an authenticated channel for other communications. However, we could not test AKA authentication in the prototype. Instead, we used SIP digest authentication [24], which provided us some new challenges. Digest authentication requires some pre-shared secret in order to accomplish authentication procedures between server and user. In the prototype, it was easy to insert another parameter, but this might be a problem for real-life situation.

The CA product in use requires also a pre-shared secret between a user and a CA, which might be a challenge for the concept. These pre-shared secrets might require appropriate specification in the local PKI concept. In that case, a decision is needed about what information should be used as such information. It might be possible to use some user information at home networks.

One problem with the implementation was how the user discovers the local CA and service providers. If the user is a visitor, it can not be assumed that he would know the local infrastructure in advance. Therefore, we presented the service portal, which includes links to the pages of the local service providers and that way the user can easily choose the services he wants to utilize. The CA contact information is delivered to the user within the first contact message from the service provider. Otherwise, the user might not know where to get his certificate.

The implementation experiences have shown that the concept of local does not differ that significantly from global with PKI networks. It is difficult to implement closed environments, which would not have any communication to outside world. The experiences with SIP and CA illustrated that connection to home network might be needed. Therefore, despite of original purpose, in some cases the local PKI concept evidently extends to global by cross-certification, certificate chains etc.

7 Conclusions

The PKI in a global scale is a phenomenon that has not yet really appeared [1, 18], and many applications and protocols are still waiting for the services of it. Experiences have shown that PKI deployment in general is a very demanding task. However, one widely used example is the one used to secure TLS based Web transactions [8].

It seems that local solution and especially local PKI is needed in the markets, indeed. However, it is questionable how the concept of local is actually defined and fulfilled. We have continued the definition of the local PKI to make it correspond to a more generic network situation.

There are many advantages in the local PKI proposal and we have gathered some of them here. The advantages include: Reuse of network authentication mechanism, it requires only local PKI for service request authentication, efficiency, it does not require any per-user configuration in subscriber databases before a user is allowed to access new services, and it does not require trusting external entities and can enable non-repudiation. In addition, it makes possible the utilization of long-term public keys, if required.

We discovered also some of the open issues with the solution and tried to give them solution alternatives. The open issues include: Roaming situations, localization services, the binding of a certain CA to a local PKI, the binding of a CA to service providers, the possibility to share a CA within a network or geographical area, local CA discovery, CA configuration and management, and interoperability and migration to global PKI.

The prototype was implemented in order to demonstrate how difficult it is to create and use the local PKI concept. The prototype has gained us many experiences that are discussed here as well. The implementation problems have focused mainly on roaming, interoperability, authentication and service discovery.

Our implementation experiences have shown that the concept of local may not differ very significantly from global within PKI networks. Any network implementation tends to require communication to outside world. In network visits, the experiences with authentication illustrated that connection to home network might be needed. Therefore, sometimes the concept of local PKI eventually extends to global by cross-certification, certificate chains etc.

Acknowledgments

We would like to thank Pekka Nikander, who encouraged us to work on this paper and who helped us to shape up the paper to its present form.

References

1. Global PKI Markets 2001-2003, http://www.datamonitor.com/~121f97570fd84a3697a928c1d13bac11~/technology/reports/product_summary.asp?pid=DMTC0617

2. Menezes A.J., van Oorschot P.C., Vanstone S.A.: Handbook of Applied Cryptography, CRC Press LLC, 1997
3. Fillingham D.: Federal Bridge Certification Authority (BCA) Border Directory System Proposal, http://csrc.nist.gov/pki/twg/papers/twg-99-03.pdf
4. Update1 on Regulatory and Policy Developments, www.apectel25.org.vn/documents/plen03.doc
5. Design Issues in a Public Key Infrastructure (PKI), http://www.csu.edu.au/special/auugwww96/proceedings/barmoroco/barmoroco.html
6. Arkko J., Haverinen H.: EAP AKA authentication, IETF, http://search.ietf.org/internet-drafts/draft-arkko-pppext-eap-aka-03.txt
7. Haverinen H.: EAP SIM Authentication, IETF, http://www.ietf.org/internet-drafts/draft-haverinen-pppext-eap-sim-03.txt
8. PKI Forum, http://www.pkiforum.org/resources.html
9. Joachim Biskup, Yücel Karabulut: A Hybrid PKI Model with an Application for Secure Mediation, 2002, http://ls6-www.informatik.uni-dortmund.de/~karabulu/researchinterests.html
10. 3GPP TSG SA WG3 Security – S3#19, S3-010353
11. PKI Market Summary, http://www.nss.co.uk/Articles/PKI%20Market%20Survey.htm
12. Arsenault A., Turner S.: Internet X.509 Public Key Infrastructure: Roadmap, IETF, http://www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-07.txt
13. Gehrmann C., Horn G., Jefferies N., Mitchell C.: Securing Access to Mobile Networks beyond 3G, http://www.mobilesummit2001.org/mcs2001/papers/MOBCS4VYJM6.pdf
14. Yki Kortesniemi: Validity management in SPKI, 1st Annual PKI Research Workshop, 2002, http://www.cs.dartmouth.edu/~pki02/
15. Koponen J. P. T., Nikander P., Paajarvi J., Rasanen J.: Internet Access through WLAN with XML encoded SPKI certificates, Nordsec'2000, Reykjavik, Iceland, pp. 239-247
16. 3GPP TSG SA WG3 Security: Draft Report of SA WG3 Meeting #19
17. Y.K. Hsu, S. P. Seymour: An Intranet Security Framework Based on Short-Lived Certificates, IEEE INTERNET COMPUTING, IEEE, Vol. 2, No. 2; MARCH-APRIL 1998, pp. 73-79
18. Stephen Kent: Rethinking PKI: What's Trust Got to do with It?, Eurocrypt 2002, http://www.ec2002.tue.nl/
19. IETF AAA Working Group, http://www.ietf.org/html.charters/aaa-charter.html
20. IETF Securely Available Credentials (sacred) Working Group, http://www.ietf.org/html.charters/sacred-charter.html
21. Christopher Metz: AAA protocols, IC Online, http://www.computer.org/internet/v3n6/w6onwire.htm
22. Christian Gehrmann, Thomas Kuhn, Kaisa Nyberg, Peter Windirsch: Trust model, communication and configuration security for Personal Area Networks, IST Mobile Summit, June 2002
23. Christian Gehrmann, Kaisa Nyberg, Chris J. Mitchell: The personal CA – PKI for a Personal Area Network, IST Mobile Summit, June 2002
24. Session Initiation Protocol Working Group (IETF), http://www.ietf.org/html.charters/sip-charter.html

A Prototype Tool for JavaCard Firewall Analysis

René Rydhof Hansen*

Informatics and Mathematical Modelling, Technical University of Denmark,
DK-2800 Kongens Lyngby, Denmark
E-mail: rrh@imm.dtu.dk

Abstract. JavaCard is a variant of the Java programming language specifically designed for use on Smart Cards. In order to support the secure execution of several different applets on a Smart Card, the JavaCard Virtual Machine implements a *firewall* that isolates applets from each other by preventing unwanted information sharing and communication between applets.
In this paper we report on a prototype tool that can statically verify that a JavaCard applet does not (try to) violate the firewall rules. Such a tool is useful for increasing confidence in the security of an applet. Furthermore, a developer can use the tool for guaranteeing in advance, ie. before the applet is deployed, that it will not throw a security exception at an inopportune moment, thus leading to more robust and user-friendly applets.

Keywords: JavaCard, firewall, static analysis, prototype tool.

1 Introduction

The JavaCard platform is a multi-applet platform, meaning that a given JavaCard may contain and execute several different applets from several different, possibly competing, vendors. In order for an applet to protect sensitive data from other, malicious, applets the JavaCard platform implements a firewall to separate applets from each other.

The firewall mediates all method invocations and field accesses between applets and determines whether or not to allow it. If a given method invocation or field access is not allowed a *security exception* is thrown. Note that security exceptions can not be caught by the program itself, but is rather communicated all the way back to the card access terminal, ie. the user.

In this paper we discuss the formal development of a prototype tool, based on well-known static analysis techniques, for verifying that a JavaCard program does not (try to) violate the JavaCard firewall. Such a tool is useful for increasing confidence in a programs behaviour (by ensuring that a program does not try to violate the security policy) and also for enhancing the robustness of a program (by guaranteeing that no security exceptions are thrown).

* This work was partially funded by the SecSafe project (EU IST-1999-29075).

Furthermore the developments in this paper are intended to show that the Flow Logic framework and methodology used for specifying and extending the analysis is well suited for adding functionality in a structured fashion in step with increasing demand for guarantees that a program has (or lacks) certain properties, thereby reducing the workload when designing analyses for such validation.

The rest of the paper is structured as follows. Section 2 discusses the language used, Section 3 gives a brief overview of the JavaCard firewall, following that Section 4 specifies a so-called *ownership analysis* that will conservatively approximate the set of possible owner contexts an object can have. The result of this analysis will form the basis for verifying programs as shown in Section 5. Section 6 then shows how the analysis and validation can be implemented by constraint generation and solving. The paper concludes with an example in Section 7 and conclusions and future work in Section 8.

2 Introducing Carmel

The JavaCard language contains more than 100 instructions, a significant number of which are very specialised, eg. the instruction `push0` pushes the integer constant 0 on top of the stack. Such highly specialised instructions are mainly used for optimisation purposes and are not, as such, essential features of the language.

The large number of instructions makes JavaCard rather unwieldy and work-intensive for formal treatment. Therefore we have decided to base our work upon a JavaCard derivative, called Carmel, developed as part of the SecSafe EU project, cf. [13].

Carmel is directly derived from JavaCard, mainly by removing non-essential instructions and adding more generality to the remaining instructions ending up with a language consisting of only 30 instructions while retaining all the power, flexibility and expressibility of JavaCard but in a more manageable form. Thus Carmel can be seen as a "rational reconstruction" of JavaCard. In this paper we do not consider the instructions for subroutines.

In this section we briefly, and informally, review the Carmel language. For a formal definition of the Carmel language, including an operational semantics, see [14, 15].

2.1 The Carmel Language

In the following, square brackets are used to denote [*optional*] arguments to a given instruction. Many instructions are explicitly annotated with the type of their argument and/or result. For the purpose of this paper, such annotations are ignored.

The instructions for basic stack manipulation in Carmel are as follows:

`push` t c	push constant c
`pop` n	pop the top n values
`dup` n d	duplicate the n top values at depth d
`swap` m n	swap the top m values with the following n values

All arithmetic and boolean operators are combined into one instruction parameterised on the particular operation to perform. Operators pop their argument(s) from the stack and push the result:

`numop` t *op* $[t']$	use operator *op*

For control flow Carmel has the following instructions

`goto` L	unconditional jump
`if` t *cmpOp* [*nullCmp*] `goto` L	conditional
`lookupswitch` t $(k_i$`=>`$L_i)_1^n$, `default=>`L_0	branch on key
`tableswitch` t l`=>`$(L_i)_1^n$, `default=>`L_0	branch on index

The conditional usually compares the two top elements on the stack (using *cmpOp*); optionally it compares the top element to null. The `lookupswitch` and `tableswitch` instructions are convenient for case constructs.

Methods in the Carmel language can access and modify local variables:

`load` t x	retrieve the value of x
`store` t x	store a new value in x
`inc` t x c	add c to the value in x

The instructions for manipulating objects are as follows:

`new` σ	create a new instance of class σ
`getstatic` f	get the value of the static field f
`putstatic` f	store a value in the static field f
`getfield [this]` f	get the value of field f
`putfield [this]` f	store a value in field f

The instructions `getfield` and `putfield` look for a reference to the object whose field is being accessed, on top of the stack. The optional "`this`"-modifier indicates that rather than looking for such a reference on top of the stack, the current object (the one in which the `getfield` or `putfield` instruction is executed) should be used.

Two instructions are provided for dynamically checking that an object is of a certain type:

`checkcast` σ	check if an object is a subclass of σ
`instanceof` σ	check if an object is a subclass of σ

The above two instructions differ only in their return value: `checkcast` throws an exception if the object is not of the specified type, whereas `instanceof` simply returns 0 in that case.

The instructions below cover the various forms of method invocation:

`invokevirtual` m	invoke a virtual method
`invokestatic` m	invoke a static (class) method
`invokespecial` m	invoke class initialisation method
`invokeinterface` m	invoke an interface
`return`	return from a method invocation

Static methods are class methods and can be invoked directly from a class as opposed to virtual methods that require a class instance (an object).

Certain static information not directly available in the Carmel syntax is accessed through special auxiliary functions, eg. to obtain the return type of a method m we can use the function returnType(m) which, following tradition, we write as m.returnType.

JavaCard, and thus Carmel, programs rely heavily on the use of arrays, both for storage and communication. The instructions supporting arrays follow

`new (array` t`)`	create new array
`arraylength`	return length of array
`arrayload` t	load a value from an array
`arraystore` t	store a value in an array

Finally exceptions can be thrown using the `throw` instruction:

`throw`	throw an exception

Handlers for exceptions are resolved at a higher level and thus have no direct representation in Carmel (or JavaCard).

In addition to the instructions outlined above, a typical Carmel program will use a number of higher-level library functions, eg. for copying the contents of an array or creating cryptographic keys, and standard APIs, eg. for communicating with a terminal. In the present paper we do not model the libraries and APIs.

3 The JavaCard Firewall

Smart Cards are often used to store sensitive information, such as cryptographic keys and personal information, and for this reason it is important that applets are protected against malicious access to its sensitive data. The JavaCard platform implements a firewall to separate applets from each other and to make sure that no unwanted access to an applets data is allowed.

The firewall policy is based on the notion of *contexts*: applets belonging to different contexts are not allowed to access each others data, neither fields nor methods, with a few exceptions discussed below.

JavaCards package structure forms the basis for contexts: two applets that are instances of classes from the same package are assigned the same context. Additionally a "system" context is defined by the JavaCard Runtime Environment (JCRE), and applets belonging to the JCRE context may access applets in all other contexts without the firewall intervening.

During execution of an applet, objects that are created are assigned an *owner context* based on the context of the applet that created it. A method executes in the context of its owner, with the exception of static methods that are executed in the context of the invoker.

Certain specialised applets may wish to share some data with another applet in a different context, eg. a wallet applet may wish to share some data with a so-called loyalty applet in order to award "bonus-points" for purchases made with the wallet. For such situations JavaCard defines two ways in which the firewall can be circumvented in a controlled manner: JCRE entry points and sharable objects. The JCRE entry points are objects owned by the JCRE specifically designed to be accessed from all other contexts. The primary example of such an entry point being the APDU, through which all communication outside the card is handled. Sharable objects can be used to grant limited access to an objects methods (not the fields) across contexts.

It should be noted that the above mechanisms merely allow for data to cross firewall boundaries, it is still the responsibility of the applet wishing to share data that it properly authenticates the applet with which to share data. In support of this, the JavaCard system library implements a limited form of stack inspection, in the form of a method called `getPreviousContextAID` that allows an applet to find out the owner context of the method executing immediately prior to the last context switch. For the sake of clarity and brevity, we do not consider the facilities sharing and stack inspection in this paper. The analyses and techniques discussed in a later section are easily extended to handle these concepts and this is briefly indicated where relevant in the following sections.

In this paper we shall not go further into the formal details of the firewall semantics, merely refer to [14, 15]. For a thorough introduction to the JavaCard firewall and sharable objects and their use, see [2].

4 Analysing Carmel Programs

In this section we specify a so-called *ownership analysis*, which is a static analysis that conservatively approximates the set of *owner contexts* assigned to an object. This will form the basis for verifying that no security exceptions could possibly be raised by executing the program.

In order for the ownership analysis to be semantically sound, it needs to consider all possible program executions; rather than start from scratch and designing an ownership analysis that directly considers all program executions, we designed the ownership analysis as an extension to a previously developed control flow analysis (CFA) for Carmel. This CFA is described in detail in [5], including a formal statement and proof of semantic correctness.

The CFA, and hence the ownership analysis, is specified in the *Flow Logic* framework of Nielson and Nielson, cf. [8, 12, 7]. In the following subsections we first introduce the abstract domains for the static analysis, following that is a brief overview of the Flow Logic specification framework and finally we discuss a few specification clauses for the ownership and control flow analysis in detail.

4.1 Abstract Domains

The abstract domains are based on a simplified version of the concrete domains used in the semantics, cf. [14]. The simplified domains ignore semantic information that is not pertinent to the analysis. This minimises unnecessary notation and increases legibility of both analysis and theoretical results.

Objects are abstracted into their class, thus object references are modeled as classes (similar to the *class object graphs* of [16]) whereas arrays are abstracted into their elementtype:

$$\mathsf{ObjRef} = \mathsf{Class} \qquad \mathsf{ArRef} = \mathsf{Type}$$

In order to enhance readability we write (Ref σ), rather than merely σ, for object references and (Ref (`array` τ)) rather than τ for array references.

References are either object references or array references:

$$\mathsf{Ref} = \mathsf{ObjRef} + \mathsf{ArRef}$$

Values are taken to be either numerical values (since we are only interested in control flow and ownership, numerical values are abstracted to a single constant) or reference values and abstract values to be sets of such values:

$$\mathsf{Val} = \mathsf{Num} + \mathsf{Ref} \qquad \widehat{\mathsf{Val}} = \mathcal{P}(\mathsf{Val}) \qquad \mathsf{Num} = \{\mathsf{INT}\}$$

In [4] the control flow analysis is extended with a data flow component.

For the ownership analysis we model owners simply as a set of concrete owners and also associate an abstract owner, ie. a set of possible owners, to each method in the program, called an "owner cache":

$$\widehat{\mathsf{Owner}} = \mathcal{P}(\mathsf{Owner}) \qquad \widehat{\mathsf{OwnerCache}} = \mathsf{Method} \to \widehat{\mathsf{Owner}}$$

In order to support stack inspection the above should be extended to record not only the set of possible owners but also the set of possible context switches.

Abstract objects comprise a mapping from the field ID's of the object to the set of abstract values possibly contained in that field and also a set of possible owners for that object:

$$\widehat{\mathsf{Object}} = (\text{fieldValue} : \mathsf{FieldID} \to \widehat{\mathsf{Val}}) \times (\text{owner} : \widehat{\mathsf{Owner}})$$

Information about a given objects status as a JCRE entry point or its shareability can trivially be added here and then checked when verifying a program, cf. Section 5.

Arrays are modeled in the simplest possible way, namely as an abstract value. This means that the structure (and length) of the array is abstracted away:

$$\widehat{\mathsf{Array}} = \widehat{\mathsf{Val}}$$

Adresses consist of a method and a program counter, making adresses unique in a program. In order to correctly handle return values from method invocations

a special "placeholder" address is defined for every method. This placeholder address is encoded using a special END-token instead of the regular program counter

$$\mathsf{Addr} = \mathsf{Method} \times (\mathbb{N} \uplus \{\mathsf{END}\})$$

The first instruction in a method is assumed to be at program counter 1 and we write (m, END_m) for the placeholder address belonging to the method m.

We let $|m|$ denote the arity of method m, meaning the number of arguments the method expects on the operand stack.

The local heap is modeled as a (curried) map from adresses to (local) variables to abstract values. Thus in our model there is a local heap associated with *every instruction* in a method. This is similar to Freund and Mitchells approach, cf. [3].

$$\widehat{\mathsf{LocHeap}} = \mathsf{Addr} \to \mathsf{Var} \to \widehat{\mathsf{Val}}$$

For $\hat{L} \in \widehat{\mathsf{LocHeap}}$ we shall write $\hat{L}(m_1, pc_1) \sqsubseteq \hat{L}(m_2, pc_2)$ to mean

$$\forall x \in \mathrm{dom}(\hat{L}(m_1, pc_1)) : \hat{L}(m_1, pc_1)(x) \sqsubseteq \hat{L}(m_2, pc_2)(x)$$

and $\hat{L}(m_1, pc_1) \sqsubseteq_{\{x\}} \hat{L}(m_2, pc_2)$ to mean

$$\forall y \in \mathrm{dom}(\hat{L}(m_1, pc_1)) \setminus \{x\} : \hat{L}(m_1, pc_1)(y) \sqsubseteq \hat{L}(m_2, pc_2)(y)$$

Note that local variables are denoted by natural numbers and zero, ie. $\mathsf{Var} = \mathbb{N}_0$.

We now turn to the operand stack. Since the model has to be able to cope with potentially infinite operand stacks, we use the following domain as the basis for the stack model:

$$\widehat{\mathsf{Val}}^\infty = \widehat{\mathsf{Val}}^\omega \cup \widehat{\mathsf{Val}}^*$$

However, in anticipation of later developments and applications of the analysis, rather than using the above domain directly, we use it to induce a more convenient domain via a Galois connection (cf. [8]):

$$\widehat{\mathsf{Val}}^\infty \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\widehat{\mathsf{Val}}^*)^\top$$

where the abstraction function, α, simply acts as the identity on finite stacks and maps infinite stacks to top.

With the basic domain for abstract stacks in place, we now associate an abstract operand stack with every instruction in a method in order to track operations on the stack in that method:

$$\widehat{\mathsf{Stack}} = \mathsf{Addr} \to (\widehat{\mathsf{Val}}^*)^\top$$

Elements of $(\widehat{\mathsf{Val}}^*)^\top$ are written much in the same style as SML lists, thus $(A_1 :: A_2 :: \cdots :: X) \in (\widehat{\mathsf{Val}}^*)^\top$ represents a stack with $A_1 \in \widehat{\mathsf{Val}}$ as its top element and $X \in \widehat{\mathsf{Val}}^\omega$ as the "bottom" of the stack. The empty stack is denoted by ϵ.

We introduce the following ordering on abstract stacks, $A_1 :: \cdots :: A_n$ and $B_1 :: \cdots :: B_m$, from $(\widehat{\mathsf{Val}^*})^\top$:

$$(A_1 :: \cdots :: A_n) \sqsubseteq (B_1 :: \cdots :: B_m) \iff m \geq n \wedge \forall i \in \{1, \ldots, n\} : A_i \subseteq B_i$$

In the interest of succinctness we shall abuse the above notation slightly by writing $(A_0 :: \cdots :: A_n) \sqsubseteq \hat{L}(m_0, pc_0)[0..n]$ as a shorthand for

$$\forall i \in \{0, \ldots, n\} : A_i \subseteq \hat{L}(m_0, pc_0)(i)$$

The abstract global heap comprises two components: an object component, keeping track of instance fields of individual objects, and a static component, that tracks the values of static fields for each class:

$$\widehat{\mathsf{StaHeap}} = \mathsf{FieldID} \to \widehat{\mathsf{Val}} \qquad \widehat{\mathsf{Heap}} = (\mathsf{ObjRef} \to \widehat{\mathsf{Object}}) + (\mathsf{ArRef} \to \widehat{\mathsf{Array}})$$

4.2 The Flow Logic Framework

The Flow Logic framework can be seen as a "specification approach" to static analysis, rather than an "implementation approach". In the framework, instead of detailing how a particular static analysis is to be carried out, it is specified what it means for an analysis result (or rather a *proposed* analysis result) to be acceptable (correct) with respect to a program. Flow Logic specifications are usually classified as either *verbose* or *succinct* according to the style of specification: succinct resembling the style of type-systems in only reporting "top-level" information and verbose more like traditional data flow and constraint based analyses in recording all internal flows. The specification in this paper is a verbose specification. We shall not go into further detail with the framework here, merely refer to [8, 12, 7] for further information.

The judgements of the Flow Logic specification for the analysis of Carmel will be on the form

$$(\hat{K}, \hat{H}, \hat{O}, \hat{L}, \hat{S}) \models \texttt{addr} : \texttt{instr}$$

where $\hat{S} \in \widehat{\mathsf{Stack}}$, $\hat{L} \in \widehat{\mathsf{LocHeap}}$, $\hat{H} \in \widehat{\mathsf{Heap}}$, $\hat{K} \in \widehat{\mathsf{StaHeap}}$, $\hat{O} \in \widehat{\mathsf{OwnerCache}}$, $\texttt{addr} \in \mathsf{Addr}$ and `instr` is the instruction at `addr`. Intuitively the above states that $(\hat{K}, \hat{H}, \hat{O}, \hat{L}, \hat{S})$ is an *acceptable* analysis for the instruction `instr` at address `addr`. A detailed discussion of the clauses and judgements for Carmel are given in the following sections.

4.3 Example Clauses

The `putfield` Instruction First the specification for the `putfield` instruction:

$$(\hat{K}, \hat{H}, \hat{O}, \hat{L}, \hat{S}) \models (m_0, pc_0) : \texttt{putfield } f$$

The `putfield` instruction transfers the value of the top element of the stack to the field named as argument to the instruction in the object referenced in the second element of the stack. Thus the stack must contain at least two elements:

$$A :: B :: X \lhd \hat{S}(m_0, pc_0) :$$

The specific object to be accessed is resolved at runtime, and a reference to that object is stored in the second (from the top) element of the stack. The value of the top element is then stored in the field of the object so referenced:

$$\forall(\text{Ref } \sigma') \in B : A \sqsubseteq \hat{H}(\text{Ref } \sigma').\text{fieldValue}(f)$$

Here we use the abstract global heap to hold information about the fields of abstract objects. As noted in Section 4.1 objects are abstracted into their class. Thus field information for all objects of the same class is merged and stored in the abstract global heap.

The bottom of the stack is then transferred to the next instruction:

$$X \sqsubseteq \hat{S}(m_0, pc_0 + 1)$$

and since no local variables were modified, the abstract local heap is transferred unchanged to the next instruction

$$\hat{L}(m_0, pc_0) \sqsubseteq \hat{L}(m_0, pc_0 + 1)$$

We then arrive at the following clause for `putfield` instructions:

$$\begin{array}{rl} (\hat{K}, \hat{H}, \hat{O}, \hat{L}, \hat{S}) \models (m_0, pc_0) : \texttt{putfield } f & \\ \quad \text{iff} & A :: B :: X \lhd \hat{S}(m_0, pc_0) : \\ & \forall(\text{Ref } \sigma') \in B : \\ & \quad A \sqsubseteq \hat{H}(\text{Ref } \sigma').\text{fieldValue}(f) \\ & X \sqsubseteq \hat{S}(m_0, pc_0 + 1) \\ & \hat{L}(m_0, pc_0) \sqsubseteq \hat{L}(m_0, pc_0 + 1) \end{array}$$

Note that while the `putfield` instruction is mediated by the firewall, there is no switch in owner context and therefore no constraints specific to the ownership analysis in the above specification (contrary to the case for `invokevirtual`).

The `invokevirtual` Instruction Finally we discuss the specification for the `invokevirtual` instruction:

$$(\hat{K}, \hat{H}, \hat{O}, \hat{L}, \hat{S}) \models (m_0, pc_0) : \texttt{invokevirtual } m$$

In order to call an instance method, the `invokevirtual` instruction is used. Arguments to the method is found at the top of the stack, and as was the case for the `putfield` instruction, a reference to the specific object containing the invoked method is found on the stack, immediately following the arguments to the method:

$$A_1 :: \cdots :: A_{|m|} :: B :: X \lhd \hat{S}(m_0, pc_0) :$$

Next a method lookup is needed in order to find the actual method that is executed:

$$m_v = \mathit{methodLookup}(m, \sigma')$$

The arguments are transferred to the called method as *local variables* of the called method. Furthermore a reference to object containing the called method is passed as the first local variable (in effect a `this` pointer):

$$\{(\text{Ref } \sigma')\} :: A_1 :: \cdots :: A_{|m|} \sqsubseteq \hat{L}(m_v, 1)[0..|m_v|]$$

Furthermore when a method is invoked in an object, the method will execute in the same owner context as the object from which it was invoked. This is modeled in the ownership analysis in the following way:

$$\hat{H}(\text{Ref } \sigma').\text{owner} \sqsubseteq \hat{O}(m_v)$$

Had we chosen to model not only the set of possible owners but also the set of possible context switches (to support stack inspection, as mentioned in Section 4.1) a further constraint, updating the set of context switches, would be needed here.

When a method invocation returns, there are two possibilities: either it does not return a value, ie. it has return type `void`, or it does return a value. In the first case, $m.\text{returnType} = \texttt{void}$, we simply copy the rest of the stack on to the next instruction:

$$\begin{array}{c} m.\text{returnType} = \texttt{void} \Rightarrow \\ X \sqsubseteq \hat{S}(m_0, pc_0 + 1) \end{array}$$

In the latter case, $m.\text{returnType} \neq \texttt{void}$, the return value is the top element of the stack of the invoked method. In order to handle multiple returns from the invoked method correctly a special address is used, indicated by the END-token discussed in Section 4.1; it is the responsibility of the clause for the `return` instruction to ensure, that all the possible stacks at all possible `return` instructions are transferred to the stack at the special address.

In order for the invoking method to access the return value, it must be transferred from the top of the stack of the *invoked* method to the top of the stack of the *invoking* method (less the arguments and the object reference):

$$\begin{array}{l} m.\text{returnType} \neq \texttt{void} \Rightarrow \\ \quad T :: Y \triangleleft \hat{S}(m_v, \mathsf{END}_{m_v}) : \ T :: X \sqsubseteq \hat{S}(m_0, pc_0 + 1) \end{array}$$

Finally, none of the local variables (of the invoking method) have been altered and are therefore passed on to the next instruction:

$$\hat{L}(m_0, pc_0) \sqsubseteq \hat{L}(m_0, pc_0 + 1)$$

Joining the above equations we obtain the following clause for `invokevirtual` instructions:

$$
\begin{array}{l}
(\hat{K},\hat{H},\hat{O},\hat{L},\hat{S}) \models (m_0,pc_0) : \texttt{invokevirtual } m \\
\quad \text{iff} \quad A_1 :: \cdots :: A_{|m|} :: B :: X \triangleleft \hat{S}(m_0,pc_0) : \\
\qquad \forall(\text{Ref } \sigma') \in B : \\
\qquad\quad m_v = \mathit{methodLookup}(m,\sigma') \\
\qquad\quad \{(\text{Ref } \sigma')\} :: A_1 :: \cdots :: A_{|m|} \sqsubseteq \hat{L}(m_v,1)[0..|m_v|] \\
\qquad\quad \hat{H}(\text{Ref } \sigma').\text{owner} \sqsubseteq \hat{O}(m_v) \\
\qquad\quad m.\text{returnType} \neq \texttt{void} \Rightarrow \\
\qquad\qquad T :: Y \triangleleft \hat{S}(m_v,\mathsf{END}_{m_v}) : \; T :: X \sqsubseteq \hat{S}(m_0,pc_0+1) \\
\qquad\quad m.\text{returnType} = \texttt{void} \Rightarrow \\
\qquad\qquad X \sqsubseteq \hat{S}(m_0,pc_0+1) \\
\qquad \hat{L}(m_0,pc_0) \sqsubseteq \hat{L}(m_0,pc_0+1)
\end{array}
$$

5 Verifying Carmel Programs

In this section we show how an acceptable analysis result, as specified in Section 4, can be used to guarantee that Carmel programs do not (try to) breach the firewall.

The `putfield` instruction is checked by the firewall and may potentially throw a security exception. The instruction is allowed to proceed only if either the method executing the `putfield` instruction is in the `JCRE` system-context, or if it has the same owner context as the object whose field is being accessed. This can be formalised as a formula to be checked against the analysis of a program: for every `putfield` instruction the following predicate must hold:

$$
\begin{array}{l}
\hat{O}(m_0) = \{\texttt{JCRE}\} \vee \\
(|\hat{O}(m_0)| = |\hat{H}(\text{Ref } \sigma').\text{owner}| = 1) \wedge (\hat{O}(m_0) = \hat{H}(\text{Ref } \sigma').\text{owner})
\end{array}
$$

where m_0 and (Ref σ') are quantified as in the analysis (cf. Section 4.3). If it does hold for every `putfield` instruction then none of those instructions will violate the firewall rules. The same rationale applies to the `invokevirtual` instruction and gives rise to exactly the same predicate that should be checked.

Note that due to the conservative nature of the ownership analysis, guarantees can only be made when we are sure of the owner, ie. when $|\hat{O}(m_0)| = |\hat{H}(\text{Ref } \sigma').\text{owner}| = 1$.

Had we chosen to support sharable objects, entry points and stack inspection the observation predicates would of course be more involved, but still easily definable.

6 Implementation

Following the tradition of the Flow Logic framework, analyses are implemented by first converting the high-level Flow Logic specification into a corresponding *constraint generator* over a suitable constraint language.

In this section we briefly outline how the flow logic specification given in Section 4 systematically can be transformed into a specification for generating constraints in the Alternation-free Least Fixed-Point logic (ALFP). Constraints over this logic can be solved efficiently using the techniques described in [10, 11].

6.1 Alternation-free Least Fixed-Point logic

Formulae in ALFP consists of clauses of the following form:

$$\mathsf{pre} ::= R(x_1, \ldots, x_k) \mid \mathsf{pre}_1 \wedge \mathsf{pre}_2 \mid \mathsf{pre}_1 \vee \mathsf{pre}_2 \mid \exists x : \mathsf{pre}$$

$$\begin{aligned} \mathsf{clause} ::= {} & R(x_1, \ldots, x_k) \mid \mathbf{1} \mid \mathsf{clause}_1 \wedge \mathsf{clause}_2 \\ \mid {} & \mathsf{pre} \Rightarrow \mathsf{clause} \mid \forall x : \mathsf{clause} \end{aligned}$$

where R is a k-ary relation symbol for $k \geq 1$ and $x_1, \ldots$ denote variables while $\mathbf{1}$ is the always true clause.

We shall not go in to any detail here, but it is straightforward to define the satisfaction relation, $(\rho, \sigma) \models_{\mathrm{ALFP}} t$, for ALFP over a universe of atomic values and interpretations ρ and σ of relation symbols and free variables respectively. For a given interpretation, σ, of constants we call an interpretation, ρ, of relation symbols a *solution* to a clause clause if indeed $(\rho, \sigma) \models_{\mathrm{ALFP}} \mathsf{clause}$.

Using the techniques of [10] it is possible to efficiently find solutions to given clauses. An implementation, called Succinct Solver, using these and other advanced techniques has been made by Nielson and Seidl and is described in [11].

6.2 Representing the Abstract Domains

In order to use ALFP as the basis for an implementation of the analysis, we must first find a way to represent the abstract domains of the analysis in ALFP.

Here we only show how to represent the abstract stack and ownership information and refer to [6] for a more detailed discussion on how to generate ALFP constraints from Flow Logic specifications and also how to optimise the generated constraints for speed and memory.

Stacks. In order to model the abstract stack we use a quarternary relation, S, relating adresses and stack positions to values, thus the clause

$$\mathsf{S}(m_0, pc_0, [3], \mathsf{INT})$$

is intended to mean that the abstract stack at address (m_0, pc_0) contains an integer value at stack position three.

Since we must be able to manipulate and calculate stack positions directly within the clauses, stack positions must be represented explicitly:

$$\begin{aligned} [0] &= \mathtt{zero} \\ [n+1] &= \mathtt{suc}([n]) \end{aligned}$$

since only stack positions, and not eg. local variable indices, are manipulated or calculated directly within the clauses, only these need to be represented using the above.

We can now model an abstract stack, $\hat{S}(m_0, pc_0) = A_1 :: \cdots :: A_n$, at address (m_0, pc_0) where $A_i = \{a_i^1, \ldots, a_i^{j_i}\}$ as follows:

$$\begin{array}{c} \mathsf{S}(m_0, pc_0, [0], a_1^1) \wedge \ldots \wedge \mathsf{S}(m_0, pc_0, [0], a_1^{j_1}) \wedge \\ \vdots \\ \mathsf{S}(m_0, pc_0, [n-1], a_n^1) \wedge \ldots \wedge \mathsf{S}(m_0, pc_0, [n-1], a_n^{j_n}) \end{array}$$

Thus, the top of the stack is at position zero, [0], with the the rest of the stack in the following positions.

Ownership. The owner cache is simply modeled as a binary relation, $\mathsf{O}(m, o)$, relating a method m to an owner o thus representing $o \in \hat{O}(m)$. Similarly the owner field of an abstract object is modeled by another binary relation, $\mathsf{H\_OWNER}(r, o)$, relating an object reference (in effect an object) to an owner, o which represents $o \in \hat{H}(r)$.owner.

The other components of the Flow Logic specification are modeled in a similar manner.

6.3 Generating Constraints

As for the flow logic specification of the analysis, we only show a few representative cases for the constraint generation.

The constraint generation is specified as a relation, $\rightsquigarrow$, between an instruction (at a given address) and a clause in ALFP.

The `putfield` Instruction Storing values in instance fields is accomplished by the `putfield`-instruction. Based on the analysis of the instruction (cf. Section 4.3) we see that the value on top of the stack is copied into the field pointed to by the reference found in the second position of the stack. Converting this to constraints we get

$$\forall r : \forall a : \mathsf{S}(m_0, pc_0, [1], r) \wedge \mathsf{S}(m_0, pc_0, [0], a) \Rightarrow \mathsf{H}(r, f, a)$$

Now the remainder of the stack, the original stack less the top two elements, should be copied onwards to the next instruction:

$$\forall y : \forall a : \forall i : y = [i+2] \wedge \mathsf{S}(m_0, pc_0, y, a) \Rightarrow \mathsf{S}(m_0, pc_0 + 1, i, a)$$

None of the local variables were modified and should simply be copied onwards:

$$\forall x : \forall a : \mathsf{L}(m_0, pc_0, x, a) \Rightarrow \mathsf{L}(m_0, pc_0 + 1, x, a)$$

Combining the above constraints we can formulate the clause for `putfield`:

$$
\begin{array}{l}
(m_0, pc_0) : \texttt{putfield}\ f \rightsquigarrow \\
\quad \forall r : \forall a : \mathsf{S}(m_0, pc_0, [1], r) \wedge \mathsf{S}(m_0, pc_0, [0], a) \Rightarrow \mathsf{H}(r, f, a) \\
\quad \forall y : \forall a : \forall i : y = [i+2] \wedge \mathsf{S}(m_0, pc_0, y, a) \Rightarrow \mathsf{S}(m_0, pc_0 + 1, i, a) \\
\quad \forall x : \forall a : \mathsf{L}(m_0, pc_0, x, a) \Rightarrow \mathsf{L}(m_0, pc_0 + 1, x, a)
\end{array}
$$

Finally, having generated the above constraints for the control flow and ownership analyses, we now generate an *observation predicate* which is a formalisation of the analysis checks described in Section 5 as an ALFP formula. Thus we obtain an implementation where the verification of the analysis result is actually carried out while computing the analysis result and thereby leveraging the efficiency of the Succinct Solver technology; alternatively we could have computed the result first and then carried out the verification at the possible cost of adding further overhead.

The observation predicate below will check if the instruction in question can possibly violate the firewall rules and if so it records the addrss of the potential violation in an auxiliary relation called ALERT:

$$
\begin{array}{l}
\forall r : \mathsf{S}(m_0, pc_0, [1], r) \Rightarrow \\
\quad (\exists x \exists y : \mathsf{O}(m_0, x) \wedge x \neq \texttt{JCRE} \wedge x \neq y \wedge \mathsf{H\_OWNER}(r, y)) \Rightarrow \\
\quad\quad \mathsf{ALERT}(m_0, pc_0)
\end{array}
$$

Once the solver has found a solution all that remains is to extract the list of addresses of potential violations and present them to the user. Of course more information than merely the address can be recorded, eg. exactly which owner contexts gave rise to a potential violation; what additional information will prove to be most useful can only be discovered by experimentation.

The `invokevirtual` Instruction First the reference to the object where the invoked method resides is copied (as a self reference) to local variable 0 of the invoked method:

$$
\forall r \forall mv : \mathsf{S}(m_0, pc_0, [|m|], r) \wedge \mathsf{ML}(m.\mathrm{id}, r, mv) \Rightarrow \mathsf{L}(mv, 1, \mathsf{var\_0}, r)
$$

Next the parameters are transferred from the stack of the current method to the local variables of the invoked method:

$$
\begin{array}{c}
\forall r \forall mv \forall a : \mathsf{S}(m_0, pc_0, [|m|], r) \wedge \mathsf{ML}(m.\mathrm{id}, r, mv) \wedge \\
\quad \mathsf{S}(m_0, pc_0, [0], a) \Rightarrow \mathsf{L}(mv, 1, \mathsf{var\_1}, a) \\
\vdots \\
\forall r \forall mv \forall a : \mathsf{S}(m_0, pc_0, [|m|], r) \wedge \mathsf{ML}(m.\mathrm{id}, r, mv) \wedge \\
\quad \mathsf{S}(m_0, pc_0, [|m| - 1], a) \Rightarrow \mathsf{L}(mv, 1, \mathsf{var\_|m|}, a)
\end{array}
$$

And then the ownership information is copied forward:

$$
\begin{array}{c}
\forall r \forall mv \forall o : \mathsf{S}(m_0, pc_0, [|m|], r) \wedge \mathsf{ML}(m.\mathrm{id}, r, mv) \wedge \\
\quad \mathsf{H\_OWNER}(r, o) \Rightarrow \mathsf{O}(mv, o)
\end{array}
$$

In case the method returns a value, that value should be put on top of the stack for the next instruction, and the rest of the current stack, less the arguments to the invoked method, is also copied forward. Thus if m.returnType $\neq$ `void` then the following constraints are generated:

$$\begin{array}{l}\forall y : \forall z : \forall a : \forall i : \\ \quad y = [i + |m| + 1] \wedge z = [i + 1] \wedge \mathsf{S}(m_0, pc_0, y, a) \Rightarrow \mathsf{S}(m_0, pc_0 + 1, z, a) \\ \forall r \forall mv \forall endmv \forall a : \mathsf{S}(m_0, pc_0, [|m|], r) \wedge \mathsf{ML}(m.\mathrm{id}, r, mv) \wedge \\ \quad \mathsf{END}(mv, endmv) \wedge \mathsf{S}(mv, endmv, [0], a) \Rightarrow \mathsf{S}(m_0, pc_0 + 1, [0], a)\end{array}$$

If on the other hand the invoked method does not return a value, then only the current stack, less the arguments to the invoked method, is copied forward. Thus if m.returnType = `void` then the following constraints are generated:

$$\forall y : \forall a : \forall i : y = [i + |m| + 1] \wedge \mathsf{S}(m_0, pc_0, y, a) \Rightarrow \mathsf{S}(m_0, pc_0 + 1, [i], a)$$

Finally, since the local variables of the invoking method are not modified, they are simply copied along as well:

$$\forall x \forall a : \mathsf{L}(m_0, pc_0, x, a) \Rightarrow \mathsf{L}(m_0, pc_0 + 1, x, a)$$

As for `putfield` we construct an observation predicate for `invokevirtual` along the same lines:

$$\begin{array}{l}\forall r : \mathsf{S}(m_0, pc_0, [|m|], r) \Rightarrow \\ \quad (\exists x \exists y : \mathsf{O}(m_0, x) \wedge x \neq \texttt{JCRE} \wedge x \neq y \wedge \mathsf{H\_OWNER}(r, y)) \Rightarrow \\ \quad\quad \mathsf{ALERT}(m_0, pc_0)\end{array}$$

6.4 Solving the Constraints

Once the constraints are generated for all the instructions, all that remains is to solve them. As mentioned earlier an efficient solver, called Succint Solver, has been implemented by Seidl and Nielson. We shall not go into more detail here, merely refer to [10, 11] for more information.

6.5 The Prototype

A prototype tool for parsing Carmel programs, generating constraints as discussed in this section and interfacing with the Succinct Solver has been implemented. At the moment the tool simply presents the analysis result in its entirety, however, work on a better presentation based on the program source and using syntax colouring and hyperlinks is under way.

7 An Example

In Figure 1 a small Carmel example program is shown. The program defines two classes: `Account` and `Bad`. The `Account` class is intended as a (very) simplified version of an electronic purse with only two methods: `credit` that checks

```
class Account {
  void credit (int) {
    /* Do some checking... */
    load 0
    load 1
    invokevirtual Account.add
    return
  }
  void add (int) {
    load 1
    getfield this Account.balance
    numop add
    putfield this Account.balance
    return
  }
}
```

```
class Bad {
  void steal (void) {
    getstatic Account.leak
    dup 1 0
    getfield Account.balance
    push 5000
    numop add
    invokevirtual Account.add
    return
  }
}
```

Fig. 1. Example Carmel Program

and validates crediting an account and `add` that does the actual update of the balance.

The `Bad` class is intended as a (very) simplified attacker that credits an account with 5000 units, bypassing the sanity checks. We assume that an object reference to the account is leaked through a static field in the `Account` class, called `leak`. The firewall cannot detect and stop access through static fields since they have no owners. However, when the `Bad` class tries to access the account and get the balance, this is in violation with the firewall policy. Likewise the invocation of the `add` method is also in violation with the firewall policy. Figure 2 shows the constraints generated for the `invokevirtual Account.add` instruction in the `steal` method of class `Bad` and Figure 3 the generated observation predicate for that instruction.

```
(A r. S(cl_Bad,steal,pc_6,suc(zero),r) =>
  L(r,add,pc_1,var_0,r)) &
(A r. A a. S(cl_Bad,steal,pc_6,suc(zero),r) &
  S(cl_Bad,steal,pc_6,zero,a) => L(r,add,pc_1,var_1,a)) &
(A x. A i. A a. x = suc(suc(i)) & S(cl_Bad,steal,pc_6,x,a) =>
  S(cl_Bad,steal,pc_7,i,a)) &
(A x. A a. L(cl_Bad,steal,pc_6,x,a) => L(cl_Bad,steal,pc_7,x,a))&
(A r. A o. S(cl_Bad,steal,pc_6,suc(zero),r) & OWNER(r,o) =>
  O(r,add,o))
```

Fig. 2. Constraints generated for `invokevirtual Account.add` in method `steal`

```
(A r. ((S(cl_Bad,steal,pc_6,suc(zero),r)) =>
  ((E x. (E y.
    ((O(cl_Bad,steal,x) & (x != cl_JCRE) &
     (x != y) & (OWNER(r,y)))))) => (ALERT(cl_Bad,steal,pc_6)))))
```

Fig. 3. Observation predicate generated for `invokevirtual Account.add` in method `steal`

Since we are not modelling the JCRE we must explicitly set up the initialisation otherwise done by the JCRE:

```
OWNER(cl_Account,OWN_bank) &
OWNER(cl_Bad,OWN_hacker) &
O(cl_Account,credit,OWN_bank) &
O(cl_Account,add,OWN_bank) &
O(cl_Bad,steal,OWN_hacker)
```

The above contraints correspond to installing an application owned by `bank` and having only one class, namely `Account`, with two methods: `credit` and `add`, and then installing another application owned by `hacker` consisting of the class `Bad` with only one method `steal`. These initialisation constraints could easily be computed automatically, but they are only necessary when the JCRE is not modelled explicitly. The constraints given here assume that all methods in all classes can be invoked from "the outside" and so is a safe approximation of the actual call-patterns.

Finally, in Figure 4 the result of solving the constraints is shown, namely the `ALERT` relation. The analysis finds two possible breaches of the firewall rules,

```
Relation ALERT/3:
        (cl_Bad, steal, pc_6), (cl_Bad, steal, pc_3),
```

Fig. 4. Solution for `invokevirtual Account.add` in method `steal`

both in method `steal` of class `Bad`, in program line three and six respectively, corresponding to the `getfield` and `invokevirtual` instruction.

Even though the analysis decsribed in this paper is rather imprecise, we belive it is sufficient for JavaCard programs that follow the current practices. In particular, most JavaCards do not (yet) have a garbage collector and thus all memory allocation is done in the initialisation phase and only rarely are classes instantiated more than once. This justifies our abstraction of objects into classes and also the rather simple notion of ownership. In the future this may well change and then the analysis might not be precise enough, however, by adding more information to the abstract object references, eg. adding ownership information, we believe that sufficient precision can be achieved.

8 Conclusion and Future Work

In this paper we have shown how a prototype tool for validating JavaCard programs has been constructed by extending an existing control flow analysis with ownership information. From this extended analysis a constraint generator for the Succint Solver was systematically derived and it was discussed how verification checks are formalised, also as constraints, and verified by the Succinct Solver. We believe that such a tool will be very useful, in particular for developers, for ensuring the robustness of their programs and also for increasing confidence in the security behaviour of their programs.

Work on extending and consolidating the ideas in this paper is already underway mainly with respect to adding support for sharable objects (including stack inspection) and entry points. As noted various places in the paper, there are no conceptual difficulties in supporting sharable objects, entry points and stack inspection and requires only a few modifications.

Another direction, where work is also underway, is to make the tool more user friendly, in particular regarding the presentation of the analysis result. Here a number of possibilities are being considered and tested.

Finally we would like to investigate how well the approach scales to "real-world" JavaCard programs. While the current prototype is a rather naive implementation of the analysis we believe that by employing different optimisation strategies, discussed in [1, 6, 10, 11], we can achieve the desired scalability. Anecdotal evidence based on experiments with a similar constraint generator (only for control flow analysis), seems to indicate that time complexity will be less of an issue than space consumption, indeed constraint generation and solution for a program of a few thousand lines only takes around 30 seconds. In [6] a number of ideas for reducing space consumption is described. Furthermore, even real-world JavaCard programs are of (relatively) small size which of course makes them even more amenable to using formal methods for validation. Even so, in anticipation of future needs work should be done on investigating various ways of modularising and partitioning analyses so that analysis of a system might be done in a stepwise fashion. For applets that do not communicate via shared objects such partitioning should be fairly straightforward. For applets that do communicate, analyses for determining communication interfaces is needed; earlier work on "hardest attackers" for mobile ambients, cf. [9], may be a suitable basis for such work.

References

1. Mikael Buchholtz, Hanne Riis Nielson, and Flemming Nielson. Experiments with Succinct Solvers. SECSAFE-IMM-002-1.0. Also published as DTU Technical Report IMM-TR-2002-4, February 2002.
2. Zhiqun Chen. *Java Card Technology for Smart Cards.* The Java Series. Addison Wesley, 2000.
3. Stephen N. Freund and John C. Mitchell. A Formal Framework for the Java Bytecode Language and Verifier. In *ACM Conference on Object-Oriented Programming,*

Systems, Languages, and Applications, OOPSLA'99, pages 147–166, Denver, CO, USA, November 1999. ACM Press.
4. René Rydhof Hansen. Extending the Flow Logic for Carmel. SECSAFE-IMM-003-1.0 (available from the author upon request), 2002.
5. René Rydhof Hansen. Flow Logic for Carmel. SECSAFE-IMM-001-1.5, 2002.
6. René Rydhof Hansen. Implementing the Flow Logic for Carmel. SECSAFE-IMM-004-DRAFT. Forthcoming, 2002.
7. Flemming Nielson. Flow Logic. Web page: `http://www.imm.dtu.dk/~nielson/FlowLogic.html`.
8. Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis*. Springer Verlag, 1999.
9. Flemming Nielson, Hanne Riis Nielson, and René Rydhof Hansen. Validating Firewalls using Flow Logics. *Theoretical Computer Science*, 283(2):381–418, 2002.
10. Flemming Nielson and Helmut Seidl. Control-Flow Analysis in Cubic Time. In *Proc. ESOP'01*, April 2001. Also appears as SECSAFE-DAIMI-006-1.0.
11. Flemming Nielson and Helmut Seidl. Succinct solvers. Technical Report 01-12, University of Trier, Germany, 2001.
12. Hanne Riis Nielson and Flemming Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. Lecture Notes in Computer Science. Springer Verlag, 2002. To appear.
13. SecSafe Home Page. `http://www.doc.ic.ac.uk/~siveroni/secsafe/`.
14. Igor Siveroni and Chris Hankin. A Proposal for the JCVMLe Operational Semantics. SECSAFE-ICSTM-001-2.2, October 2001.
15. Igor Siveroni, Thomas Jensen, and Marc Eluard. A Formal Specification of the Java Card Firewall. In Hanne Riis Nielson, editor, *Proc. of Nordic Workshop on Secure IT-Systems, NordSec'01*, pages 108–122, Lyngby, Denmark, November 2001. Proceedings published as DTU Technical Report IMM-TR-2001-14.
16. Jan Vitek, R. Nigel Horspool, and James S. Uhl. Compile-Time Analysis of Object-Oriented Programs. In *Proc. CC'92*, volume 641 of *Lecture Notes in Computer Science*, pages 236–250. Springer Verlag, 1992.

Design and Implementation of a Firewall and a Packet Manipulator for Network Simulation Using SSFNet

Cheol-Won Lee, Eul Gyu Im[1], and Dong-Kyu Kim[2]

[1] National Security Research Institute
62-1 Hwa-am-dong, Yu-seong-gu
Daejeon, 305-718, Republic of Korea
{cheolee,imeg}@etri.re.kr
[2] Department of Computer Engineering
Ajou University, Suwon 442-749
Republic of Korea
dkkim@madang.ajou.ac.kr

Abstract. Cyber attacks affect many things on networks, such as network latency, server loads, data integrity of files and so on. One of the best feasible ways to research cyber attacks and their consequences is simulations. To simulate cyber attacks, a simulation model must be able to describe various network components as well as cyber attacks and security mechanisms. We designed and implemented modules to support firewalls and packet manipulation in the Scalable Simulation Framework (SSF) [1] that is a process-based discrete event simulation framework. These modules, which were added to one of SSF components, that is SSFNet [2, 3], allow users more comprehensive network simulations including diverse cyber attack simulations. With the addition of these new modules, SSFNet can simulate various network attacks and their damages, verify correctness and effects of security policies, and determine appropriate security mechanisms against a certain attack.
Keywords: cyber attack, simulation, SSF, security

1 Introduction

Cyber attacks are increasingly taking on a grander scale. Attacks may be launched from a wide base against a large number of targets with the intention of acquiring resources as well as disrupting services. Research on cyber attacks and their effects is fundamental to defend computer systems against cyber attacks in real world [4, 5]. Even though lots of research was done on Internet attacks, there still needs more work to do because the sizes of networks are very big and they keep growing and attacks are getting very complex and diverse. There are many ways to track down and study cyber attacks. Among them, the best way to get data of network changes and damages against a certain attack is to apply the attack on real environments and get results from them. But it is almost impossible to get real data of all the possible attacks since many attacks are done in

very large distributed networks and experiments of attacks can disrupt services of the servers or can even cause to shut down the whole servers on the networks. Experiments with a small set of servers can be done to collect data, but this method also has limitations because there should be a new network configuration for each experiment and some experiments may require special equipments. Simulations can provide data on effects of cyber attacks without interfering real systems or services, and information about new hardwares can be easily fed into a simulator. Therefore simulations are one of the best feasible ways to investigate impacts of Internet attacks, and many people are working on this topic [6].

To simulate cyber attacks and changes of network environments, the simulator must satisfy the following conditions. First, the simulator must be able to model structures and environments of target networks. In addition, the simulator must be able to simulate attacks on widely distributed networks.

Second, network subsystems, their characteristics, and their functions must be able to be represented in the network model. For example, it must be able to represent various packet processing steps performed in communication protocol stacks of operating systems because most network attacks are based on processes on network protocol stacks.

Third, results of simulated attacks must be very similar to those of real cyber attacks. Rather than simple sequences based on statistics, the sequences of attack packets in simulations must be close to patterns of real attacks.

Lastly, it must be able to express characteristics and mechanisms of defense systems. Firewalls are widely used to protect networks against attacks, so a simulator must be able to model networks with firewalls and IDSs (Intrusion Detection Systems) [7–9].

Unfortunately, there is no such simulator that satisfies all the above requirements. Simulators, introduced so far, allow simulating only specific attacks in specific areas. To establish a framework for network simulations of cyber attacks in very large networks, in this paper we introduced firewalls and a packet manipulator to Scalable Simulation Framework (SSF) [1], and these additions enable users to simulate more comprehensive cyber attacks and detections especially those required packet manipulations.

This paper is structured as follows. Section 2 gives a brief introduction of Scalable Simulation Framework. Designs of a firewall and a packet manipulator are addressed in Section 3 and Section 4. Section 5 provides experimental results followed by conclusions in Section 6.

2 Scalable Simulation Framework(SSF)

SSF is a process-based discrete event-oriented simulation kernel, and with the SSFNet component SSF can describe and simulate networks with more than 100,000 nodes. In SSFNet, objects of network subsystems, such as routers, links, and network interface cards, and so on are programmed in Java, and their properties and roles also can be programmed and can be easily changed and expanded. In addition, network protocol stacks, such as TCP, UDP, ICMP, and IP layers

are also implemented, so attacks using these protocols can be simulated using SSFNet. Since many application programs for simulation are also programmed in Java, SSFNet can be easily incorporated with these applications and it is possible to do various experiments with attacks [3]. Even though SSFNet has many advantages, to simulate attacks on network and their consequences, some functionalities need to be implemented and added. Some attacks cannot be expressed using protocol stacks implemented in SSFNet because there are no class libraries to capture network packets. To simulate defense mechanisms, security system modules, like firewalls, also need to be added to SSFNet.

We designed and implemented modules to support firewalls and a packet manipulator. This expansion of SSFNet allows users to be able to perform more diverse simulations. In the following sections, details about the newly added modules will be addressed.

3 Firewall

A firewall protects computers and networks by dropping harmful packets and enforces an access control policy for accesses going in both directions. We designed and added modules for firewalls to SSFNet, and details about firewall components and their usages are addressed in this chapter. This paper models packet-filtering firewalls by expanding an IP layer rather than implementing an independent component. To do this, Secure IP classes were added using IP classes in SSFNet.

3.1 SecureIP Classes

Figure 1 shows packet flows in a Secure IP layer with packet filtering facilities. Packets pass through the Secure IP layer, and the Secure IP Layer has packet analyzers, packet checkers, and rule sets for packet checkers. Packets are divided into three categories: in packets, out packets, and forwarded packets. Packets that are sent to the firewall server are called 'in' packets, and packets from the firewall server are called 'out' packets. Forwarded packets pass through the firewall server from one server to another. A packet analyzer distinguishes forwarded packets from in or out packets. A packet checker applies the corresponding rule set to packets.

This Secure IP layer can model a firewall that analyzes all packets passing through a server, and applies corresponding filtering rules to drop malicious packets. If the result of filtering rules for a packet shows 'permit,' then packets are passed to the next layer normally or forwarded to other servers. Otherwise, packets are dropped. If more than one rule is matched for an input packet, then the first matched rule will be applied. Classes that represent this Secure IP layer are called "SecureIP" classes.

3.2 Design of a firewall

To model firewalls using the SecureIP classes, IP classes in SSFNet must be substituted with the SecureIP classes. But this selection is transparent to simulation application programmers because classes to be used in an IP layer are determined by applications in a lower layer rather than by applications in a higher layer. Therefore, if applications in a lower layer can handle these changes of IP classes dynamically, there is nothing needed to be done in applications for simulations. The following is the definition of a firewall in Domain Modeling Language(DML). Specifications of DML can be found in [10].

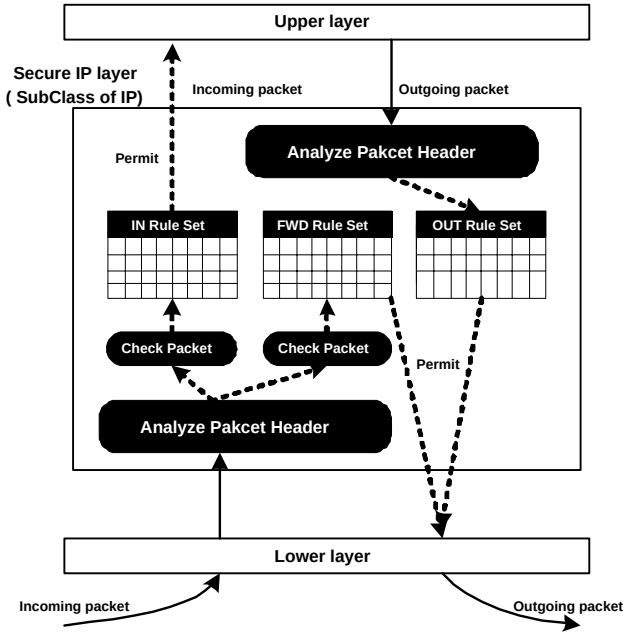

Fig. 1. Packet flow in Secure IP classes

```
ProtocolSession [ name ip use SSF.OS.secureIP
Firewall [
  Rule [
    Type         %S   # Rule Type
    Id           %I   # Rule id
    SrcAddr      %S   # Source Address
    SrcPort      %S   # Source Port
    DestAddr     %S   # Destination Address
    DestPort     %S   # Destination Port
```

```
    ProtocolName %S   # Using Protocol Name (ex: TCP, UDP)
    Direction    %S   # Packet direction ('Outbound' or 'Inbound')
    Action       %S   # Filtering Action ('Permit' or 'Deny')
  ]
]

# Other rules of the protocol go here.
]
```

The value of the 'Type' field can be 'IN', 'OUT', or 'FWD'. If the type of a rule is 'IN', this rule is applied to packets that are sent from other hosts to the server that the firewall is running on. If it is 'FWD', this rule is applied to forwarded packets. Since the main purpose of a firewall is to monitor network activities, 'FWD' rules will be most frequently used. The value of 'Id' is a unique identifier of a rule and it is represented in integer. The 'SrcAddr' field and the 'DestAddr' field are IP addresses of source hosts and destination hosts. The 'SrcPort' field and the 'DestPort' field are port numbers of source hosts and destination hosts. The 'ProtocolName' field specifies the name of a protocol to be used. Currently, 'TCP', 'UDP', and 'ICMP' can be used, and other protocols can be easily supported, if necessary. The 'Direction' field specifies whether a packet comes from outside or inside of the firewall. The 'Action' field can be either 'Permit' or 'Deny.'

3.3 An Example of a firewall

This section shows examples of two filtering rules of a firewall. The following example shows a definition of a router in SSF. In this example, we assume that the internal network has an address range from 210.1.30.1 to 210.1.30.254. The example has a definition of a firewall that has two filtering rules.

```
router [id 1
  interface [ id 0 _extends .dictionary.10Mb]
  interface [ id 1 _extends .dictionary.100Mb]
  graph [

    # Protocol Session Definition.
    # To use a firewall, SSF.OS.SecureIP which is responsible
    # for IP layers must be included.
    ProtocolSession [name ip use SSF.OS.SecureIP

      Firewall[
        Rule[
          # Prevent ftp service in hosts in its own domain
          Type            FWD
          Id              1
          SrcAddr         *
```

```
        SrcPort         *
        DestAddr        210.1.30.*
        DestPort        21
        ProtocolName    TCP
        Direction       *
        Action          Deny
      ]

      Rule[
        # If the source address of an inbound packet is set to
        # an IP address in the same domain of the firewall,
        # the packet will be dropped.
        Type            FWD
        Id              2
        SrcAddr         210.1.30.*
        SrcPort         *
        DestAddr        210.1.30.*
        DestPort        *
        ProtocolName    ICMP
        Direction       Inbound
        Action          Deny
      ]
    ] #end of Firewall
  ] #end of ProtocolSession ip

  ProtocolSession [name icmp use SSF.OS.OSPF]

] #end of graph
] #end of router
```

Between two filtering rules, the first rule disallows ftp services to internal servers, i.e. servers with ip addresses of 210.1.30.*. In other words, this rule drops packets sent to port number 21 of hosts 210.1.30.* using a TCP protocol. The second rule restricts ICMP packets that are widely used for DOS attacks. If attackers change source addresses of ICMP packets to internal addresses, routers regard these packets as internal packets because domain addresses between source and destination are same. This attack tries to use a potential security hole of firewalls that allows data transfer within internal hosts. To prevent this kind of attacks, the second rule of the above example drops inbound packets with internal source addresses.

4 Packet Manipulator

SSFNet provides only basic socket APIs to send or receive TCP or UDP packets, and there is no API that allows modifying packet headers. But it is essential

for an attack simulator to modify packet headers. Therefore, we designed and developed a packet manipulator for SSFNet.

Packet Manipulator (PM) has two main functionalities.

- Packet manipulation: This functionality is critical to develop attack programs for simulation.
- Packet capturing: to intercept packets from other hosts.

PM provides socket APIs, such as bind(), listen(), connect(), accept(), write(), read(), and the like, that use to communicate with other hosts in TCP or UDP. To simulate network attacks or defense mechanisms, PM provides APIs to set or modify IP headers, like a version number, IP header length, types of service, total length of pack, identification, flags, fragment offsets, time to live, protocols, header checksums, a source IP address, and a destination IP address, etc. Using these APIs, IP packets are easily manipulated and sent to other servers by attack programs. PM can also generate complete TCP packets and manipulate them. UDP packets and ICMP packets can be handled by PM, too. The manipulated packets must be transferred using APIs provided by only PM. Because attack programs need to be able to receive response packets of attacks, PM also provides packet-capturing facility.

Figure 2 shows relationship among packet generation classes in PM. To make new packets, the IpHeader class generates IP headers and sends them to appropriate classes depending on protocol types. Each protocol class generates its own header and inserts it into the IP payload part of the packet, and a complete IP packet is generated. This newly generated IP packet is sent to a remote host using methods in the PM class.

The RawIpPkt class generates IP headers that can be used in real networks and it allows programmers to be able to set parameters of TCP/IP or UDP/IP. IP headers can be generated as follows:

```
RawIpPkt r_ip = new RawIpPkt();
r_ip.MakeIpPkt();
```

ICMP packets can be generated by the RawIcmpPkt class. Users can define parameters for ICMP packets and the generated ICMP header is added to an IP header. Methods to generated ICMP packets are as follows:

```
RawIcmpPkt r_icmp = new RawIcmpPkt();
r_icmp.MakeIcmpPkt(r_ip);
```

The RawTcpPkt class is used to generate TCP packets. The generated TCP header is added to an IP header. Methods of the RawTcpPkt class are used as follows:

```
RawTcpPkt r_tcp = new RawTcpPkt();
r_tcp.MakeTcpPkt(r_ip);
```

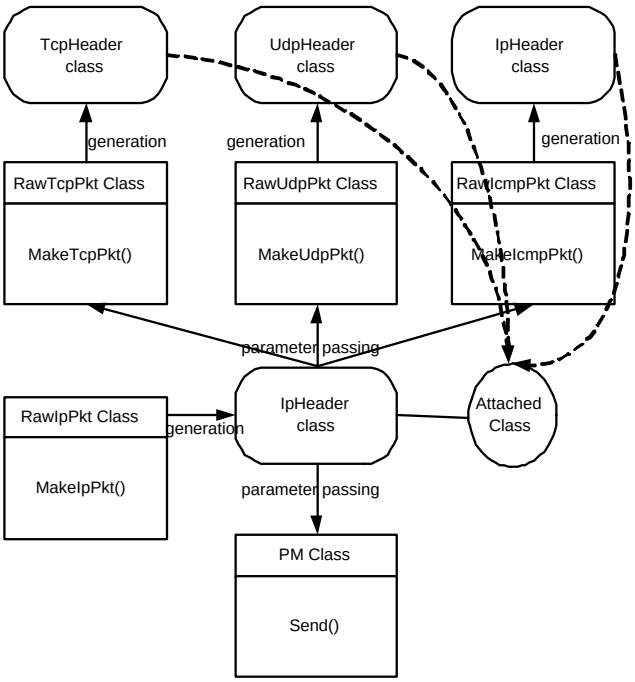

Fig. 2. Classes and their relationship in Packet Manipulator

The RawUdpPkt class is similar to other classes. Its methods are used as follows:

```
RawUdpPkt r_udp = new RawUdpPkt();
r_udp.MakeUdpPkt(r_ip);
```

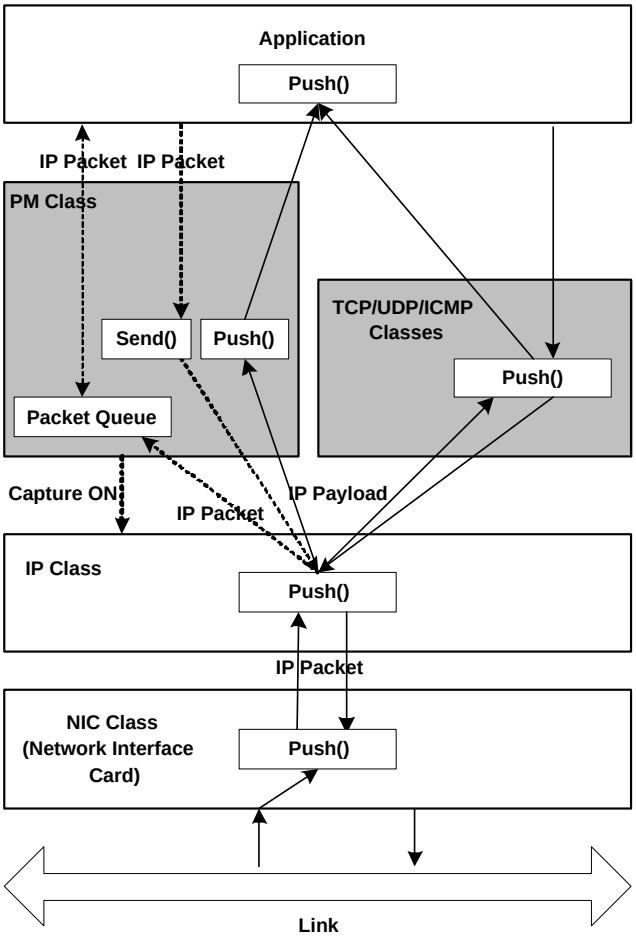

Fig. 3. Packet flows of a normal mode and a packet-capturing mode

The IP packets that are generated with the above methods are sent using the following methods.

```
PM pm = new PM(this);
```

```
pm.Send( r_ip, this );
```

Figure 3 shows packet flows of both a normal mode and a packet-capturing mode. Solid lines represent packet flows in a normal mode and dotted lines show packet flows in a packet-capturing mode. Even though the packet-capturing mode is set to 'OFF', received packets are passed to applications through a push() method in PM.

4.1 An Example Program using PM

The next program shows a simulation code of a smurf attack using APIs provided by PM. In the "smurf" attack, attackers are using ICMP echo request packets directed to IP broadcast addresses from remote locations to generate denial-of-service attacks. There are three parties in these attacks: the attacker, the intermediary, and the victim (note that the intermediary can also be a victim). The intermediary receives an ICMP echo request packet directed to the IP broadcast address of their network. Many of the machines on the network will receive this ICMP echo request packet and send an ICMP echo reply packet back. When (potentially) all the machines on a network respond to this ICMP echo request, the result can be severe network congestion or outages. When the attackers create these packets, they do not use the IP address of their own machine as the source address. Instead, they create forged packets that contain the spoofed source address of the attacker's intended victim. The result is that when all the machines at the intermediary's site respond to the ICMP echo requests, they send replies to the victim's machine. The victim is subjected to network congestion that could potentially make the network unusable [11].

In the next program, the address of a target host and a broadcast address of a network are acquired from a DML file. A DML file also provides the number of packets to be transmitted, an interval between packet transmissions, the size of a packet, and so on. The program creates an IP header, makes up an ICMP message, and sends the created packet to 'bcastaddr'.

```
/* define packet manipulatior instance */
PM pm = new PM(this);

/* get following items from DML file
target = get target host IP address to hit
bcastaddr = get network-directed broadcast address(x.x.x.255)
num = get number of packets to send (0 = flood)
pktdelay = get packet delay(wait) between each packet (in ms)
pktsize = get packet size ;

/* Using Timer class, we will send the packet periodically */
sequence = 1;
(new Timer(inGraph(),Net.seconds(pktdelay)) {
  public void callback() {
```

```
        try {
            /* make RawIpPkt instance */
            RawIpPkt r_ip = new RawIpPkt();

            /* Here, you must set RawIpPkt class fields */
            /* total length, src_ip, dest_ip, protocol_no,
               tos, ttl */
            IpHeader ip_h = r_ip.MakeIpPkt();

            /* make RawIcmpPkt instance */
            RawIcmpPkt r_icmp = new RawIcmpPkt;

            /* Here, you must set RawIcmpPkt class fields */
            r_icmp.icmpType = ICMP.ICMP_ECHO_REQUEST;
            r_icmp.id = 1; r_icmp.seq = sequence++; r_icmp.datalen=pkt_size;

            r_icmp.MakeIcmpPkt(ip_h);

            /* send this packet to remote hosts */
            pm.send(ip_h, this);
            num --;
            if ( num > 0 )
                /* setting Timer again */
                set();
        } catch (ProtocolException pex) {
            pex.printStackTrace();
        }
    }
  }).set();

} /* end of methods */

public boolean push(ProtocolMessage msg, ProtocolSession from)
{
   // you can implement this method optionally.
}
```

5 Experiments and Analysis

To test firewalls and Packet Manipulator proposed here, in this chapter we modeled and established a virtual communication network using SSF, and on top of this network we performed simulations of network attacks. Based on the network model in [2], our network model has 13000 clients, 40 servers, and 270 routers. Other network parameters are same as those in [2].

Using the above network configuration, 'smurf' attack simulations are done as follows. An 'smurf' attack is a kind of distributed denial of service attacks [12, 13]. An attacker sends ICMP echo request packets with a victim's address as a source address, and hosts that receive ICMP echo requests send responses for ICMP echo request packets to the victim. Since there is no mechanism to measure loads of a specific server in SSFNet, we measured response time of the victim to find out effects of the attack. To measure response time, a host which does not take part in the attack sends normal ping packets to the victim. DML codes for the 'smurf' attack are as follows. It is assumed that the HTTP protocol is used to access the victim.

```
Client10Mb_attack [
    interface [id 0 bitrate 10000000 latency 0.001]
    graph [
        ProtocolSession [name ip use SSF.OS.IP]
        ProtocolSession [name icmp use SSF.OS.ICMP]
        ProtocolSession [name pingerBroad use pingClientBroad
                         count 500 interval 0.1 len 1024 ]
    ] # End of graph
] # End of client
```

Figure 4 shows average response time of the victim with different numbers of intermediate subnetworks that participate in this attack. The size of an attack packet is 1024 bytes, and the size of a ping packet is 48 bytes. 500 packets are sent from each host in subnetworks in every 0.1 second, and average response time was measured. One subnetwork has 100 hosts, and PM was used to generate attack packets. Without any DDoS attack, the average response time for ping packets is 0.79 second, and as shown in Figure 6, the average response time was increased drastically when the number of input packets is over the capacity of the victim. In this simulation, it was when the number of subnetworks was over 12, i.e. when about 1200 ICMP packets are sent.

Figure 5 shows the change of average response time of the target server as the size of attack packets increases. The number of subnetworks was 12, and other parameters are same as the previous simulation. If the size is greater than a certain value, the average response time is same. This is because the number of packets processed by a bottlenecked router is same, i.e. the router is fully used. But as the size of attack packets increases, the period that a router is bottlenecked becomes longer.

6 Conclusions

Cyber attacks affect many aspects of networks, such as network latency, server loads, data integrity of files and so on. One of the best feasible ways to research cyber attacks and their consequences is simulations. To simulate cyber attacks, a simulation model must be able to describe various network components as well as cyber attacks and security mechanisms.

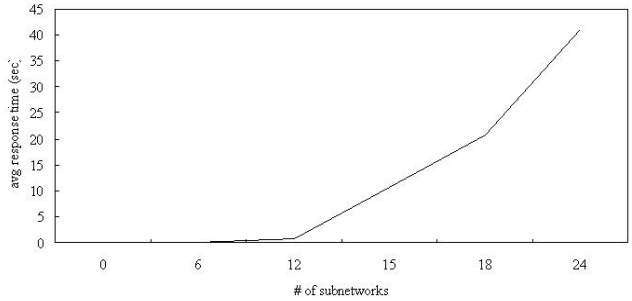

Fig. 4. Average response time of ping packets with different number of subnetworks

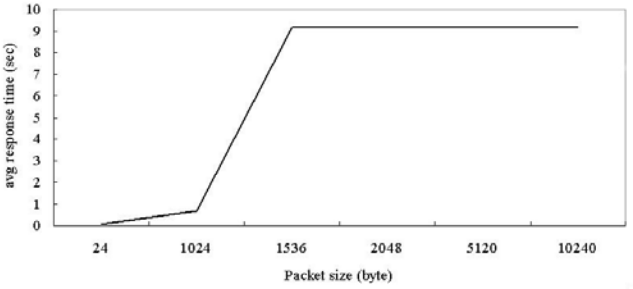

Fig. 5. Average response time of ping packets with different sizes of attack packets

We designed and added new modules to support firewalls and packet manipulation. The newly designed modules are programmed in Java and added to SSFNet. Since a large portion of cyber attack applications are programmed in Java, these modules can be easily incorporated with attack applications to simulate their effects on networks. Using the modules introduced in this paper, more diverse simulations of attacks and security policies become possible.

To test and verify the newly added modules, the 'smurf' attack was simulated on virtual networks that we composed. The simulation results show that the modules proposed here are quite satisfactory in large network simulations. Besides the 'smurf' attack, other network attacks that use packet manipulation can be done with new modules.

As future directions, there should be more researches on how to simulate cyber attacks and network damages using these newly added modules in SSFNet.

References

1. Cowie, J.H., Nicol, D.M., Ogielski, A.T.: Modeling the global internet. Computing in Science and Engineering (1999) 42–50
2. Cowie, J.H., Nicol, D.M., Ogielski, A.T.: Modeling 100,000 nodes and beyond: Self-validating design. In: DARPA/NIST Workshop on Validation of Large Scale Network Simulation Models. (1999)
3. Cowie, J., Liu, H., Liu, J., Nicol, D., Ogielski, A.: Towards realistic million-node internet simulations. In: Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada (1999)
4. Vigna, G., Eckmann, S., Kemmerer, R.: The STAT tool suite. In: Proceedings of the DISCEX 2000, Hilton Head, South Carolina, IEEE Computer Society Press (2000)
5. Vigna, G., Kemmerer, R.A.: NetSTAT: A network-based intrusion detection system. Journal of Computer Security **7** (1999) 37–71
6. Networks, S.: Custom Attack Simulation language(CASL). (1998)
7. Brett: Building bastion routers using Cisco IOS. In: Phrack Magazine. Volume 9. phrack.org (1999)
8. Durst, R., Champion, T., Witten, B., Miller, E., spagnuolo, L.: Testing and evaluating computer intrusion detection systems. Communications of the ACM **42** (1999) 53–61
9. Ilgun, K.: USTAT: A real-time intrusion detection system for UNIX. In: IEEE Symposium on Security and Privacy, Oakland, CA (1993)
10. SSF Research Network http://www.ssfnet.org/SSFdocs/dmlReference.html: (Domain Modeling Language Reference Manual)
11. Center, C.C.: CERT advisories, (http://www.cert.org/advisories/)
12. Geng, X., Whinston, A.B.: Defeating distributed denial of service. In: Proceedings of the IT Professional. Volume 2 of 4. (2000) 36–42
13. Kaufman, S., Ying, S.: DARPA information assurance program experimental confirmation-DDoS. In: Proceedings of the DARPA Information Survivability Conference & Exposition 2001. (2001) 152–154

A Comparison of Publicly Available Tools for Static Intrusion Prevention

John Wilander and Mariam Kamkar

Dept. of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping
Sweden
{johwi, marka}@ida.liu.se
http://www.ida.liu.se/~johwi

Abstract. The size and complexity of today's software systems is growing, increasing the number of bugs and thus the possibility of security vulnerabilities. Two common attacks against such vulnerabilities are buffer overflow and format string attacks. In this paper we implement a testbed of 44 function calls in C to empirically compare five publicly available tools for static analysis aiming to stop these attacks. The results show very high rates of false positives for the tools building on lexical analysis and very low rates of true positives for the tools building on syntactical and semantical analysis. ...

Keywords: security intrusions, intrusion prevention, static analysis, security testing, buffer overflow, format string attack

1 Introduction

As our software systems are growing larger and more complex the amount of bugs increase. Many of these bugs constitute security vulnerabilities. According to statistics from CERT Coordination Center at Carnegie Mellon University the number of reported security vulnerabilities in software has increased with nearly 500% in two years [5].

Now there is good news and bad news. The good news is that there is lots of information out there on how these security vulnerabilities occur, how the attacks against them work and most importantly how they can be avoided. The bad news is that this information apparently does not lead to less vulnerabilities. The same mistakes are made over and over again which for instance is shown in the statistics for the infamous *buffer overflow* vulnerability. David Wagner et al from University of California at Berkeley show that buffer overflows alone stand for about 50% of the vulnerabilities reported by CERT [33]. Equally dangerous is the *format string* vulnerability which was publicly unknown until 2000.

In the middle of January 2002 the discussion about responsibility for security intrusions took an interesting turn. The US National Academies released

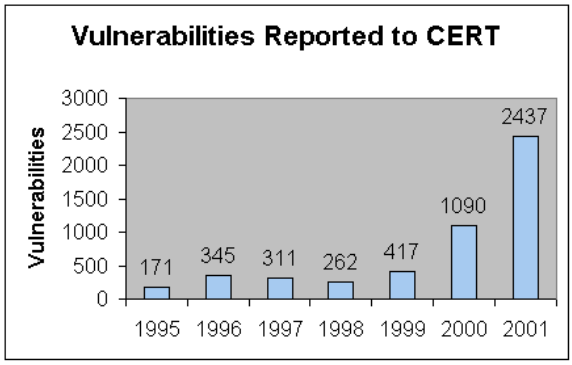

Fig. 1. Software vulnerabilities reported to CERT 1995–2001.

a prepublication recommending policy-makers to create laws that would hold companies accountable for security breaches resulting from vulnerable products [24] which got global media attention [3, 20]. So far, only the intruder can be charged in court. In the future software companies may be charged for not preventing intrusions. This stresses the importance of helping software engineers to produce more secure software. Automated development and testing tools aimed for security could be one of the solutions for this growing problem.

A good starting point would be tools that can be applied directly to the source code and solve or warn about security vulnerabilities. This means trying to solve the problems in the implementation and testing phase. Applying security related methodologies throughout the whole development cycle would most probably be more effective, but given the amount of existing software, the strive for modular design reusing software components, and the time it would take to educate software engineers in secure analysis and design, we argue that security tools trying to clean up vulnerable source code are necessary. A further discussion on this issue can be found in the January/February 2002 issue of IEEE Software [13].

In this paper we investigate the effectiveness of five publicly available static intrusion prevention tools—namely the security testing tools ITS4, Flawfinder, RATS, Splint and BOON. Our approach has been to first get an in-depth understanding of how buffer overflow and format string attacks work and from this knowledge build up a testbed with identified security bugs. We then make an empirical test with our testbed. This work is a follow-up of John Wilander's Master's Thesis [36].

The rest of the paper is organized as follows. Section 2 describes process memory management in UNIX and how buffer overflow and format string attacks work. Here we define our testbed of 23 vulnerable functions in C. Section 3 presents the concept of intrusion prevention and describes the techniques used in the five analyzed tools. Section 4 presents our empirical comparison of the

tools' effectiveness against the previously described vulnerabilities. Related work is presented in section 5. Finally section 6 contains our conclusions.

2 Attacks and Vulnerabilities

The analysis of intrusions in this paper concerns a subset of all violations of security policies that would constitute a security intrusion according definitions in for example the Internet Security Glossary [27]. In our context an intrusion or a successful attack aims for *changing the flow of control*, letting the attacker execute arbitrary code. Software security bugs, or *vulnerabilities*, allowing these kind of intrusions are considered the worst possible since "arbitrary code" often means starting a new *shell*. This shell will have the same access rights to the system as the process attacked. If the process had *root access*, so will the attacker in his or her new shell, leaving the whole system open for any kind of manipulation.

2.1 Changing the Flow of Control

Changing the flow of control and executing arbitrary code involves two steps for an attacker:

1. Injecting *attack code* or *attack parameters* into some memory structure (e.g. a buffer) of the vulnerable process.
2. Abusing some vulnerable function writing to memory of the process to alter data that controls execution flow.

Attack code could mean assembly code for starting a shell (less than 100 bytes space will do) whereas attack parameters are used as input to code already existing in the vulnerable process, for example using the parameter `"/bin/sh"` as input to the `system()` library function would start a shell.

Our biggest concern is step two—redirecting control flow by writing to memory. That is the hard part and the possibility of changing the flow of control in this way is the most unlikely condition of the two to hold. The possibility of injecting attack code or attack parameters is higher since it does not necessarily have to violate any rules or restrictions of the program.

Changing flow of control is made by altering a *code pointer*. A code pointer is basically a value which gives the *program counter* a new memory address to start executing code at. If a code pointer can be made to point to attack code the program is vulnerable. The most popular code pointer to target is the return address on the stack. But programmer defined *function pointers*, so called *longjmp buffers*, and the old *base pointer* are equally effective targets of attack.

2.2 Buffer Overflow Attacks

Buffer overflow attacks are the most common security intrusion attack [33, 11] and has been extensively analyzed and described in several papers and on-line

documents [22, 17, 8, 6]. Buffers, wherever they are allocated in memory, may be overflown with too much data if there is no check to ensure that the data being written into the buffer actually fits there. When too much data is written into a buffer the extra data will "spill over" into the adjacent memory structure, effectively overwriting anything that was stored there before. This can be abused to overwrite a code pointer and change the flow of control.

The most common buffer overflow attack is shown in the simplified example below. A local buffer allocated on the stack is overflown with 'A's and eventually the return address is overwritten, in this case with the address `0xbffff740`.

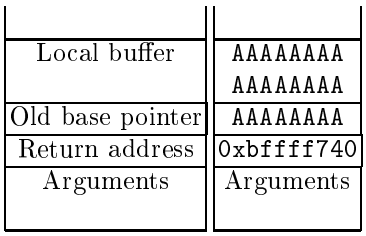

Fig. 2. A buffer overflow overwriting the return address.

If an attacker can supply the input to the buffer he or she can design the data to redirect the return address to his or her attack code.

2.3 Buffer Overflow Vulnerabilities

So how come there is no check whether the data fits into the destination buffer? The problem is that several of ANSI C's standard library functions rely on the programmer to do the checking, which they often do not. Many of these functions are powerful for handling strings and thus popular. More secure versions have in some cases been implemented but are not always know by programmers. There are lists of these dangerous C functions often involved in published buffer overflows [35, 30, 31]. From these lists we have chosen to take the fifteen functions considered most risky into our testbed:

1. `gets()`
2. `cuserid()`
3. `scanf()`
4. `fscanf()`
5. `sscanf()`
6. `vscanf()`
7. `vsscanf()`
8. `vfscanf()`
9. `sprintf()`
10. `strcat()`
11. `strcpy()`
12. `streadd()`
13. `strecpy()`
14. `vsprintf()`
15. `strtrns()`

This list is not exhaustive but should provide useful test data for our comparison of the tools.

2.4 Format String Attacks

22nd of June 2000 the first *format string attack* was published [29]. Comments in the exploit source code dates to the 15th of October 1999. Until then this whole category of security bugs was publicly unknown. Since then format string attacks have been acknowledged for being as dangerous as buffer overflow attacks. They are described in an extensive article by Team Teso [25] and also in a shorter article by Tim Newsham [21].

String functions in ANSI C often handle so called *format strings*. They allow for dynamic composition or formatting of strings using *conversion specifications* starting with the character `%` and ending with a conversion specifier. Each conversion specification results in fetching zero or more subsequent arguments.

Let's say a part of a program looks like this:

```
void print_function_1(char *string) {
  printf("%s", string); }
```

A call to `print_func_1()` would print the string argument passed. The same functionality could (seemingly) be achieved with somewhat simpler code:

```
void print_function_2(char *string) {
  printf(string); }
```

Using the function argument `string` directly will still print the argument passed to `print_function_2()`. But what if we call `print_function_2()` with a string containing conversion specifications, for example `print_function_2("%d %d%d%d")`? Then `printf()` will interpret the string as a format string and in this case assume that there are four integers stored on the stack and thus pop four times four bytes of stack memory and print the values stored there. So if programmers take this shortcut when using format string functions, the possibility arises for an attacker to inject conversion specifications that will be evaluated.

Now, considering the conversion specifier `%n` things get dangerous. `%n` will cause the format string function to pop four bytes of the stack and use that value as a memory pointer for storing the number of characters so far in the format string (i.e. the number of characters before `%n`.). So by injecting a format string containing `%n` an attacker can *write* data into the process' memory.

If an attacker is able to provide the format string to a an ANSI C format function in part or as a whole a format string vulnerability is present. By combining the various conversion specifications and making use of the fact that the format string itself is stored on the stack we can view and write on arbitrary memory addresses.

2.5 Format String Vulnerabilities

While the `scanf()`-family is involved in numerous of buffer overflow exploits [1] the format string attacks published concern the `printf()`-family of format string functions [25, 7]. For that reason our test only concerns the latter subset

of the ANSI C format functions. So we add another eight function calls to our testbed (`sprintf()` and `vsprintf()` are used differently here than in the buffer overflow case):

16. `printf()`
17. `fprintf()`
18. `sprintf()`
19. `snprintf()`
20. `vprintf()`
21. `vfprintf()`
22. `vsprintf()`
23. `vsnprintf()`

3 Intrusion Prevention

There are several ways of trying to prohibit intrusions. Halme and Bauer present a taxonomy of *anti-intrusion techniques* called *AINT* [14] where they define:

Intrusion prevention. Precludes or severely handicaps the likelihood of a particular intrusion's success.

We divide intrusion prevention into *static intrusion prevention* and *dynamic intrusion prevention*. In this section we will first describe the differences between these two categories. Secondly, we describe five publicly available tools for static intrusion prevention, describe shortly how they work, and in the end compare their effectiveness against vulnerabilities described in section 2.2. This is not a complete survey of static intrusion prevention tools, rather a subset with the following constraints:

- Tools used in the testing phase of the software.
- Tools that require no altering of source code to detect security vulnerabilities.
- Tools that are implemented and publicly available, not system specific tools.

Our motivation for this is to evaluate and compare tools that could easily and quickly be introduced to software developers and increase software quality from a security point of view.

3.1 Dynamic Intrusion Prevention

The dynamic or *run-time* intrusion prevention approach is to change the run-time environment or system functionality making vulnerable programs harmless or at least less vulnerable. This means that in an ordinary environment the program would still be vulnerable (the security bugs are still there) but in the new, more secure environment those same vulnerabilities cannot be exploited in the same way—it protects *known* targets from attacks. Their general weakness lies in the fact that the protection schemes all depend on how bugs are known to be exploited today, but they do not get rid of the actual bugs. Whenever an attacker has figured out a new attack target reachable with the same security bug, these dynamic solutions often stand defenseless. On the other hand they will be effective against exploitation of any new bugs aiming for the same target.

3.2 Static Intrusion Prevention

Static intrusion prevention tries to prevent attacks by finding the security vulnerabilities in the source code so that the programmer can remove them. Removing all security bugs from a program is considered infeasible [16] which makes the static solution incomplete. Nevertheless, removing bugs known to be exploitable brings down the likelihood of successful attacks against all possible security targets in the software. Static intrusion prevention removes the attackers tools, the security bugs. The two main drawbacks of this approach is that someone has to keep an updated database over programming flaws to test for, and since the tools only *detect* vulnerabilities the user has to know how to fix the problem once a warning has been issued. In this paper we have chosen to focus on five publicly available tools for static intrusion prevention.

3.3 ITS4

In late 2000 researchers at Reliable Software Technologies, now Cigital, presented a static analysis tool for detecting security vulnerabilities in C and C++ code—*It's the Software Stupid! Security Scanner* or *ITS4* for short [30]. The tool does a lexical analysis building a token stream of the code. Then the tokens are matched with known vulnerable functions in a database. The reason for not performing a deeper analysis with the help of syntactic analysis (parsing) is that such an analysis cannot be made on the fly during programming. ITS4 is built to give developers support while coding, highlighting potential security problems as they are written. Parsing also suffers from being build dependent, not always covering the whole source code because of pre-processor conditionals.

When writing their paper the vulnerability database contained 131 potential vulnerabilities including problems with *race conditions* (not included in this paper, for reference see article by Bishop and Dilger [2]) and buffer overflows. *Pseudo random functions* are also considered risky since they're often used wrongly in security-critical applications. An entry in the database consists of:

- A brief description of the problem
- A high-level description of how to code around the problem.
- A grading of the vulnerability on the scale `NO_RISK`, `LOW_RISK`, `MODERATE_RISK`, `RISKY`, `VERY_RISKY`, `MOST_RISKY`.
- An indication of what type of analysis to perform whenever the function is found.
- Whether or not the function can retrieve input from an external source such as a file or a network connection.

ITS4 has a modular design which allows for integration in various development environments by replacing its front-end or back-end. In fact that was one of the design goals for ITS4. For the moment it only supports integration with GNU Emacs.

The ITS4 security tool is available for download on the Internet.
`http://www.cigital.com/its4/`

3.4 Flawfinder and Rats

Two new security testing tools where released in May 2001—*Flawfinder* developed by David A. Wheeler [34] and *Rough Auditing Tool for Security* (RATS) developed by Secure Software Solutions [28]. They both scan source code on the lexical level, searching for security bugs. Their solutions are very similar to ITS4. When it was noticed that the two teams where developing similar tools they decided on a common release date and on trying to combine the two tools into one in the future.

Just as ITS4 Flawfinder works by using a built-in database of C/C++ functions with well-known problems, such as buffer overflow risks, format string problems, race conditions, and more. The tool produces a list of potential vulnerabilities sorted by risk. This risk level depends not only on the function, but on the values of the parameters of the function. For example, constant strings are considered less risky than fully variable strings. The Flawfinder 0.19 vulnerability database contains 55 C security bugs.

RATS scans not only C and C++ code but also Perl, PHP and Python source code and flags common security bugs such as buffer overflows and race conditions. Just as Flawfinder and ITS4, RATS has a database of vulnerabilities and sorts found security bugs by risk. The RATS 1.3 vulnerability database contains 102 C security bugs.

Both these security testing tools are invoked from a shell with source code as input. They traverse the code and produce output with risk grading and short descriptions of the potential problems.

The security tools Flawfinder and RATS are available for download on the Internet.

`http://www.dwheeler.com/flawfinder/`

`http://www.securesw.com/rats/`

3.5 Splint

The next static analysis tool we describe is *LCLint* implemented by David Evans et al [10, 23]. The name and some of its functionality originates from a popular static analysis tool for C called *Lint* released in the seventies [15]. LCLint has later been enhanced to search for security specific bugs [16] and the first of January 2002 LCLint got the name *Secure Programming Lint* or *Splint* for short.

The Splint approach is to use programmer provided semantic comments, so called *annotations*, to perform static analysis on the syntactic level, making use of the program's parse tree. This means that the tool has a much better chance of differentiating between correct and incorrect use of functions than the tools working on the lexical level.

The annotations specify function constraints in the program—what a function *requires* and *ensures*. Here is a simplified example from the annotated library `standard.h` in the Splint package:

```
char *strcpy (char *s1, char *s2)
      /*@requires maxSet(s1) >= maxRead(s2) @*/
      /*@ensures  maxRead(s1) == maxRead (s2) @*/
```

The `requires` clause specifies that buffer `s1` must be big enough to hold all characters readable from buffer `s2`. The `ensures` clause says that, upon return, the length of buffer `s1` is equal to the length of buffer `s2`. If a program contains a call to `strcpy()` with a destination buffer `s1` smaller than the source buffer `s2`, a buffer overflow vulnerability is present and Splint should report the bug.

To detect bugs the constraints in the annotations have to be resolved. Low level constraints are first *generated* at the subexpression level (i.e. they are not defined by annotations). Then statement constraints are generated by cojoining these subexpression constraints, assuming that two different subexpressions cannot change the same data. The generated constraints are then matched with the annotated constraints to determine if the latter hold. If they do not Splint issues a warning.

Note that we will not add any annotations to our test source code since that would be a violation of the second testing constraint defined in section 3. We rely fully on Splint's annotated libraries to make a fair comparison.

The Splint security tool is available for download on the Internet.
`http://www.splint.org/`

3.6 BOON

David Wagner et al presented a tool in 2000 describing aiming for detecting buffer overflow vulnerabilities in C code [33]. In July 2002 their tool, or rather working prototype, was publicly released under the name *BOON* which stands for *Buffer Overrun detectiON*. Under the assumption that most buffer overflows are in string buffers they model string variables (i.e. the string buffers) as two properties—the allocated size, and the number of bytes currently in use. Then all string functions are modeled in terms of their effects on these two properties of the string variable. The constraints are solved and matched to detect inconsistencies similarly to Splint.

Before analyzing the source code you have to use the C preprocessor on it to expand all macros and #include's. Then BOON parses the code and reports any detected vulnerabilities as belonging to one of three categories, namely "Almost certainly a buffer overflow", "Possibly a buffer overflow" and "Slight chance of a buffer overflow". The user needs to go check the source code by hand and see whether it is a real buffer overflow or not. Note that BOON does not detect format string vulnerabilities and is thus not tested for that.

The BOON security tool is available for download on the Internet.
`http://www.cs.berkeley.edu/~daw/boon/`

3.7 Other Static Solutions

There are several other approaches to static intrusion prevention. The area connects to general software testing which provides a broad range of potential methodologies.

A tool yet to be published is *Czech* by Jose Nazario [18]. Czech is a C source code checking tool that will do full out static analysis and variable tainting.

Software Fault Injection A technique originally used in hardware testing called *fault injection* has also been used to find errors in software [32]. This has been used for security testing. By injecting faults, the system being tested is forced into an anomalous state during execution and the effects on system security is observed and evaluated.

Anup Ghosh et al implemented a prototype tool called *Fault Injection Security Tool*, or FIST for short [12]. The tool shows promising results but preparations of the source code have to be made by hand which means that the process is not automated. Also FIST is not available for download so we have excluded it from our analysis.

Also Wenliang Du and Aditya Mathur have done research on software fault injection for security testing [9]. They inject faults from the environment of the application, i.e. anomalous user input, erroneous environment variables and so on. In their paper they describe a methodology not yet implemented. Therefore their approach is not part of our analysis.

Constraint-Based Testing Umesh Shankar et al from University of California at Berkeley present an interesting solution to finding format string vulnerabilities [26]. They add a new C type qualifier called *tainted* to tag data that has originated from an untrustworthy source. Then they set up typing rules so that tainted data will be propagated, keeping its tag. If tainted data is used as a format string the tester is warned of the possible vulnerability. Sadly, we did not manage to get their tool to report any vulnerabilities with the supplied annotated library functions.

4 Comparison of Static Intrusion Prevention Tools

Our testbed contains 20 vulnerable functions chosen from ITS4's vulnerability database (category `RISKY` to `MOST_RISKY`), Secure programming for Linux and UNIX HOWTO [35], and the whole `[fvsn]printf()`-family (see section 2.3 and 2.5 for a complete list). We do not claim that this test suite is perfectly fair, nor complete. But the sources from where we have chosen the vulnerabilities seem reasonable and the test result will at least provide us with an interesting comparison. Our 20 vulnerable functions are used in 13 safe buffer writings, 15 unsafe buffer writings, 8 safe format string calls and 8 unsafe format string calls, in total 44 function calls. We did not go into complex constructs to implement

	Flawfinder	ITS4	RATS	Splint	BOON *
True Positives	22 (96%)	21 (91%)	19 (83%)	7 (30%)	4 (27%)
False Positives	15 (71%)	11 (52%)	14 (67%)	4 (19%)	4 (31%)
True Negatives	6 (29%)	10 (48%)	7 (33%)	17 (81%)	9 (69%)
False Negatives	1 (4%)	2 (9%)	4 (17%)	16 (70%)	11 (73%)

Table 1. Overall effectiveness and accuracy of static intrusion prevention. "Positive" means a warning was issued, "Negative" means no warning was issued. In total 44 function calls, 23 unsafe and 21 safe. * BOON only tested with buffer overflow vulnerabilities.

the safe function calls, rather a straight forward solution. An example of the difference between safe and unsafe calls is shown below:

```
char buffer[BUFSIZE];

if(strlen(input_string)<BUFSIZE)
  strcpy(buffer, input_string); /* Safe */
strcpy(buffer, input_string);   /* Unsafe */
```

Overall results from our tests is presented in table 1 and detailed results are presented in table 2. The source code in short form can be found in Appendix A. The exact source code and the print-outs from the various testing tools can be found on our homepage:

`http://www.ida.liu.se/~johwi`

4.1 Observations and Conclusions

As you would think all three lexical testing tools ITS4, Flawfinder and RATS, perform about the same on the true positive side. After all, a great part of our tested vulnerabilities where found in their databases or in publications connected to them, as stated before. But they differ considerably on the false positives where ITS4 is best.

For security aware programmers with knowledge of how buffer overflow and format string attacks work these tools can be very helpful. They will most probably get minor testing output, be able to sort out what is important and most importantly know how to solve the reported problems. For less experienced programmers the output might be too large and since the tools give no instructions on how to solve the problems they will need some other form of help.

Quite interesting is that Splint and BOON finds so few bugs. We contacted Splint author David Larochelle concerning this and he responded that the undetected bugs where not considered a serious threat since they are known to the security community and easily found with the UNIX command `grep`. We disagree with him—why not detect as many security bugs as possible? And why not help the developers that are not aware of the security vulnerabilities coming from misuse of several C functions?

Vulnerable Function	Flawfinder		ITS4		RATS		Splint		BOON	
	True	False	True	False	True	False	True	False	True	False
gets()	1	-	1	-	1	-	1	-	1	-
scanf()	1	0	1	0	1	1	0	0	0	0
fscanf()	1	0	1	0	1	1	0	0	0	0
sscanf()	1	0	1	0	1	1	0	0	0	0
vscanf()	1	0	1	0	1	1	0	0	0	0
vsscanf()	1	0	1	0	1	1	0	0	0	0
vfscanf()	1	0	1	0	1	1	0	0	0	0
cuserid()	0	-	1	-	1	-	0	0	0	0
sprintf()	1	1	1	0	1	1	0	0	1	1
strcat()	1	1	1	1	1	1	1	0	1	1
strcpy()	1	1	1	1	1	1	1	0	1	1
streadd()	1	1	1	1	1	0	0	0	0	0
strecpy()	1	1	1	1	1	0	0	0	0	0
vsprintf()	1	1	1	0	1	1	1	1	0	0
strtrns()	1	1	1	1	1	0	0	0	0	0
printf()	1	1	1	1	1	1	1	1	-	-
fprintf()	1	1	1	1	1	1	1	1	-	-
sprintf()	1	1	1	1	1	1	1	1	-	-
snprintf()	1	1	1	1	0	0	0	0	-	-
vprintf()	1	1	0	0	0	0	0	0	-	-
vfprintf()	1	1	0	0	0	0	0	0	-	-
vsprintf()	1	1	1	1	1	1	0	0	-	-
vsnprintf()	1	1	1	1	0	0	0	0	-	-

Table 2. Detailed effectiveness and accuracy of intrusion prevention. True = 1 means an unsafe call was found, False = 1 means a safe function call was deemed unsafe. "-" means no such test is possible.

Splint is the only tool that can distinguish between safe and unsafe calls to `strcat()` and `strcpy()`. This implicates that Splint has a good possibility to accurately detect security bugs with a low rate of false positives, just as you would think considering its deeper analysis of the code.

The general feeling we get after running the constraint-based testing tools is that they are still in some kind of a prototype state. Splint has been around under the name LCLint for some time and is used for general syntactical and semantical testing. But the security part needs to be completed. BOON is published as a prototype and should of course be judged as such.

None of the tools has high enough true positives combined with low enough false positives. Our conclusion is that none of them can really give the programmer peace of mind. And combining their output would be tedious.

5 Related Work

We have found one comparative study made of static intrusion prevention tools—"Source Code Scanners for Better Code" [19] by Jose Nazario. He compares the result from ITS4, Flawfinder and RATS when testing a part of the source code for OpenLDAP known to be vulnerable. It only contains one call to one of our 23 vulnerable functions—`vsprintf()`. No test for false positives is done either.

A study with another focus but relating to ours is "A Comparison of Static Analysis and Fault Injection Techniques for Developing Robust System Services" by Broadwell and Ong [4]. They investigate the strengths of static analysis versus software fault injection in finding errors in several large software packages such as Apache and MySQL. In static analysis they use ITS4 to find race conditions and BOON to find buffer overflows.

6 Conclusions

We have shown that the current state of static intrusion prevention tools is not satisfying. Tools building on lexical analysis produce too many false positives leading to manual work, and tools building on deeper analysis on syntactical and semantical level produce too many false negatives leading to security risks. Thus the main usage for these tools would be as support during development and code auditing, not as a substitute for manual debugging and testing.

References

1. Arash Baratloo, Navjot Singh, and Timothy Tsai. Libsafe: Protecting critical elements of stacks. White Paper `http://www.research.avayalabs.com/project/libsafe/`, December 1999.
2. Matt Bishop and Michael Dilger. Checking for race conditions in file accesses. *Computing Systems*, 2(2):131–152, Spring 1996.
3. Lisa M. Bowman. Companies on the hook for security. `http://news.com.com/2100-1023-821266.html`, January 2002.
4. Pete Broadwell and Emil Ong. A comparison of static analysis and fault injection techniques for developing robust system services. Technical report, Computer Science Division, University of California, Berkeley, `http://www.cs.berkeley.edu/~pbwell/saswifi.pdf`, May 2002.
5. CERT Coordination Center. Cert/cc statistics 1988-2001. `http://www.cert.org/stats/`, February 2002.
6. Matt Conover and w00w00 Security Team. w00w00 on heap overflows. `http://www.w00w00.org/files/articles/heaptut.txt`, January 1999.
7. Crispin Cowan, Matt Barringer, Steve Beattie, Greg Kroah-Hartman, Mike Frantzen, and Jamie Lokier. FormatGuard: Automatic protection from printf format string vulnerabilities. In *Proceedings of the 2001 USENIX Security Symposium*, Washington DC, USA, August 2001.
8. DilDog. The tao of Windows buffer overflow. `http://www.cultdeadcow.com/cDc_files/cDc-351/`, April 1998.

9. Wenliang Du and Aditya P. Mathur. Vulnerability testing of software system using fault injection. COAST, Purdue University, Technical Report 98-02 http://www.cerias.purdue.edu/coast/coast-library.html, April 1998.
10. David Evans, John Guttag, James Horning, and Yang Meng Tan. LCLint: A tool for using specifications to check code. In *Proceedings of the ACM SIGSOFT '94 Symposium on the Foundations of Software Engineering*, pages 87–96, December 1994.
11. David Evans and David Larochelle. Improving security using extensible lightweight static analysis. *IEEE Software*, 19(1):42–51, February 2002.
12. Anup Ghosh, Tom O'Connor, and Gary McGraw. An automated approach for identifying potential vulnerabilities in software. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 104–114, May 1998.
13. Anup K. Ghosh, Chuck Howell, and James A. Whittaker. Building software securely from the ground up. *IEEE Software*, 19(1):14–16, February 2002.
14. Lawrence R. Halme and R. Kenneth Bauer. AINT misbehaving: A taxonomy of anti-intrusion techniques. http://www.sans.org/newlook/resources/IDFAQ/aint.htm, April 2000.
15. S. C. Johnson. Lint, a C program checker. AT&T Bell Laboratories: Murray Hill, NJ. http://citeseer.nj.nec.com/johnson78lint.html, July 1978.
16. David Larochelle and David Evans. Statically detecting likely buffer overflow vulnerabilities. In *Proceedings of the 2001 USENIX Security Symposium*, Washington DC, USA, August 2001.
17. Gary McGraw and John Viega. An analysis of how buffer overflow attacks work. IBM developerWorks: Security: Security articles http://www-106.ibm.com/developerworks/security/library/smash.html?dwzon%e=security, March 2000.
18. Jose Nazario. Project pedantic—source code analysis tool(s). http://pedantic.sourceforge.net/, March 2002.
19. Jose Nazario. Source code scanners for better code. The Linux Journal http://www.linuxjournal.com/article.php?sid=5673, January 2002.
20. BBC News. Software security law call. http://news.bbc.co.uk/hi/english/sci/tech/newsid_1762000/1762261.stm, January 2002.
21. Tim Newsham. Format string attacks. White Paper http://www.guardent.com/rd_whtpr_formatNewsham.html, September 2000.
22. Aleph One. Smashing the stack for fun and profit. http://immunix.org/StackGuard/profit.html, November 1996.
23. C E Pramode and C E Gopakumar. Static checking of C programs with LCLint. *Linux Gazette*, 51 http://www.linuxgazette.com/issue51/pramode.html, March 2000.
24. Computer Science and National Research Council Telecommunications Board. Cybersecurity today and tomorrow: Pay now or pay later (prepublication). Technical report, National Academies, USA, http://www.nap.edu/books/0309083125/html/, January 2002.
25. Scut and Team Teso. Exploiting format string vulnerabilities. http://teso.scene.at/articles/formatstring/, September 2001.
26. Umesh Shankar, Kunal Talwar, Jeffrey S. Foster, and David Wagner. Automated detection of format-string vulnerabilities using type qualifiers. In *Proceedings of the 10th USENIX Security Symposium*, http://www.cs.berkeley.edu/~ushankar/, August 2001.
27. Robert W. Shirey. Request for comments: 2828, Internet security glossary. http://www.faqs.org/rfcs/rfc2828.html, May 2000.

28. Secure Software Soliutions. Rough auditing tool for security, RATS 1.3. `http://www.securesw.com/rats/`, September 2001.
29. tf8. Bugtraq id 1387, Wu-Ftpd remote format string stack overwrite vulnerability. `http://www.securityfocus.com/bid/1387`, June 2000.
30. John Viega, J.T. Bloch, Tadayoshi Kohno, and Gary McGraw. ITS4: A static vulnerability scanner for C and C++ code. In *Proceedings of the 16th Annual Computer Security Applications Conference*, December 2000.
31. John Viega and Gary McGraw. *Building Secure Software : How to Avoid Security Problems the Right Way*. Addison–Wesley, 2001.
32. Jeffrey Voas and Gary McGraw. *Software Fault Injection: Inoculating Programs Against Errors*. John Wiley & Sons, 1997.
33. David Wagner, Jeffrey S. Foster, Eric A. Brewer, and Alexander Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In *Proceedings of Network and Distributed System Security Symposium*, pages 3–17, Catamaran Resort Hotel, San Diego, California, February 2000.
34. David A. Wheeler. Flawfinder. Web page `http://www.dwheeler.com/flawfinder/`, May 2001.
35. David A. Wheeler. Secure programming for Linux and Unix HOWTO v2.89. `http://www.dwheeler.com/secure-programs/`, October 2001.
36. John Wilander. Security intrusions and intrusion prevention. Master's thesis, Linkopings universitet, `http://www.ida.liu.se/~johwi`, April 2002.

A Testbed for Buffer Overflow and Format String Vulnerabilities

In this appendix we have included the 44 function calls used to compare publicly available tools for static intrusion prevention. To shorten it down we have only included the interesting parts. The full code can be downloaded from our homepage `http://www.ida.liu.se/~johwi`.

```
#define BUFSIZE 9
static char static_global_buffer = 'A';
static char global_buffer[BUFSIZE];

/***** Buffer Overflow Vulnerabilities *****/

pointer = gets(buffer); /* Unsafe */

scanf("%8s", buffer_safe);  /* Safe */
scanf("%s", buffer_unsafe); /* Unsafe */

fscanf(fopen(file_name, "w"), "%8s", buffer_safe);  /* Safe */
fscanf(fopen(file_name, "w"), "%s", buffer_unsafe); /* Unsafe */

sscanf(input_string, "%8s", buffer_safe);  /* Safe */
sscanf(input_string, "%s", buffer_unsafe); /* Unsafe */

if(choice==0) vscanf("%8s", arglist); /* Safe */
else vscanf("%s", arglist);           /* Unsafe */

if(choice==0) vsscanf(input_string, "%8s", arglist); /* Safe */
else vsscanf(input_string, "%s", arglist);           /* Unsafe */

if(choice==0)
  vfscanf(fopen(file_name, "w"), "%8s", arglist); /* Safe */
else
  vfscanf(fopen(file_name, "w"), "%s", arglist);  /* Unsafe */

sprintf(buffer_safe, "%8s", input_string);  /* Safe */
sprintf(buffer_unsafe, "%s", input_string); /* Unsafe */

if(strlen(input_string)<BUFSIZE)
  strcat(buffer_safe, input_string);   /* Safe */
strcat(buffer_unsafe, input_string);   /* Unsafe */

if(strlen(input_string)<BUFSIZE)
  strcpy(buffer_safe, input_string);   /* Safe */
strcpy(buffer_unsafe, input_string);   /* Unsafe */

cuserid(buffer_unsafe); /* Unsafe */

if(choice==0) vsprintf (buffer_safe, "%8s", arglist); /* Safe */
```

```
else vsprintf (buffer_unsafe, "%s", arglist);         /* Unsafe */

res = streadd(buffer_safe, "a", "");            /* Safe */
res = streadd(buffer_unsafe, input_string, ""); /* Unsafe */

res = strecpy(buffer_safe, "a", "");            /* Safe */
res = strecpy(buffer_unsafe, input_string, ""); /* Unsafe */

res = strtrns("a", "a", "A", buffer_safe);            /* Safe */
res = strtrns(input_string, "a", "A", buffer_unsafe); /* Unsafe */

/***** Format String Vulnerabilities *****/

printf(&static_global_buffer); /* Safe */
printf(global_buffer);         /* Unsafe */

fprintf(stdout, &static_global_buffer); /* Safe */
fprintf(stdout, global_buffer);         /* Unsafe */

char local_buffer[BUFSIZE];
/* Safe */
sprintf(local_buffer, &static_global_buffer, input_string);
/* Unsafe */
sprintf(local_buffer, global_buffer, input_string);

char local_buffer[BUFSIZE];
/* Safe */
snprintf(local_buffer, BUFSIZE, &static_global_buffer, input_string);
/* Unsafe */
snprintf(local_buffer, BUFSIZE, global_buffer, input_string);

if(choice==0) vprintf(&static_global_buffer, arglist); /* Safe */
else vprintf(global_buffer, arglist);                  /* Unsafe */

if(choice==0) /* Safe */
  vfprintf(stdout, &static_global_buffer, arglist);
else /* Unsafe */
  vfprintf(stdout, global_buffer, arglist);

char local_buffer[BUFSIZE];
if(choice==0) /* Safe */
  vsprintf(local_buffer, &static_global_buffer, arglist);
else /* Unsafe */
  vsprintf(local_buffer, global_buffer, arglist);

char local_buffer[BUFSIZE];
if(choice==0) /* Safe */
  vsnprintf(local_buffer, BUFSIZE, &static_global_buffer, arglist);
else /* Unsafe */
  vsnprintf(local_buffer, BUFSIZE, global_buffer, arglist);
```

Encryption Cycles and Two Views of Cryptography

Peeter Laud*

Tartu University, Liivi 2, Tartu, Estonia
Cybernetica AS, Tartu Lab, Lai 6, Tartu, Estonia
peeter@cyber.ee

Abstract. The work by Abadi and Rogaway has started the process of bringing together the two approaches —formal and computational— to cryptography. Their work has also shown, that it is impossible to completely unify these two approaches in their typical forms — there are some principal differences in their security definitions. The difference is in the security of *encryption cycles*. An encryption cycle is a sequence of keys, where each key is encrypted under the next one, and the last key is encrypted under the first one. In formal treatment, they are considered to be secure, but in computational treatment, insecure. In this paper we make the encryption cycles insecure in the formal model (the *Dolev-Yao model*) as well, by slightly strengthening the attacker. For the modified formal model and the classical computational model, the unifying results by Abadi and Rogaway hold unconditionally.

1 Introduction

This work makes a step towards bridging the gap between formal and computational treatments of cryptography in cryptographic protocols. A cryptographic protocol is a sequence of messages between the entities participating in the protocol, together with rules that specify how these messages have to be constructed and which identities they must satisfy. The protocols have been designed with the aim to achieve their security objectives even when there is an adversary present in the system. The adversary can read, intercept and analyse the messages sent by the parties of the protocol, and it can also construct and send new messages.

Two different models for describing, how the adversary can analyse and construct messages, have evolved over the years. In one of them, called the *formal model*, the messages are considered to be formal expressions from a term algebra, the cryptographic (and other) operations are modeled as constructors of terms, and the adversaries abilities are defined by the structure of messages that are known to it. Some of the most prominent examples of this treatment of cryptographic protocols are [9, 8, 1]. In the other model, called the *computational model*

* Supported by Estonian Science Foundation grant #5279

[22, 10, 5], the messages are considered to be bit-strings, the cryptographic operations are modeled as [probabilistic] functions over the set of bit-strings, and the adversary can be any efficient algorithm.

Of these two models, the computational one is arguably more realistic, as messages are encoded as bit-strings in real life. Also, in reality the adversary can be any efficient algorithm. On the other hand, the formal model is easier to reason about, because of its simple algebraic structure. Until very recently, these two models have evolved mostly separately. Only some years ago the first results relating these two models have started to appear. Abadi and Rogaway [3] have shown that if two formal expressions (i.e. messages in the formal model) "look the same" for an adversary, then the [distributions of] bit-strings corresponding to these two expressions also look the same for the adversaries in the computational model. Abadi and Jürjens [2] and Laud [12, 13] have generalised their results for richer formal languages (instead of the values of formal expressions they are handling outputs of programs).

There is an interesting case that the results of Abadi and Rogaway [3], as well as Abadi and Jürjens [2] and Laud [12] (but not [13]) do not cover. Namely, these same-looking formal expressions may not contain *encryption cycles*. The simplest encryption cycle is $\{K\}_K$ — a key whose encryption under itself is made available to the adversary. In the formal model, an adversary that knows just $\{K\}_K$ is unable to obtain K. In the computational model, the encryption primitive is traditionally required to be secure against attacks, whose setup implies that the adversary cannot obtain $\{K\}_K$. One cannot therefore say that if an adversary in the computational model has obtained the encryption of a key under itself, then it cannot find the key itself. Encryption cycles may also be longer, for example, $(\{K_1\}_{K_2}, \{K_2\}_{K_1})$ is an encryption cycle of length 2. Again, they are secure in the formal model but possibly insecure in the computational model.

There are basically two possible approaches to overcome this discrepancy — something has to change in either the formal or the computational model. Changing the computational model involves giving stronger definitions for security of the encryption primitive and constructing primitives satisfying this security definition. This has been done by Black et al. [7]. However, their construction satisfies their security definition only in the *random oracle model* [6]; the existence of random oracles is another security assumption. Changing the formal model involves strengthening the adversary, such that it is able to obtain K from $\{K\}_K$ (and also from longer encryption cycles involving K). Strengthening the adversary is the topic of this paper.

Considering, what is known about the encryption cycles, they should be avoided if possible. The definition of the strengthened adversary, given in this paper, provides a tool for avoiding them.

In this paper we define the strengthened attacker for the formal model. We show that if two formal expressions look the same to this attacker, then the distributions of bit-strings corresponding to these two expressions look the same for the adversaries in the computational model, no matter whether these expressions contain encryption cycles or not. We also show, that if two formal expressions

do not contain encryption cycles, then they look the same to the strengthened attacker, if and only if the look the same to the normal attacker.

This paper has the following structure. In Sec. 2 we give a short overview of the work done so far for bringing these two treatments of cryptography together. Sec. 3 introduces the formal model — in Sec. 3.1 we define the language of formal expressions and some syntactic notions, in Sec. 3.2 we recall, how powerful the attacker in its classical shape is, in Sec. 3.3 we describe, how the abilities of the attacker define an equivalence relation over formal expressions, in Sec. 3.4 we introduce the modified attacker. Sec. 4 recalls the computational model, it gives the definition of security of encryption systems in Sec. 4.1 and the translation from formal expressions to (families of probability distributions over) bitstrings in Sec. 4.2. In Sec. 5 we state and prove our main justification for modifying the attacker in the way that we did it in Sec. 3.4 (formal equivalence implies indistinguishability of interpretations). In Sec. 6 we show that for expressions without encryption cycles, the new attacker is no stronger than the classical one. Finally, Sec. 7 concludes.

2 Related Work

We have already mentioned [3, 2, 12]. These works can be considered similar in the sense, that they all attempt to show that some equivalence over a certain class of formally defined objects translates to the indistinguishablity of the computational interpretations of these objects.

A separate approach is that of Pfitzmann et al. [18–20] and Backes [4]. They have devised a framework to faithfully abstract the cryptographic primitives, such that the proofs about protocols using these abstractions would also hold if the abstractions are replaced with actual primitives. This framework is more suitable for abstracting integrity properties than confidentiality properties (which include the indistinguishability of interpretations of expressions). Their abstractions are generally more complex and elaborate than formal expressions and process algebras using them.

Recently, Guttman et al. [11] have related two treatments of cryptography for a certain class of authentication protocols. One of the treatments is the formal treatment, where the messages are formal expressions, the other is based on bitstrings, but in our opinion it should not be called "computational". Rather, it should be called "statistical" — the security definitions are not based on computational complexity, but on statistical closeness. The relationship of the treatments is the usual one — the correctness of the protocol in the formal model implies its correctness in the statistical model.

Mitchell et al. [14, 15, 17] have not actually related the two aspects of cryptography, but they have devised a computational semantics for π-calculus, which is a process algebra that is often used for the formal treatment of cryptographic protocols. They have given correctness proofs for some cryptographic protocols, based on their semantics and corresponding (computational) security definitions.

The proofs are computational in kind, though. There is no translation of proofs from formal to computational world.

3 Formal Model

In the formal model, messages are considered to be formal expressions — elements of a (free) term algebra. Their construction reflects, how they have been put together from simpler messages. The set **Exp** of all formal expressions is defined inductively.

3.1 Formal Expressions

Let **Keys** and **Consts** be two sets, we assume that their intersection is empty. The elements of the set **Keys** represent keys in the formal expressions. The elements of **Consts** represent constants, for example booleans or integers. This representation is really only informal, formally the elements of **Keys** and **Consts** have no further structure.

Definition 1. *The set* **Exp** *of* formal expressions *is the smallest set containing the following elements:*

- *If* $K \in$ **Keys**, *then* $K \in$ **Exp**. *If* $C \in$ **Consts**, *then* $C \in$ **Exp**.
- *If* $E_1, E_2 \in$ **Exp**, *then the term* (E_1, E_2) *is also an element of* **Exp**.
- *If* $E \in$ **Exp** *and* $K \in$ **Keys**, *then the term* $\{E\}_K$ *is also an element of* **Exp**.

The set **Exp** is the "smallest useful" set for describing the exchanged messages in cryptographic protocols. Of course, only protocols that use symmetric encryption as their only cryptographic primitive, can be described. The set **Exp** can be easily extended to include more cryptographic primitives. However, they would only complicate our arguments.

Note that although the elements of **Keys** are intended to model cryptographic keys, they are formally just objects without any further structure. Similarly, the term $\{E\}_K$ is meant to model the encryption of the message modelled by E by the key modelled by K. Formally, however, it has no further structure than being a term, consisting of a binary constructor (denoted $\{\cdot\}_{(\cdot)}$) and two immediate subterms.

As the next step we are going to define, when a formal expression has encryption cycles. This property is defined by the structure of the formal expression. We start with some definitions of simple properties of the structure of formal expressions.

Definition 2. *Let* $E, E' \in$ **Exp**. *Expression* E' *is a* subexpression *of* E *(denoted* $E' \sqsubseteq E$*), if one of the following holds:*

- $E' = E$;
- $E = (E_1, E_2)$ *and either* $E' \sqsubseteq E_1$ *or* $E' \sqsubseteq E_2$;
- $E = \{E''\}_K$ *and* $E' \sqsubseteq E''$.

We see that K is generally not a subexpression of $\{E\}_K$. We define a second notion, specifically for keys, that also includes the keys that are used for encryption.

Definition 3. *Let* $K \in$ **Keys** *and* $E \in$ **Exp**. *We say that* K occurs *in* E, *if one of the following holds:*

- $K = E$;
- $E = (E_1, E_2)$ *and either* K *occurs in* E_1 *or* K *occurs in* E_2;
- $E = \{E'\}_{K'}$ *and either* $K = K'$ *or* K *occurs in* E'.

Let $Keys(E) = \{K \in \mathbf{Keys} : K \text{ occurs in } E\}$.

Each formal expression E defines a binary relation "encrypts" on the set of keys occuring in E. Namely, let K, K' be two keys occuring in E, we say that K *encrypts* K', if there exists an expression $E' \in$ **Exp**, such that $K' \sqsubseteq E'$ and $\{E'\}_K \sqsubseteq E$. For example, K_1 encrypts K_3 in the expression $\{\{(K_3, K_4)\}_{K_2}\}_{K_1}$. We say that E *has encryption cycles*, if the relation "encrypts" is cyclic.

3.2 The Attacker (Classical)

As next we are going to describe the equivalence relation over formal expressions given by Abadi and Rogaway [3]. We already mentioned that two formal expressions are equivalent, if they "look the same" to the attacker. "Looking the same" is given by defining, how each expression "looks". This in turn is given by defining, which subexpressions the attacker can see in an expression.

An *entailment relation* $\vdash \subseteq \mathbf{Exp} \times \mathbf{Exp}$ is given. Intuitively, $E \vdash E'$ means that the attacker can compute the expression E' from the expression E. It is the least relation with the following properties:

- $E \vdash C$ for all $C \in$ **Consts**;
- $E \vdash E$;
- if $E \vdash (E_1, E_2)$, then $E \vdash E_1$ and $E \vdash E_2$;
- if $E \vdash \{E'\}_K$ and $E \vdash K$, then $E \vdash E'$.

This definition models, what an attacker can obtain from an expression E without any prior knowledge of other keys or expressions. This model of the attacker is ubiquituous in the formal treatment of cryptography.

Example 1. Consider the term $(\{\{K_1\}_{K_2}\}_{K_3}, K_3)$. The attacker can obtain from this term the key K_3, as well as the subterm $\{\{K_1\}_{K_2}\}_{K_3}$. By using the key on that subterm, the attacker can also obtain $\{K_1\}_{K_2}$. But the attacker cannot obtain the keys K_2 or K_1.

3.3 Equivalence Relation

The next step is to define, how a formal expression "looks" to the attacker. The "look" of the expression E is an element of the set of *patterns* (or extended expressions) **Pat**. It is the smallest set that contains the following elements:

– If $K \in \mathbf{Keys}$, then $K \in \mathbf{Pat}$. If $C \in \mathbf{Consts}$, then $C \in \mathbf{Pat}$.
– If $P_1, P_2 \in \mathbf{Pat}$, then the term (P_1, P_2) is also an element of **Pat**.
– If $P \in \mathbf{Pat}$ and $K \in \mathbf{Keys}$, then the term $\{P\}_K$ is also an element of **Pat**.
– An object denoted by $\Box$ is an element of **Pat**.

Intuitively, $\Box$ denotes a term $\{E\}_K$, where the attacker does not know the key K and therefore cannot see E. An element $P \in \mathbf{Pat}$ denotes some expression that may have ciphertexts that the attacker cannot decrypt as subexpressions.

If the attacker knows the keys in the set $\mathbf{K} \subseteq \mathbf{Keys}$, then the formal expression $E \in \mathbf{Exp}$ "looks" to it like $p(E, \mathbf{K}) \in \mathbf{Pat}$, which is defined inductively over the structure of E as follows:

$$\begin{aligned} p(K, \mathbf{K}) &:= K \qquad (\text{where } K \in \mathbf{Keys}) \\ p(C, \mathbf{K}) &:= C \qquad (\text{where } C \in \mathbf{Consts}) \\ p((E_1, E_2), \mathbf{K}) &:= (p(E_1, \mathbf{K}), p(E_2, \mathbf{K})) \\ p(\{E\}_K, \mathbf{K}) &:= \begin{cases} \{p(E, \mathbf{K})\}_K, & \text{if } K \in \mathbf{K} \\ \Box, & \text{if } K \notin \mathbf{K} \end{cases} . \end{aligned}$$

This assumes that the attacker does not obtain any further keys from E. The most natural value for $\mathbf{K}$ is the set of all keys that can be obtained from E. We therefore define

$$\mathit{pattern}(E) := p(E, \{K \in \mathbf{Keys} \,:\, E \vdash K\}) \ .$$

The pattern $\mathit{pattern}(E)$ defines, how an expression E "looks" to the attacker. For example, $\mathit{pattern}((\{\{K_1\}_{K_2}\}_{K_3}, K_3)) = (\{\Box\}_{K_3}, K_3)$.

Finally, Abadi and Rogaway [3] define, that two expressions E_1 and E_2 are *equivalent* (denoted $E_1 \equiv E_2$), if $\mathit{pattern}(E_1)$ and $\mathit{pattern}(E_2)$ are equal, and they are *equivalent up to renaming* (denoted $E_1 \cong E_2$), if there exists a bijection σ over the set **Keys**, such that E_1 is equivalent to $E_2\sigma$, i.e. σ is used to rename all the keys occuring in E_2. The equivalence $\cong$ is more natural — the elements of **Keys** should be considered as bound names, they may be subjected to α-conversion. The theorem relating two aspects of cryptography also holds for $\cong$ — if $E_1 \cong E_2$ and E_1 and E_2 have no encryption cycles, then the computational interpretations of E_1 and E_2 are indistinguishable, as shown in [3].

3.4 The Attacker (Our Definition)

Strengthening the attacker involves changing the entailment relation $\vdash$. After giving a new definition for $\vdash$, the equivalence of formal expressions can be defined exactly as in Sec. 3.3.

In fact, we are not going to give just a single relation $\vdash$, but a family of relations $\vdash_{\mathbf{K}}$, parametrised by $\mathbf{K} \subseteq \mathbf{Keys}$. Intuitively, $E \vdash_{\mathbf{K}} E'$ means that the attacker can compute E' from E, if it has access to oracles that decrypt with K, where $K \in \mathbf{K}$. The relations $\vdash_{\mathbf{K}}$ are defined as follows:

(i). $E \vdash_{\emptyset} C$ for all $C \in \mathbf{Consts}$;
(ii). $E \vdash_{\emptyset} E$;
(iii). if $E \vdash_{\mathbf{K}} E'$ and $\mathbf{K} \subseteq \mathbf{K}'$, then $E \vdash_{\mathbf{K}'} E'$;
(iv). if $E \vdash_{\mathbf{K}} (E_1, E_2)$, then $E \vdash_{\mathbf{K}} E_1$ and $E \vdash_{\mathbf{K}} E_2$;
(v). if $E \vdash_{\mathbf{K}} \{E'\}_K$, then $E \vdash_{\mathbf{K} \cup \{K\}} E'$;
(vi). if $E \vdash_{\mathbf{K} \cup \{K\}} E'$ and $E \vdash_{\mathbf{K}} K$, then $E \vdash_{\mathbf{K}} E'$;
(vii). if $E \vdash_{\mathbf{K} \cup \{K\}} K$, then $E \vdash_{\mathbf{K}} K$.

In this definition, the items (i), (ii) and (iv) are the same as in the definition of $\vdash$. The item (iii) is an obvious monotonicity property. The item (v) is a bit more optimistic (from the attacker's point of view) than the corresponding item in the definition of $\vdash$ — one does not need the key K to decrypt a ciphertext $\{E'\}_K$, it is enough to have access to an oracle that decrypts with K. The item (vi) says that one way to obtain that access is to have K itself. The item (vii) breaks encryption cycles. It says that if a key K is obtainable by using an oracle that decrypts with K, then this key is (or may be) also obtainable without that oracle. Let us see some examples (with encryption cycles in them).

Example 2. The expression E is $\{K\}_K$.

(1). $E \vdash_{\{K\}} K$ is derived by items (ii) and (v);
(2). $E \vdash_{\emptyset} K$ is derived from (1) by item (vii).

Example 3. The expression E is $(\{K_1\}_{K_2}, \{K_2\}_{K_1})$.

(1). $E \vdash_{\{K_2\}} K_1$ is derived by items (ii), (iv) and (v);
(2). $E \vdash_{\{K_1\}} K_2$ is derived similarly;
(3). $E \vdash_{\{K_1, K_2\}} K_1$ is derived from (1) by item (iii);
(4). $E \vdash_{\{K_1\}} K_1$ is derived from (3) and (2) by item (vi);
(5). $E \vdash_{\emptyset} K_1$ is derived from (4) by item (vii).

Example 4. The expression E is $\{\{(K_1, K_2)\}_{K_1}\}_{K_2}$.

(1). $E \vdash_{\{K_1, K_2\}} (K_1, K_2)$ is derived by items (ii) and (v);
(2). $E \vdash_{\{K_1, K_2\}} K_1$ is derived from (1) by item (iv);
(3). $E \vdash_{\{K_2\}} K_1$ is derived from (2) by item (vii);
(4). $E \vdash_{\{K_2\}} (K_1, K_2)$ is derived from (1) and (3) by item (vi);
(5). $E \vdash_{\{K_2\}} K_2$ is derived from (4) by item (iv);
(6). $E \vdash_{\emptyset} K_2$ is derived from (5) by item (vii);
(7). $E \vdash_{\emptyset} K_1$ is derived from (3) and (6) by item (vi).

We again define, how a formal expression "looks" to an attacker. Now we use the relation $\vdash_{\emptyset}$ instead of the relation $\vdash$. Let

$$pattern^{\mathrm{EC}}(E) := p(E, \{K \in \mathbf{Keys} \,:\, E \vdash_{\emptyset} K\}) \ .$$

Similarly, we define

$$\begin{aligned} E_1 &\equiv^{\mathrm{EC}} E_2 && \text{if } pattern^{\mathrm{EC}}(E_1) = pattern^{\mathrm{EC}}(E_2) \\ E_1 &\cong^{\mathrm{EC}} E_2 && \text{if } \exists \text{ bij. } \sigma \text{ over } \mathbf{Keys} \,:\, E_1 \equiv^{\mathrm{EC}} E_2\sigma \ . \end{aligned}$$

In Sec. 5 we will show that if $E_1 \cong^{\mathrm{EC}} E_2$, then the computational interpretations of E_1 and E_2 are indistinguishable.

4 Computational Model

In this section we give an overview, what constitutes a symmetric encryption system and what does its security mean. We also relate two models, by associating a family of distributions over bit-strings to each formal expression. The content of this section is not original, it repeats [3, Sec. 4 and Sec. 5.1].

Some general concepts have to be introduced first. Let $\mathsf{String} = \{0,1\}^*$ denote the set of all bit-strings. A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is *negligible*, if for all polynomials p over $\mathbb{R}$, $|\epsilon(n)|$ is asymptotically smaller that $|1/p(n)|$. Let D_n be a probability distribution over String. We write $x \leftarrow D_n$ to denote that the variable x is sampled according to D_n. Writing $x, y \leftarrow D_n$ means that the variables x and y are sampled according to D_n *independently of each other*. Let $D = \{D_n\}_{n\in\mathbb{N}}$ and $D' = \{D'_n\}_{n\in\mathbb{N}}$ be two families of probability distributions over String. We say that D and D' are *indistinguishable* (denoted $D \approx D'$), if for every probabilistic algorithm $\mathcal{A}$ that works in polynomial time in its first argument, the difference of probabilities

$$\Pr[\mathcal{A}(n,x) = 1 \,:\, x \leftarrow D_n] - \Pr[\mathcal{A}(n,x) = 1 \,:\, x \leftarrow D'_n] \tag{1}$$

is negligible in n. Intuitively, indistinguishability means "being/looking the same" in the computational model.

A simple property of indistinguishability is the transitivity — if $D \approx D'$ and $D' \approx D''$, then $D \approx D''$. For the proof of this statement, see, for example [13, Sec. 2.2.2].

4.1 Encryption Systems

A symmetric encryption system is a triple of (probabilistic) algorithms $(\mathcal{G}, \mathcal{E}, \mathcal{D})$, whose inputs and outputs are bit-strings. The *key generation algorithm* $\mathcal{G}$ takes as its input a *security parameter* $n \in \mathbb{N}$ (somehow encoded as a bit-string) and returns a key k from a suitable domain $\mathsf{Key} \subseteq \mathsf{String}$. The *encryption algorithm* $\mathcal{E}$ takes as its input a key k and a *plaintext* m and produces a *ciphertext* $\mathcal{E}_k(m)$. The *decryption algorithm* $\mathcal{D}$ takes as its input a key k and a ciphertext c and produces the underlying plaintext, such that $\mathcal{D}_k(\mathcal{E}_k(m)) = m$ for all m and k. The running times of the algorithms $\mathcal{G}$, $\mathcal{E}$ and $\mathcal{D}$ must be polynomial in n.

Let $\mathbf{0}$ be a fixed member of String. The encryption system is *type-0 secure*[1] (see [3] for discussions), if for all algorithms $\mathcal{A}$ running in polynomial time of its argument, the difference of probabilities

$$\Pr[\mathcal{A}^{\mathcal{E}_k(\cdot),\mathcal{E}_{k'}(\cdot)}(n) = 1 \,:\, k, k' \leftarrow \mathcal{G}(n)] - \Pr[\mathcal{A}^{\mathcal{E}_k(\mathbf{0}),\mathcal{E}_k(\mathbf{0})}(n) = 1 \,:\, k \leftarrow \mathcal{G}(n)] \tag{2}$$

is negligible (in n).

In the following we assume that $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ is a fixed type-0 secure encryption system. Abadi and Rogaway [3] have shown, how to construct such systems.

[1] alternative name of this property is: *repetition concealing, which-key concealing and message-length concealing*

algorithm SAMPLE(n, E)
  INITIALISE$(n, Keys(E))$
  return CONVERT(E)

algorithm INITIALISE$(n, \mathbf{K})$
  for all $K \in \mathbf{K}$ **do** $\tau(K) \leftarrow \mathcal{G}(n)$

algorithm CONVERT(E)
  case E **of**
    $K \in$ **Keys**: **return** $\langle \tau(K), \mathsf{key} \rangle$
    $C \in$ **Consts**: **return** $\langle [\![C]\!], \mathsf{const} \rangle$
    (E_1, E_2): **return** $\langle$CONVERT(E_1), CONVERT$(E_2), \mathsf{pair} \rangle$
    $\{E'\}_K$: **return** $\langle \mathcal{E}_{\tau(K)}($CONVERT$(E')), \mathsf{ciphertext} \rangle$

Fig. 1. Algorithm that samples $[\![E]\!]$

4.2 Computational Interpretation of Formal Expressions

Two models of cryptography are related by associating a family of distributions over bit-strings $[\![E]\!]$ to each formal expression E. The formal expression serves here as a "program" that specifies, how the algorithm sampling these distributions works.

Let key, const, pair and ciphertext be four fixed bit-strings. For bit-strings $x_1, \ldots, x_k$ let $\langle x_1, \ldots, x_k \rangle$ be the encoding of the tuple $(x_1, \ldots, x_k)$ as a bit-string. I.e. $\langle x_1, \ldots, x_k \rangle$ is not just the concatenation $x_1 \cdots x_k$, but it contains enough information to recover $x_1, \ldots, x_k$ from it. In other words, $\langle \cdot \rangle : \mathsf{String}^* \to \mathsf{String}$ is an injective function. The n-th component of the family of distributions $[\![E]\!]$ is defined by stating, that it is sampled by the algorithm SAMPLE(n, E) in Fig. 1.

This algorithm starts by picking a value for each key K that occurs in the expression E, where the values for different keys are picked independently of each other. The value that is picked for the key K is denoted by $\tau(K)$. After that, the sampling algorithm proceeds recursively over the expression structure. A key K is mapped to $\tau(K)$. A constant $C \in$ **Consts** is mapped to the representation of C as a bit-string $[\![C]\!]$ (i.e. $[\![\cdot]\!] :$ **Consts** $\to \mathsf{String}$ is a mapping that is fixed beforehand). The mappings of (E_1, E_2) and $\{E\}_K$ are created from the mappings of their components in the most natural way. All interpretations of expressions are tagged by the type of their outermost constructor. This tagging ensures that all bit-strings produced by the sampling algorithm can be uniquely decomposed. Unique decomposition makes the interpretations of formal expressions more similar to those expressions themselves, however, it is not necessary for the proof of the next theorem.

5 Equivalence of Models

The equivalence relations over the formal expressions and their computational interpretations are related by the following theorem:

algorithm SAMPLE(n, P)
  INITIALISE$(n, Keys(P))$
  return CONVERT(P)

algorithm INITIALISE$(n, \mathbf{K})$
  for all $K \in \mathbf{K}$ **do** $\tau(K) \leftarrow \mathcal{G}(n)$
  $\kappa \leftarrow \mathcal{G}(n)$

algorithm CONVERT(P)
  case P **of**
    $K \in \mathbf{Keys}$: **return** $\langle \tau(K), \mathsf{key} \rangle$
    $C \in \mathbf{Consts}$: **return** $\langle [\![C]\!], \mathsf{const} \rangle$
    (P_1, P_2): **return** $\langle$CONVERT(P_1), CONVERT$(P_2), \mathsf{pair} \rangle$
    $\{P'\}_K$: **return** $\langle \mathcal{E}_{\tau(K)}($CONVERT$(P')), \mathsf{ciphertext} \rangle$
    $\Box$: **return** $\langle \mathcal{E}_\kappa(\mathbf{0}), \mathsf{ciphertext} \rangle$

Fig. 2. Algorithm that samples $[\![P]\!]$

Theorem 1. *Let* $E_1, E_2 \in \mathbf{Exp}$*, such that* $E_1 \cong^{\mathrm{EC}} E_2$*. Then* $[\![E_1]\!] \approx [\![E_2]\!]$*.*

Proof. The proof is rather similar to that in [3], only the setup of the hybrid argument requires a more elaborate step for sorting the hidden keys of the expressions. The *recoverable* and *hidden* keys of an expression E are defined by

$$\begin{aligned} \mathit{recoverable}(E) &:= \{K \in \mathbf{Keys} \,:\, E \vdash_\emptyset K\} \\ \mathit{hidden}(E) &:= \mathit{Keys}(E) \backslash \mathit{recoverable}(E) \ . \end{aligned}$$

First we are going to extend the interpretation of expressions given in Fig. 1 to patterns. Then we are going to show $[\![E]\!] \approx [\![\mathit{pattern}^{\mathrm{EC}}(E)]\!]$ for all $E \in \mathbf{Exp}$. This more or less conlcudes the proof, because obviously $[\![E]\!] = [\![E\sigma]\!]$ for all bijections σ over **Keys**. Also, for concluding the proof we need to use the transitivity property of indistinguishability.

The extension of the computational interpretation to the language of patterns is given by redefining the algorithm SAMPLE as in Fig. 2. We see that there are exactly two changes with respect to Fig. 1. First, the algorithm INITIALISE generates an extra key κ. Second, the algorithm CONVERT uses this extra key for interpreting the symbol $\Box$.

The next step in the proof is defining the hybrids for showing that $[\![E]\!]$ and $[\![\mathit{pattern}^{\mathrm{EC}}(E)]\!]$ are indistinguishable. This starts with suitably ordering the keys in $\mathit{hidden}(E)$; the hybrids are defined by successively "opening up" the hidden keys of E. This way, the first hybrid (where all hidden keys are covered) is equal to $\mathit{pattern}^{\mathrm{EC}}(E)$ and the last hybrid (where all hidden keys are visible) is equal to E. In [3], the keys were topologically sorted according to the relation "encrypts"; any order possibly resulting from such sort was suitable. We cannot sort the hidden keys this way, because we allow the relation "encrypts" to have cycles. However, suitable orders still exist.

Let h be the cardinality of $\mathit{hidden}(E)$ and let $\Sigma := \{1, \ldots, h\} \to \mathit{hidden}(E)$. For each $\pi \in \Sigma$ and $i \in \{0, \ldots, h\}$ define

$$\begin{aligned} \mathbf{K}_i^\pi &:= \{\pi(1), \ldots, \pi(i)\} \\ \mathcal{K}_i^\pi &:= \mathbf{K}_{i-1}^\pi \cup \{K \in \mathbf{Keys} \,:\, E \vdash_{\mathbf{K}_i^\pi} K\} \end{aligned} \tag{3}$$

and let $\mathcal{K}^\pi := (\mathcal{K}_1^\pi, \ldots, \mathcal{K}_h^\pi)$. We are looking for a $\pi \in \Sigma$, such that

$$\mathcal{K}_i^\pi = \mathbf{K}_{i-1}^\pi \cup \mathit{recoverable}(E) \tag{4}$$

for each $i \in \{1, \ldots, h\}$. Note that (4) is the minimal possible value of $\mathcal{K}_i^\pi$.

Such π exists. To show it, we need to define two more notions. First, define a partial order on the set $\{\mathcal{K}^\pi \,:\, \pi \in \Sigma\}$. The order is defined componentwise, the components are ordered by set inclusion. Second, for each $\pi \in \Sigma$ and $i \in \{1, \ldots, h\}$ define $\mathsf{fo}_i^\pi \in \{1, \ldots, h\} \cup \{\infty\}$ by

$$\mathsf{fo}_i^\pi = \min\{j \in \{1, \ldots, h\} \,:\, \pi(i) \in \mathcal{K}_j^\pi\} \ .$$

An important property of fo_i^π is $\mathsf{fo}_i^\pi \neq i$. Indeed, suppose that there exist $\pi \in \Sigma$ and $i \in \{1, \ldots, h\}$, such that $\mathsf{fo}_i^\pi = i$. We have $\pi(i) \in \mathcal{K}_i^\pi$. From (3) follows $E \vdash_{\mathbf{K}_i^\pi} \pi(i)$, because $\pi(i) \notin \mathbf{K}_{i-1}^\pi$. From item (vii) of the definition of $\vdash_{\mathbf{K}}$ follows $E \vdash_{\mathbf{K}_{i-1}^\pi} \pi(i)$. Therefore $\pi(i) \in \mathcal{K}_{i-1}^\pi$ (if $i > 1$) or $\pi(i) \in \mathit{recoverable}(E)$. A contradiction with the definition of fo_i^π.

For any $\pi \in \Sigma$, where $\mathcal{K}^\pi$ is minimal, the condition (4) is satisfied. Indeed, suppose that $\mathcal{K}^\pi$ is minimal, but (4) does not hold for some i. Let i be the smallest index, such that (4) does not hold. There exists $j \in \{i+1, \ldots, h\}$, such that $\pi(j) \in \mathcal{K}_i^\pi$; we have $E \vdash_{\mathbf{K}_i^\pi} \pi(j)$ and $\mathsf{fo}_j^\pi = i$. Construct $\pi' \in \Sigma$ as follows:

$$\pi' = \left\{ \begin{array}{ccccccccccc} 1 & & i-1 & i & i+1 & & j & j+1 & & h \\ \downarrow & \cdots & \downarrow & \downarrow & \downarrow & \cdots & \downarrow & \downarrow & \cdots & \downarrow \\ \pi(1) & & \pi(i-1) & \pi(j) & \pi(i) & & \pi(j-1) & \pi(j+1) & & \pi(h) \end{array} \right\} \ .$$

For $l \in \{1, \ldots, i-1, j+1, \ldots, h\}$ we have $\mathcal{K}_l^{\pi'} = \mathcal{K}_l^\pi$. For $l \in \{i+1, \ldots, j\}$ we have

$$\{K \in \mathbf{Keys} \,:\, E \vdash_{\mathbf{K}_l^{\pi'}} K\} \subseteq \{K \in \mathbf{Keys} \,:\, E \vdash_{\mathbf{K}_{l-1}^\pi \cup \pi(j)} K\} = \{K \in \mathbf{Keys} \,:\, E \vdash_{\mathbf{K}_{l-1}^\pi} K\},$$

where the equality holds because of $E \vdash_{\mathbf{K}_{l-1}^\pi} \pi(j)$. Therefore $\mathcal{K}_l^{\pi'} = \mathcal{K}_{l-1}^\pi \subseteq \mathcal{K}_l^\pi$ for all $l \in \{i+1, \ldots, j\}$. Also, we have $\mathcal{K}_i^{\pi'} \subseteq \mathcal{K}_{i+1}^{\pi'} = \mathcal{K}_i^\pi$. We have shown that $\mathcal{K}^{\pi'} \leq \mathcal{K}^\pi$.

We also have $\mathcal{K}_i^{\pi'} \neq \mathcal{K}_i^\pi$. Namely, we have $\pi'(i) \notin \mathcal{K}_l^{\pi'}$ for $l \in \{1, \ldots, i-1\}$. This follows from the construction of π' and from the definition of i as the smallest index for which (4) is not satisfied. We also have $\pi'(i) \notin \mathcal{K}_i^{\pi'}$, because $\mathsf{fo}_i^{\pi'}$ cannot be equal to i. On the other hand, we have $\pi'(i) = \pi(j) \in \mathcal{K}_i^\pi$. Therefore $\mathcal{K}^{\pi'} < \mathcal{K}^\pi$, this contradicts the minimality of $\mathcal{K}^\pi$.

algorithm $\mathcal{A}_0^{f(\cdot),g(\cdot)}(n)$
  INITIALISE$(n, \mathbf{K}_{i-1}^{\pi} \cup \mathit{recoverable}(E))$
  return $\mathcal{A}(n, \text{CONVERT'}(E))$

algorithm INITIALISE$(n, \mathbf{K})$
  for all $K \in \mathbf{K}$ **do** $\tau(K) \leftarrow \mathcal{G}(n)$

algorithm CONVERT'(E)
  case E **of**
    $K \in \mathbf{Keys}$: **return** $\langle \tau(K), \mathsf{key} \rangle$
    $C \in \mathbf{Consts}$: **return** $\langle [\![C]\!], \mathsf{const} \rangle$
    (E_1, E_2): **return** $\langle \text{CONVERT'}(E_1), \text{CONVERT'}(E_2), \mathsf{pair} \rangle$
    $\{E'\}_K$: **if** $K \in \mathbf{K}_{i-1}^{\pi}$ **then**
      return $\langle \mathcal{E}_{\tau(K)}(\text{CONVERT'}(E')), \mathsf{ciphertext} \rangle$
    else if $K = \pi(i)$ **then**
      return $\langle f(\text{CONVERT'}(E')), \mathsf{ciphertext} \rangle$
    else
      return $\langle g(\mathbf{0}), \mathsf{ciphertext} \rangle$

Fig. 3. Algorithm that breaks the type-0 security of the encryption system

Let $\pi \in \Sigma$ be such, that $\mathcal{K}^{\pi}$ is minimal. This fixes the order on the hidden keys; the rest of the proof is more or less the same as in [3]. We define the patterns $P_0, \ldots, P_h \in \mathbf{Pat}$ by $P_i := p(E, \mathbf{K}_i^{\pi})$. Obviously $P_0 = \mathit{pattern}^{\mathrm{EC}}(E)$ and $P_h = E$.

We claim that for each $i \in \{1, \ldots, h\}$, the interpretations $[\![P_{i-1}]\!]$ and $[\![P_i]\!]$ are indistinguishable. Indeed, suppose that $\mathcal{A}$ is an algorithm, such that (1) is nonnegligible for the families of distributions $[\![P_{i-1}]\!]$ and $[\![P_i]\!]$. Consider the algorithm $\mathcal{A}_0^{(\cdot),(\cdot)}$ in Fig. 3. We claim that if its oracle f is $\mathcal{E}_k(\cdot)$ and its oracle g is $\mathcal{E}_{k'}(\cdot)$, then the second argument given to $\mathcal{A}$ is distributed according to $[\![P_i]\!]_n$. If the oracles f and g are both $\mathcal{E}_k(\mathbf{0})$, then the second argument given to $\mathcal{A}$ is distributed according to $[\![P_{i-1}]\!]_n$.

First we argue that the algorithm $\mathcal{A}_0^{(\cdot),(\cdot)}$ never reaches a state where its further action is undefined. This may happen, if the variant $K \in \mathbf{Keys}$ is chosen in the algorithm CONVERT' and $\tau(K)$ is undefined (note that INITIALISE does not necessarily define τ for all keys that occur in E). However, it is easy to see (follows from the definitions of $p(\cdot,\cdot)$ and $\vdash_{\mathbf{K}}$) that the pattern P_i only contains keys in $\mathcal{K}_i^{\pi}$. The property (4) guarantees that we have all these keys available.

When f is $\mathcal{E}_k(\cdot)$ and g is $\mathcal{E}_{k'}(\cdot)$, then the key that f uses corresponds to the key $\tau(\pi(i))$ in Fig. 2, and the key that g uses corresponds to the key κ in Fig. 2. When both f and g are $\mathcal{E}_k(\mathbf{0})$, then the key that they both use corresponds to the key κ in Fig. 2.

By our assumption, the algorithm $\mathcal{A}$ can distinguish $[\![P_{i-1}]\!]$ and $[\![P_i]\!]$, therefore the difference (2) is not negligible for the algorithm $\mathcal{A}_0$. But this contradicts our choice of the encryption system as a type-0 secure system. Hence

$[\![P_{i-1}]\!] \approx [\![P_i]\!]$ and by the transitivity of indistinguishability, $[\![pattern^{\mathrm{EC}}(E)]\!] = [\![P_0]\!] \approx [\![P_h]\!] = [\![E]\!]$. □

This gives the justification for our definition of $\vdash_{\mathbf{K}}$.

6 On the Added Strength of the Attacker

We have defined an equivalence relation $\cong^{\mathrm{EC}}$ over the formal expressions. Let us compare it to the relation $\cong$ defined in [3]. Obviously, it makes sense to only consider expressions that do not contain encryption cycles. The following result holds, showing that the relation $\cong^{\mathrm{EC}}$ is as precise as $\cong$.

Proposition 1. *Let $E_1, E_2 \in \mathbf{Exp}$, such that E_1 and E_2 have no encryption cycles. Then $E_1 \cong E_2$ iff $E_1 \cong^{\mathrm{EC}} E_2$.*

Proof. It is enough to show that if $E \in \mathbf{Exp}$ has no encryption cycles, then for all $E' \in \mathbf{Exp}$, $E \vdash E'$ holds iff $E \vdash_{\emptyset} E'$ holds. Indeed, from this equality of $\vdash$ and $\vdash_{\emptyset}$ follows, that $pattern(E) = pattern^{\mathrm{EC}}(E)$. The direction "if $E \vdash E'$ holds then $E \vdash_{\emptyset} E'$ holds" is trivial. Consider the other direction.

The claim "if $E \vdash_{\emptyset} E'$ holds then $E \vdash E'$ holds" is proved in two steps. The first step is to show that if $E \vdash_{\mathbf{K}} E'$ holds, then it can be derived without using item (vii) in the definition of $\vdash_{\mathbf{K}}$.

Let T be a derivation tree for $E \vdash_{\mathbf{K}} K$, i.e. its root is labeled with $E \vdash_{\mathbf{K}} K$, its vertices are labeled with terms of the form $E \vdash_{\mathbf{K}'} E'$ for some $E' \in \mathbf{Exp}$ and $\mathbf{K}' \subseteq \mathbf{Keys}$, and its edges correspond to the items in the definition of $\vdash_{\mathbf{K}}$. Note that $K \sqsubseteq E'$ holds for all expressions E' occuring in the vertices of T. We show how to construct a derivation tree T' for $E \vdash_{\mathbf{K}\setminus\{K\}} K$, such that for all $E \vdash_{\mathbf{K}'} E'$ that are labels of vertices of T', $K \notin \mathbf{K}'$ holds. In the tree T', item (vii) of the definition of $\vdash_{\mathbf{K}}$ cannot have been used for the key K. By repeatedly using this construction, we can remove all occurrences of item (vii) of the definition of $\vdash_{\mathbf{K}}$ from the derivation trees.

Let $E \vdash_{\mathbf{K}'} E'$ be the label of some vertex v of the tree T and let $T_{\mathbf{K}',E'}$ be the subtree of T that is rooted in this vertex. The tree $T'_{\mathbf{K}',E'}$, whose root is labeled with $E \vdash_{\mathbf{K}'\setminus\{K\}} E'$ can be constructed as follows.

Let the immediate descendants of v be v_i, where $i \in \{1, \ldots, \deg v\}$. Let v_i be labeled by $E \vdash_{\mathbf{K}_i} E_i$. First construct the trees $T'_{\mathbf{K}_i,E_i}$ whose root vertices are labeled by $E \vdash_{\mathbf{K}_i\setminus\{K\}} E_i$. The construction of $T'_{\mathbf{K}',E'}$ depends on the item in the definition of $\vdash_{\mathbf{K}}$ that was used to derive $E \vdash_{\mathbf{K}'} E'$. In most cases we have

$$\Big(\bigwedge_{i=1}^{\deg v} E \vdash_{\mathbf{K}_i} E_i \Rightarrow E \vdash_{\mathbf{K}'} E'\Big) \Rightarrow \Big(\bigwedge_{i=1}^{\deg v} E \vdash_{\mathbf{K}_i\setminus\{K\}} E_i \Rightarrow E \vdash_{\mathbf{K}'\setminus\{K\}} E'\Big) \quad (5)$$

and the tree $T'_{\mathbf{K}',E'}$ can be constructed by taking a new vertex v', labeling it with $E \vdash_{\mathbf{K}'\setminus\{K\}} E'$, and letting its immediate descendants be the root vertices of $T'_{\mathbf{K}_i,E_i}$. The implication (5) does not hold in the following cases:

- $E \vdash_{\mathbf{K}'} E'$ is derived by item (v) in the definition of $\vdash_{\mathbf{K}}$ and $E_1 = \{E'\}_K$. This case is impossible, because $K \sqsubseteq E'$ and therefore we have an encryption cycle in E.
- $E \vdash_{\mathbf{K}'} E'$ is derived by item (vi) in the definition of $\vdash_{\mathbf{K}}$, $E_1 = E'$ and $\mathbf{K}_1 = \mathbf{K}' \cup \{K\}$. In this case we just take $T'_{\mathbf{K}',E'} = T'_{\mathbf{K}_1,E_1}$.
- $E \vdash_{\mathbf{K}'} E'$ is derived by item (vii) in the definition of $\vdash_{\mathbf{K}}$. We again take $T'_{\mathbf{K}',E'} = T'_{\mathbf{K}_1,E_1}$.

The second step for proving the claim "if $E \vdash_{\emptyset} E'$ holds then $E \vdash E'$ holds" is to show, that if $E \vdash_{\mathbf{K}} E'$ holds (we assume that it has been derived without using item (vii) in the definition of $\vdash_{\mathbf{K}}$), then there exists a derivation tree for $E \vdash E'$, whose leaves can be labeled not only with $E \vdash C$ or $E \vdash E$, but also with $E \vdash K$ for $K \in \mathbf{K}$. Consider, by which item has $E \vdash_{\mathbf{K}} E'$ been derived.

(i). The derivation tree of $E \vdash C$ has one vertex, labeled with $E \vdash C$.
(ii). The derivation tree of $E \vdash E$ has one vertex, labeled with $E \vdash E$.
(iii). The derivation tree for $E \vdash E'$ that may have $E \vdash K$, where $K \in \mathbf{K}'$, as additional axioms, has already been constructed.
(iv). The derivation tree for $E \vdash E_i$ with the additional axioms $E \vdash K$, where $K \in \mathbf{K}$, consists of a root vertex labeled with $E \vdash E_i$, whose descendant is the root vertex of the derivation tree for $E \vdash (E_1, E_2)$ with the additional axioms $E \vdash K$, where $K \in \mathbf{K}$.
(v). The derivation tree for $E \vdash E'$ with the additional axioms $E \vdash K'$, where $K' \in \mathbf{K} \cup \{K\}$, consists of a root vertex labeled with $E \vdash E'$ and having two descendants. The first descendant is the root vertex of the derivation tree for $E \vdash \{E'\}_K$ with the additional axioms $E \vdash K'$, where $K' \in \mathbf{K}$. The second descendant is a vertex labeled with $E \vdash K$ having no descendants.
(vi). The derivation tree for $E \vdash E'$ with the additional axioms $E \vdash K'$, where $K' \in \mathbf{K}$, consists of the derivation tree for $E \vdash E'$ with the additional axioms $E \vdash K'$, where $K' \in \mathbf{K} \cup \{K\}$, that is modified by replacing the axioms $E \vdash K$ with the derivation trees for $E \vdash K$ with the additional axioms $E \vdash K'$, where $K' \in \mathbf{K}$.

We have shown that if $E \vdash_{\emptyset} E'$ holds then there exists a derivation tree for $E \vdash E'$. □

7 Conclusions

We have amended the Dolev-Yao model, making it in some sense more realistic. By "more realistic" we mean closer to the computational model of cryptography, which can be considered to be a good model of the real world. Our amendment does not make the model much more complex, it should still be suitable for (mechanical) analysis.

Micciancio and Warinschi [16] have refined the formal equivalence $\cong$, such that the computational interpretations of equivalent formal messages are indistinguishable also if the used encryption system is only *type-1* secure (compared to type-0 security, message length may be revealed). They change the

language of patterns, by replacing the undecryptable ciphertext $\Box$ with a collection $\Box_{\ell_1}, \Box_{\ell_2}, \ldots$, denoting the undecryptable ciphertexts of length ℓ_1, ℓ_2, etc. Obviously, the equivalence $\cong^{\mathrm{EC}}$ can be similarly refined.

References

1. Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1999.
2. Martín Abadi and Jan Jürjens. Formal Eavesdropping and Its Computational Interpretation. In Naoki Kobayashi and Benjamin C. Pierce, editors, *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001*, volume 2215 of *LNCS*, pages 82–94, Sendai, Japan, September 2001. Springer-Verlag.
3. Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *International Conference IFIP TCS 2000*, volume 1872 of *LNCS*, pages 3–22, Sendai, Japan, August 2000. Springer-Verlag.
4. Michael Backes. *Cryptographically Sound Analysis of Security Protocols.* PhD thesis, Universität des Saarlandes, 2002.
5. Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *38th Annual Symposium on Foundations of Computer Science*, pages 394–403, Miami Beach, Florida, October 1997. IEEE Computer Society Press.
6. Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, November 1993. ACM Press.
7. John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption Scheme Security in the Presence of Key-Dependent Messages. In *Proceedings of the Ninth Annual Workshop in Selected Areas in Cryptography*, St John's, Newfoundland, August 2002.
8. Michael Burrows, Martín Abadi, and Roger M. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
9. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983.
10. Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.
11. Joshua D. Guttman, F. Javier Thayer, and Lenore D. Zuck. The faithfulness of abstract protocol analysis: message authentication. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 186–195, Philadelphia, PA, November 2001. ACM Press.
12. Peeter Laud. Semantics and Program Analysis of Computationally Secure Information Flow. In Sands [21], pages 77–91.
13. Peeter Laud. *Computationally Secure Information Flow.* PhD thesis, Universität des Saarlandes, 2002.
14. Patrick Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. A Probabilistic Poly-Time Framework for Protocol Analysis. In *CCS '98, Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 112–121, San Francisco, California, November 1998. ACM Press.

15. Patrick Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. Probabilistic Polynomial-Time Equivalence and Security Analysis. In Jeanette M. Wing, Jim Woodcock, and Jim Davies, editors, *FM'99 - Formal Methods, World Congress on Formal Methods in the Development of Computing Systems*, volume 1708 of *LNCS*, pages 776–793, Toulouse, France, September 1999. Springer-Verlag.
16. Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. In *Workshop on Issues in the Theory of Security - WITS 2002*, Portland, Oregon, January 2002.
17. John C. Mitchell. Probabilistic Polynomial-Time Process Calculus and Security Protocol Analysis. In Sands [21], pages 23–29.
18. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic Security of Reactive Systems. In Steve Schneider and Peter Ryan, editors, *Workshop on Secure Architectures and Information Flow*, volume 32 of *Electronic Notes in Theoretical Computer Science*, Royal Holloway, University of London, 2000. Elsevier Science.
19. Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 245–254, Athens, Greece, November 2000. ACM Press.
20. Birgit Pfitzmann and Michael Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *2001 IEEE Symposium on Security and Privacy*, pages 184–200, Oakland, California, May 2001. IEEE Computer Society Press.
21. David Sands, editor. *Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001*, volume 2028 of *LNCS*, Genova, Italy, April 2001. Springer-Verlag.
22. Andrew C. Yao. Theory and applications of trapdoor functions (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, November 1982. IEEE Computer Society Press.

Developing security protocols in χ-Spaces

Federico Crazzolara and Giuseppe Milicia

BRICS[⋆], University of Aarhus, Denmark
{federico,milicia}@brics.dk

Abstract. It is of paramount importance that a security protocol effectively enforces the desired security requirements. The apparent simplicity of informal protocol descriptions hides the inherent complexity of their interactions which, often, invalidate informal correctness arguments and justify the effort of formal protocol verification. Verification, however, is usually carried out on an abstract model not at all related with a protocol's implementation. Experience shows that security breaches introduced in implementations of successfully verified models are rather common. The χ-Spaces framework is an implementation of SPL (Security Protocol Language), a formal model for studying security protocols. In this paper we discuss the use of χ-Spaces as a tool for developing robust security protocol implementations. To make the case, we take a family of key-translation protocols due to Woo and Lam and show how χ-Spaces aids several steps in the development of a security protocol – protocol executions can be simulated in hostile environments, a security protocol can be implemented, and security properties of implementations can be formally verified.

1 Introduction

Security protocols describe the communications between agents with the aim of exchanging information over an untrusted network in a secure manner. Experience shows that designing and implementing security protocols is a most challenging task. This is the case for a variety of reasons. If we look at the genesis of a security protocol, after determining the security requirements a protocol needs to satisfy, we isolate the following three main steps:

- a design phase in which the protocol is devised and described in a concise but often informal specification language,
- a verification phase in which the security properties of the protocol are studied, and
- an implementation phase.

The first two phases are closely related and may not occur in a clear-cut succession – the results of verification give hints on how to design a better protocol. The third step in the life cycle of a protocol is its actual implementation from its specification. The gap between the design phase and the implementation phase

⋆ Center of the Danish National Research Foundation.

is remarkable. A concise description of a protocol, often given in few lines can yield many lines of code. There is very little, if any, guarantee that the desired security properties hold for the implemented protocol, even if great care has been taken in studying a formal model of the protocol. The SSL protocol is a striking example of a provably correct protocol for which a flawed implementation was given and used in the Netscape browser [1]. We believe that the correctness of a specific protocol implementation matters more than that of an abstract model of the protocol. Unfortunately proving an implementation of a protocol correct is often too difficult and abstraction is needed.

In this paper we argue that the χ-Spaces framework helps developing security protocols. χ-Spaces is substantially a compiler for the SPL language [5, 6], in which protocols can be easily and concisely specified. Therefore a protocol written as a χ-Spaces program usually fits in less than a page. The behaviour of a protocol in a simulation gives hints on what security properties stand or fall for the implemented protocol. χ-Spaces compiles an abstract model of a security protocol into JAVA code and provides all necessary runtime support so that the protocol can be executed in a distributed environment. Security properties of the abstract protocol model transfer to its implementation provided the compiler is correct and provided the underlying cryptography is "practically unbreakable". An advantage of this approach is that the correctness of the implementation of a security protocol is checked once and for all. We plan to formally verify the χ-Spaces compiler in the near future.

2 The χ-Spaces architecture

From an abstract point of view, the architecture of χ-Spaces rests on the general notion of network middleware or tuple space similar to the one introduced in the mid-1980's by the project Linda [8].

This abstract architecture is sketched in Figure 1. Communication between principals occurs via a *space*, a *remotely available* storage area. The approach that χ-Spaces takes is not to rely on a specific implementation of a communication model. Instead, it provides interfaces to communication which can be implemented using a variety of communication frameworks, not necessarily a space. The standard χ-Spaces distribution provides a driver for IBM TSpaces [11], a coordination framework based on the tuple space idea which reflects the abstract architecture of Figure 1.

A χ-Spaces program implementing a security protocol usually makes use of cryptography. Cryptographic primitives are not embedded in the χ-Spaces framework either – any Java Cryptographic Architecture (JCA) [19] compliant provider can be used. End-users don't need to trust any particular JCA implementation and could even implement their own cryptographic primitives.

A χ-Spaces program can either be *interpreted* or *compiled* to JAVA [9] code which can be easily refined (via inheritance) and integrated into existing applications.

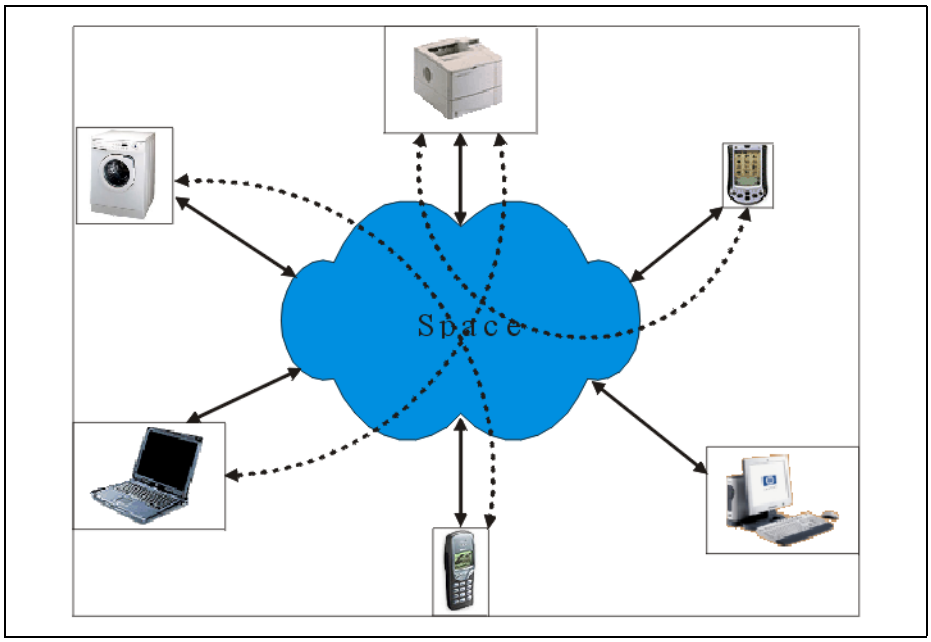

Fig. 1. χ-Spaces Architecture

3 The χ-Spaces syntax

We briefly describe the syntax of χ-Spaces programs – for further details refer to [2]. The χ-Spaces syntax is substantially that of SPL [5, 6]. The main instructions are:

- `new(`V`)` generates a nonce and binds it to the variable V,
- `in` Pat takes a message from the space matching the pattern Pat, and
- `out` M sends the message M to the space.

Instructions separated by dots are intended to be executed one after the other and are the simplest χ-Spaces processes. These simple processes can be composed in parallel to form more complex χ-Spaces programs. As in other languages inspired by the work on process calculi, and in particular by the π-calculus [15], χ-Spaces provides an operator "!" for unbounded replication. The process term `!P` stands for an unbounded number of copies of the process `P`.

A message in an output instruction can be a variable, a constant, a tuple of messages or an encryption. Examples of messages are `5`, `"Eve"`, and `(x,(3,"Bob"))`. An encryption consists of a message and a key, for example `{(1,7)}Pub("Alice")`, which is to be interpreted as the tuple `{(1,7)}` encrypted with the public key bound to the identifier `"Alice"`.

A pattern in an input acts as a binder over its free variables. An input successfully takes a message from the space if it matches against the pattern.

Input actions are blocking – execution can only proceed if the pattern is matched with success. Simple examples of patterns are `5`, `x` and `(x,(x,10))`. Non-linear patterns and nested tuples are permitted; in that case the pattern matching is performed left-to-right in a depth-first fashion. A decryption pattern consists of a variable bound to the cipher-text that is decrypted, a key expression and a pattern. For example `[*C Priv("Bob") > (1,w)]` is a decryption pattern – it first reads a cipher-text from the space and binds it to `*C`, decrypts it with the key `Priv("Bob")` and tries to match the result of the decryption against the pattern `(1,w)`.

To properly deal with keys, χ-Spaces introduces a simple type distinction: key variables (*e.g.* `$k`), basic value variables for constants (nonces or agents names), and general variables (*e.g.* `*C`) for any value, including constants, tuples, keys and cipher-texts. The pattern matching takes into account these types so for example `$k` only matches a key in the space.

A simple example. Security protocols are frequently described as a sequence of messages between distributed principals. A very simple but yet effective example is the *ISO One-Pass Symmetric Key Unilateral Authentication* protocol[4]:

$$A \rightarrow B : \{Na, B\}_{Key(A,B)}.$$

In this protocol the initiator A sends a message to the responder B. The curly brackets indicate encryption – in this example under the symmetric key shared by A and B. The value Na, the nonce, is meant to be fresh and unguessable. Its purpose is to ensure that the request of A is recent.

A possible χ-Spaces implementation of the protocol above is the following:

```
def
  Init(A,B) := {Key(A,B)}
      new(Na) . out {(Na,B)}Key(A,B);

  Resp(A,B) := {Key(A,B)}
      in [*C Key(A,B) > (X,B)];
end

Init{"Alice"}{"Bob"} | Resp{"Alice"}{"Bob"}
```

This implementation illustrates the use of parametric definitions, in this case `Init` and `Resp`. The keys that are used for decryption and encryption are declared in the preamble a definition. In the example only the principals `"Alice"` and `"Bob"` are involved. In order to extend the protocol implementation to allow another initiator, say `"Eve"`, it is enough to replace the last line in the program above with

```
Init{"Alice","Eve"}{"Bob"} | Resp{"Alice","Eve"}{"Bob"}.
```

In χ-Spaces different threads of execution are spawned for every possible combination of the actual arguments. The process `Init`, for example, is launched twice, one time with parameters `"Alice"` and `"Bob"` and the other time with `"Eve"` and `"Bob"`.

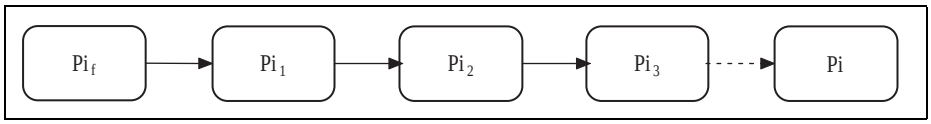

Fig. 2. The Woo-Lam hierarchy

4 The Woo-Lam key-translation protocols

Woo and Lam [22, 21] introduced a family of *key translation* protocols. The protocols they introduced have an initiator role, here A, a responder role, here B, and a trusted authority, here S. The trusted server S shares a key with both the initiator $Key(A, S)$ and the responder $Key(B, S)$. The simplest protocol in the family is named Π and goes as follows:

$$
\begin{array}{l}
(1)\ A \rightarrow B : A \\
(2)\ B \rightarrow A : Nb \\
(3)\ A \rightarrow B : \{Nb\}_{K(A,S)} \\
(4)\ B \rightarrow S : \{A, \{Nb\}_{K(A,S)}\}_{K(B,S)} \\
(5)\ S \rightarrow B : \{Nb\}_{K(B,S)}
\end{array}
$$

The server *translates* A's message $\{Nb\}_{K(A,S)}$ into $\{Nb\}_{K(B,S)}$ which can be understood by the responder B. The Π protocol is derived from a more complex protocol Π_f by repeated simplifications aiming at a more efficient protocol and giving rise to the "hierarchy" in Figure 2. The Π_3 protocol is only a slight variation of the Π protocol and is described as follows:

$$
\begin{array}{l}
(1)\ A \rightarrow B : A \\
(2)\ B \rightarrow A : Nb \\
(3)\ A \rightarrow B : \{Nb\}_{K(A,S)} \\
(4)\ B \rightarrow S : \{A, \{Nb\}_{K(A,S)}\}_{K(B,S)} \\
(5)\ S \rightarrow B : \{A, B, Nb\}_{K(B,S)}
\end{array}
$$

The only difference between Π_3 and Π is in step (5). This simple refinement makes the Π protocol vulnerable to an attack.[1]

5 Protocol simulation

The χ-Spaces framework can be used to test protocols by running local simulations. Different capabilities can be given to a spy process so gain insights in the behaviour of the protocol when executed in a variety of environments.

[1] The original version of Π_3 is prone to a simple replay attack (see [4]). The *χ-Spaces implementation* of Π_3 is immune to this attack if the name of the responder is included in message (5).

```
def

Init(A) := {Key(A,"S")}
   ! out A.
     in Nb.
     out ((A),{Nb}Key(A,"S"));

Resp(B,A,C) := {Key(B,"S")}
   ! in A.
     new(Nb).
     out Nb.
     in *C1.
     out {(A,*C1)}Key(B,"S").
     in ((C),[*C2 Key(B,"S") > Nb]).
     println (B,"took",C,"for",A).
     exit;

Server(A,B) := {Key(A,"S"),Key(B,"S")}
   ! in  [*C1 Key(B,"S") > (A,((X),[*C2 Key(A,"S") > Nb]))].
     out ((X),{Nb}Key(B,"S"));

end

  Init{"Alice","Eve"}
| Resp{"Bob"}{"Alice"}{"Eve"}
| Resp{"Bob"}{"Eve"}{"Alice"}
| Server{"Alice","Bob","Eve"}{"Alice","Bob","Eve"}
```

Fig. 3. The Woo-Lam Π protocol ready for a simulation in χ-Spaces

5.1 Implementing the Π protocol for simulation in χ-Spaces

To perform a simulation of the Woo-Lam Π protocol that is able to detect attacks we tune the χ-Spaces implementation of the protocol so that the simulation stops whenever a successful attack is carried out. The desired security property for this protocol is an authentication property (see [22]):

> "Whenever a responder finishes the execution of a protocol round, the initiator of that round is in fact the principal claimed in step (1) of the protocol."

For the purpose of detecting a violation of this property we annotate some messages with the name of the originator of the message. Figure 3 shows the implemented protocol ready for a simulation. The message that is sent in the last output performed by the initiator is tagged with the initiator's name:

```
out ((A),{Nb}Key(A,"S"))  .
```

The tag of this cipher-text is carried along until it reaches the responder's last input from the server. When the server performs the key translation, it keeps the name of the agent that labels the original encryption. Then, in the code of the responder, upon receiving the translated message from the server one can compare the name appearing in the first input of the responder with the one tagging the received cipher-text. The simulation stops if the two names are different: the agent claimed in the first step of the protocol is not the initiator that should have encrypted the responder's nonce and therefore a successful attack has been detected. In the implementation of the Π protocol of Figure 3 messages do not carry the addresses of the sender and intended receiver. The messages that are sent and received through the space are "anonymous" and therefore agents may pick messages that are not intended for them. In this way the simulation of the protocol is carried out in a harsh environment that allows messages to be redirected to different agents. Often an implementation of a protocol that is intended for *use* rather than simulation includes identifiers of sender and receiver as part of the messages – see for example the implementation of the Π_3 protocol in Section 6. In that case a spy could modify the "address part" of a message so that it is redirected. A spy of this kind can be easily programmed as a χ-Spaces process and executed together with the simulated protocol. One can in fact program a very powerful spy as a χ-Spaces process that makes the simulation environment even more hostile.

5.2 Simulation results

We run a simulation where two agents `"Alice"` and `"Eve"` can initiate the protocol and to which only `"Bob"` can respond following the protocol. Running the simple program described in Figure 3 on an Athlon 1GH processor, with a clean tuple space, an attack was detected in about 30 seconds. The attack is the one pointed out by Abadi (see [22]); a trace of the detected attack is shown in Figure 4. Threads stand for the parts of the protocol that run locally to each principal. Threads display input and output events which correspond to input and output actions of the χ-Spaces processes in Figure 3. The order of occurrence of the events in a thread is from top to bottom. Arrows between threads are among output and input events and tell where a received message comes from. Alice (thread 383) initiates a run of the protocol with Bob as responder (thread 382). Concurrently Eve (thread 384) initiates a run with Bob (thread 377) too. Alice confuses the nonce Bob generated for Eve with the one Bob generated for her – an arrow connects the second event of thread 377, the output of the nonce intended for Eve, with the second event of thread 383, an input event of Alice. Alice continues the protocol interacting with Bob and the server. Bob gets duped in thread 377 where in its last input he accepts a translation of the nonce which was meant for Eve but which got encrypted by Alice. The purpose of thread 374 is that of generating some junk to be received by the input event in thread 377 and which, however, will never be used.

Simulation is done by running a protocol implementation. As a consequence it won't find all attacks the abstract protocol model is prone to but attacks

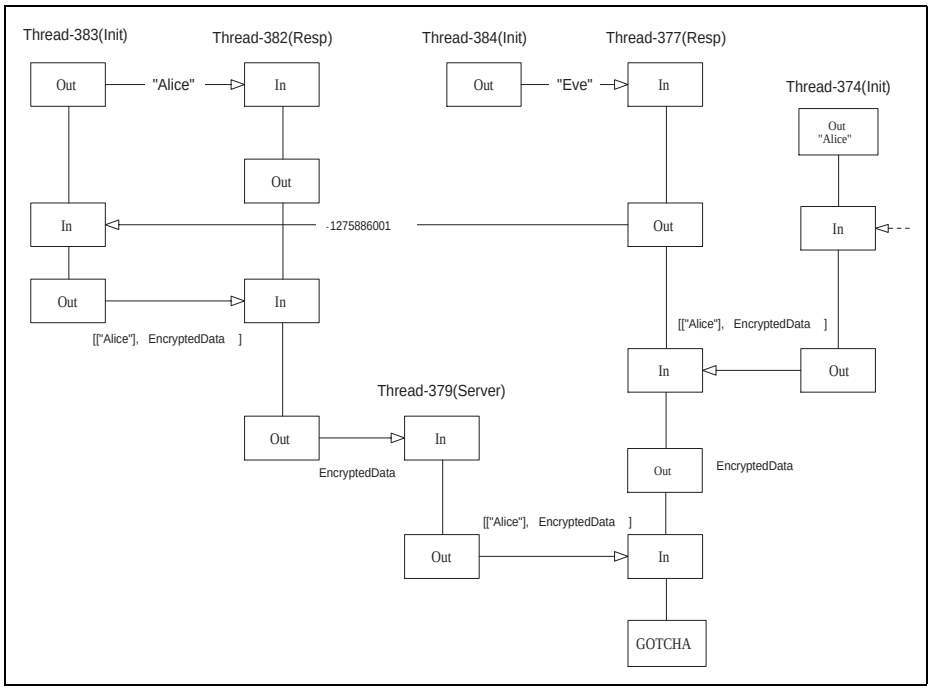

Fig. 4. The simulated attack on Woo-Lam Π

that concern the specific χ-Spaces protocol-implementation instead – our simulation, for example, did not report a possible type-confusion attack on the Π protocol [22] as this attack cannot be carried out against our particular χ-Spaces implementation of Π. The attack succeeds if the initiator of the protocol can be duped into thinking that a structured message is a nonce coming from the responder. The types of χ-Spaces make the implementation of the Woo-Lam Π protocol in Figure 3 immune to attack (for more details see [2]). Clearly, one can give a different χ-Spaces implementation of Π that would permit even that attack.

6 The implementation of a security protocol

The χ-Spaces programming language makes the task of implementing a security protocol fairly easy yielding a concise program. This short program is close to the description of a protocol as a sequence of messages but resolves all issues left open by the informal protocol description – equality tests, for example, are made explicit through the pattern matching mechanism of χ-Spaces and nonce generation is done with the action `new`.

The χ-Spaces code of the Π_3 protocol is shown in Figure 5. The agent names (addresses) are not hardwired into the χ-Spaces communication mechanism. Effi-

```
def

 Init(A,B) := {Key(A,"S")}
    ! out (A,B).
      in (B,A,Nb).
      out (A,B,{Nb}Key(A,"S"));

 Resp(B) := {Key(B,"S")}
    ! in (A,B).
      new(Nb).
      out (B,A,Nb).
      in (A,B,*C1).
      out {(A,*C1)}Key(B,"S").
      in [*C2 Key(B,"S") > (A,B,Nb)];

end

def

 Server(A,B) := {Key(A,"S"), Key(B,"S")}
    ! in [*C1 Key(B,"S") > (A, [*C2 Key(A,"S") > Nb])].
      out {(A,B,Nb)}Key(B,"S");

end
```

Fig. 5. The Woo-Lam Π_3 protocol in χ-Spaces

ciency of protocol communication however, can be sensibly improved if addresses are contained in messages. Then, through a more structured pattern on input actions, agents get only those messages that are directed to them. Therefore in the implementation of Π_3 given in Figure 5 explicit addresses are added to the messages that are sent between initiator and responder – in this case names as addresses are enough. The protocol is meant to work in a distributed environment where each agent executes its own χ-Spaces program. Typically, an ordinary agent `"Alice"` can take both the responder and the initiator role in the protocol. Therefore the program of `"Alice"` consists of a system with two components:

```
Init{"Alice"}{"Bob",...} | Resp{"Alice"} .
```

The server `"S"`, in the case of a symmetric system where each agent can be both initiator and responder, executes:

```
Server{"Alice","Bob",...}{"Alice,"Bob",...} .
```

The server process may be triggered by a request coming from any two "registered" agents.

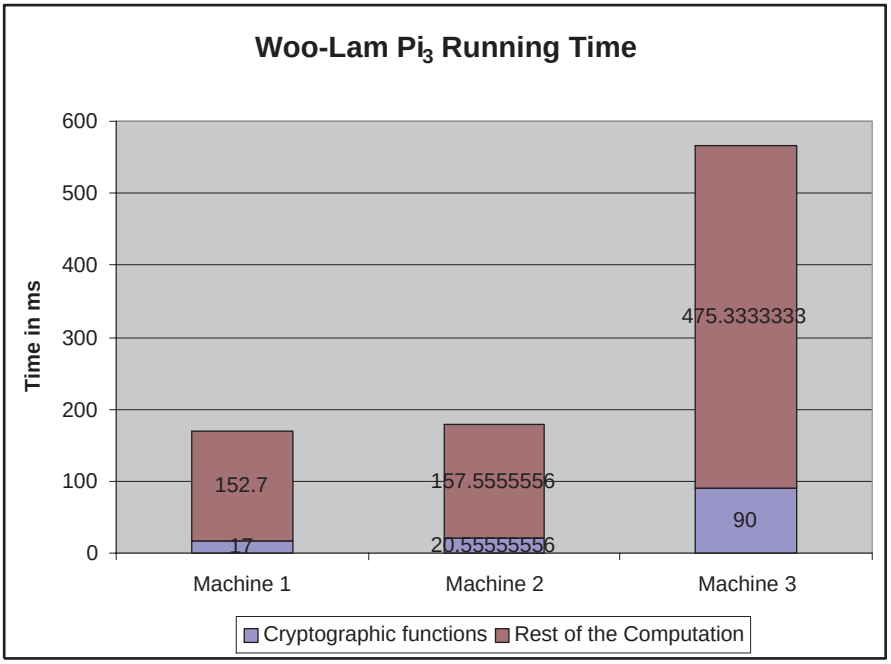

Fig. 6. Performance of the Π_3 implementation

7 Performance evaluation

A natural concern when designing a new programming language is its performances. The current χ-Spaces implementation is to be considered a prototype. In particular we did not attempt to optimise the generated code. However our benchmarks are encouraging. We conducted our tests using the BouncyCastle JCE security provider (freely available from `www.bouncycastle.org`) as cryptographic back-end. Communication to the space went through a local area network (LAN) with a standard 10Mbs transfer rate. Encryption was performed using the DES cipher with 128-bits pre-generated keys. The space implementation of choice was IBM TSpaces [11] version 1.2sp2. The χ-Spaces protocol was compiled to JAVA bytecode using the JDK1.4 compiler. The machines used were the following:

Machine 1 AMD XP 1800+ 512MB Windows 2000 JDK1.4
Machine 2 AMD Athlon 1Ghz 256Mb Windows XP JDK1.3.1
Machine 3 Celeron 300Mhz 256Mb Windows 2000 JDK1.4

As we can see in Figure 6 very little time is spent on the low-level encryption routines. As to be expected the main performance bottleneck is communication over the network. Figure 6 shows that the performances degraded sensibly on

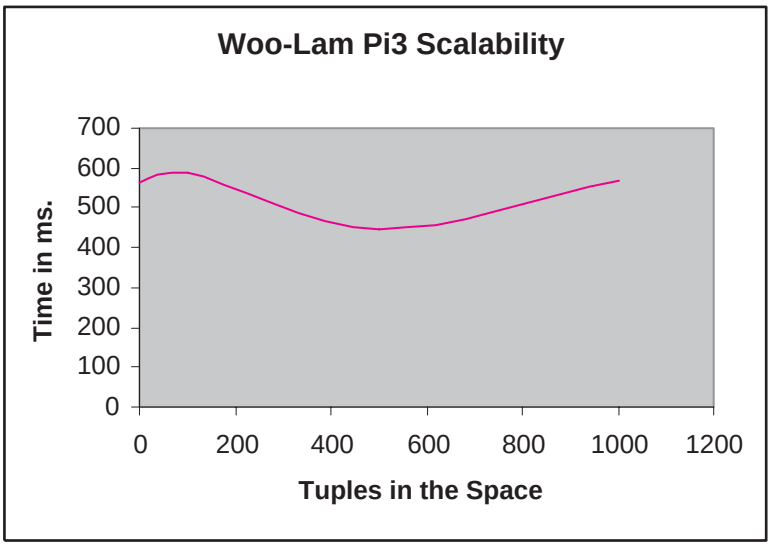

Fig. 7. Scalability Test

a low-end machine (Machine 3). However very little, if any, is gained stepping from the medium-end (Machine 2) to the high-end (Machine 1) machine. This is the case as most of the time was spent doing communication rather than computation. Machine 1, on the other hand, spends a noticeable amount of time in the low-level cryptographic routines. It is conceivable that performances can be improved using a more efficient cryptographic provider.

An interesting question regards the scalability of the system. Depending on load of the space we would expect performances to drop. We run the protocol using a space which was constantly filled. We performed the test with the space running on Machine 1 and the protocol on Machine 3. Our results can be found in Figure 7. As one can see, performance remains on acceptable levels. This test is heavily dependant on the space implementation.

8 Security theorems

The process of developing a security protocol ideally ends with the formal proofs of the security properties that one believes hold for the *protocol implementation*. Simulation gives hints to which security properties stand and which ones fall for a certain protocol – the protocol could be reviewed according to the results of a simulation. Only a formal proof however, can give certainty of correctness and in some cases it can reveal further weaknesses in the protocol.

The Petri-net semantics of SPL (see [6]) transfers directly to χ-Spaces and therefore can be used for reasoning about the correctness of protocols written

in χ-Spaces. The proofs of security properties are carried out exploiting both the dependencies among the events and the shape of events that occur in a run of the protocol. In this short paper we do not include the proof of the security theorem that we state later in this section – for a detailed proof please see [2].

8.1 The Π_3 system

The system that we want to analyse is distributed among the agents that initiate a protocol round or respond to a protocol initiation. In addition there is a server that performs the key translation. We consider the rather general situation where each agent can participate in the protocol in both initiator and responder roles. For simplicity we assume that there is only one server that performs key-translation. In the remaining part of this section, Π_3 is the SPL-process term formed substantially out of the parallel composition of the χ-Spaces programs of the protocol participants (see [2] for details of how χ-Spaces maps to SPL):

$$\begin{array}{ll} P_{Init} & \equiv \|_{A,B \in \mathbf{A}}\ Init(A,B) \\ P_{Resp} & \equiv \|_{A \in \mathbf{A}}\ Resp(A) \\ P_{Server} & \equiv \|_{A,B \in \mathbf{A}}\ Server(A,B) \\ P_{Spy} & \equiv Spy \\ \\ \Pi_3 & \equiv \|_{i \in \{Init,Resp,Server,Spy\}}\ P_i \end{array}$$

Given a set I, the term $||_{i \in I} p_i$ stands for the parallel composition of all processes p_i for $i \in I$. The set $\mathbf{A}$ is a set of agent names.

Security of the system is studied in a malicious environment and therefore a spy is included as a parallel component of Π_3. A powerful spy can be programmed as a χ-Spaces process; it allows to decompose structured messages, compose messages into tuples, encrypt and decrypt with available keys (see [2] for further details about the program of the spy). We assume however that the spy can not break the underlying cryptography that is used in the protocol and that it cannot guess random numbers.

Even if there is always only a finite number of agents interacting using the protocol, we do not restrict ourselves to a fixed finite number of agents when analysing it ($\mathbf{A}$ is an arbitrary set of agent names). Proving security for an arbitrary number of agents implies that our results still hold even when agents join or leave the protocol system. For a similar reason we allow each agent to perform an arbitrary number of rounds of the protocol.

8.2 Elements of Petri-net semantics

A run of the protocol is described by a sequence of configurations (markings) and events of a Petri net:

$$\langle \Pi_3, s_0, t_0 \rangle \xrightarrow{e_1} \ldots \xrightarrow{e_r} \langle p_r, s_r, t_r \rangle \xrightarrow{e_{r+1}} \ldots$$

Configurations $\langle p_i, s_i, t_i \rangle$ are triples consisting of a process term p_i which denotes the point of control reached by the protocol and determines the possible events that can occur from that point, a set s_i containing all the values that appeared so far in the run, and a set t_i of messages describing the contents of the tuple space. The initial configuration starts with the term Π_3 and is proper, in the sense that the set s_0 contains all values that appear in Π_3 and in messages contained in the initial tuple space t_0 – the net semantics keeps track in s_i of the values that are already present and if at step $i+1$ a "new" value needs to be created it chooses one that is not in s_i.

When an event occurs a configuration evolves into another one. Events carry actions such as $act(e) = i : out\, new\, n\ M$, the action of an output event e, where i is an index denoting the parallel component the event belongs to, n is a freshly created value, and M a message sent onto the tuple space. We write $out\ M$ if no new names are created before sending M. The action $act(e) = i : in\ M$, instead, is that of an input event. We write $e \longrightarrow e'$ when the event e precedes the event e' in a run. We also write $M \sqsubseteq t$ when there is a message in the tuple space t containing M as a submessage.

8.3 Formal correctness of Π_3

The rather informal authentication requirement for Π_3:

> "Whenever a responder finishes a round of the protocol, the initiator of that protocol round is in fact the principal claimed in step (1) of the protocol" [22].

can easily be made precise in the Petri-net model. The property should hold for each run of the protocol. We are particularly concerned about those runs in which the responder completes a protocol round.

The informal correctness requirement we recalled above can be seen, following the lead of Lowe (see [13]), as an agreement property: To a completed responder round of B done apparently with A as initiator, should correspond in that protocol run a completed initiator round of A done apparently with B as responder. The messages that are exchanged should agree on their values. Formally, one can prove the following correctness property for the Π_3 protocol:

Theorem 1. *(Authentication of the responder). If a run*

$$\langle \Pi_3, s_0, t_0 \rangle \xrightarrow{e_1} \ldots \xrightarrow{e_r} \langle p_r, s_r, t_r \rangle \xrightarrow{e_{r+1}} \ldots$$

contains the responder event b_5 with action

$$act(b_5) = B : i : in\ \{(A, B, n)\}_{Key(B,S)}$$

for some index i and if $Key(A,S), Key(B,S) \not\sqsubseteq t_0$ then the run contains the initiator event a_3 with action

$$act(a_3) = A, B : j : out\ (B, A, \{n\}_{Key(A,S)})$$

and such that $a_3 \longrightarrow b_5$ □

Even if the statement of the theorem starts from the final event of the responder and asks for the final event of the initiator, it fits the shape of the agreement property that we discussed above. The complete responder and initiator rounds can be reconstructed from those two events:

- Whenever a sequence of configurations and events obtained from the net of Π_3 contains an event corresponding to the last action of the responder one can recognise that the responder indeed completed the round and who was claimed as initiator in the first step of the protocol. Let $A, B \in \mathbf{A}$ be names of agents and S the name of a server in the Π_3 system. From the dependency among responder events in a protocol run (see [6, 2]) it follows that if a run of the protocol contains the responder event b_5 with action

$$act(b_5) = B : i : in\ \{(A, B, n)\}_{Key(B,S)}$$

  it also contains the responder event b_1 with action

$$act(b_1) = B : i : in\ (A, B)$$

  and such that $b_1 \longrightarrow b_5$. Therefore B believes that he responded to a request initiated by A.
- In a similar way as for the responder, whenever a run contains an initiator a_3 with action

$$act(a_3) = A, B : j : out\ (B, A, \{n\}_{Key(A,S)})$$

  one can construct from the event dependencies the complete initiator round obtaining $a_1 \longrightarrow a_2 \longrightarrow a_3$ where the events a_1 and a_2 have actions

$$\begin{aligned} act(a_1) &= A, B : j : out\ (A, B) \\ act(a_2) &= A, B : j : in\ (B, A, n) \quad . \end{aligned}$$

  Therefore in that round A believes that it was engaged in the protocol together with agent B as responder.

9 Related work

Much work has been done in the field of security protocol analysis and tools have been developed for that purpose. Some of the approaches that we find are more related to χ-Spaces are those taken by Casper [12], Athena [18] and CAPSL [7].

Casper translates an abstract protocol specification to a CSP [10] description of the protocol that can then be checked using FRD [16]. Similarly to χ-Spaces, Casper aids the formalisation of informal protocol descriptions in a high level language. In χ-Spaces the simulation and verification are steps in the protocol development process and they target the *implementation* of a protocol, not its abstract specification. This sets apart χ-Spaces from tools like Casper. Athena is based on the strand-space model [20] which is known to be related to the

net-semantics of SPL [6] and therefore to χ-Spaces. Athena has been used to automatically generate protocols that satisfy certain properties. Like done with "χ-Spaces protocols", JAVA code can be generated from "Athena protocols" [17]. CAPSL is a high level language close to the informal notation. Its primary goal is to serve as a common specification language that can be translated to input for different tools for the verification of security protocols. The semantics of CAPSL is given by a translation to an intermediate language based on a multiset rewriting model [3]. Recent work showed how to generate JAVA code from CIL code and so implement protocols [14]. In our opinion, for both Athena and CAPSL it is not clear weather the behaviour of the running code reflects that of the formal model for which security properties of protocols can be verified. The JAVA code for a "CAPSL protocol" allows communication via sockets to an environment (or network). An environment for demonstration that works locally has been programmed. We do not know if at present the system has been tried in a distributed setting.

10 Conclusions

This paper addresses the problem of developing robust and correct security-protocol *implementations*.

The χ-Spaces framework is only a prototype at the moment. Some extensions to the language are needed for being able to develop industrial strength protocols. The extensions that one seems to require substantially consist in allowing recursive definitions and in adding an "if then else" instruction and some other cryptographic primitives. The process of tailoring a protocol for simulation is rather ad-hoc. We plan to study a general methodology and extend χ-Spaces to better support simulation. Our goal is to devise an automatic procedure that given a security property formalised in terms of events of a protocol annotates a χ-Spaces implementation of the protocol with the necessary information to perform a simulation.

References

1. ACROS. Bypassing warnings for invalid SSL certificates in Netscape Navigator. ACROS Security Problem Report, 2000.
2. M. Cáccamo, F. Crazzolara, and G. Milicia. χ-Spaces: From a model to a working language. Technical report, BRICS, 2002. `http://www.chispaces.com`.
3. I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In R. Gorrieri, editor, *Proceedings of the 12th IEEE CSFW*, June 1999.
4. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0. `www-users.cs.york.ac.uk/~jac`, 1997.
5. F. Crazzolara and G. Winskel. Petri nets in Cryptographic protocols. In *Proceedings 6th FMPPTA Workshop*, April 2001.
6. F. Crazzolara and G. Winskel. Event in security protocols. In *Proceedings of the Eight ACM CCS*, November 2001.

7. G. Denker and J. Millen. CAPSL integrated protocol environment. In *DISCEX*, 2000.
8. D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
9. J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification Second Edition*. Sun microsystems, June 2000.
10. C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
11. T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, and P. Bowman. Hitting the distributed computing sweet spot with Tspaces. *Computer Networks*, 35(4):457–472, 2001.
12. G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proceedings of the 10th IEEE CSFW*, 1997.
13. G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE CSFW*, 1997.
14. J. Millen and F. Muller. Cryptographic protocol generation from CAPSL. Technical Report SRI-CSL-01-07, SRI International, December 2001.
15. R. Milner. *Communicating and mobile systems: The π-calculus*. Cambridge University Press, 1999.
16. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *IEEE Symposium on Foundations of Secure Systems*, 1995.
17. D. Song, A. Perrig, and D. Phan. AGVI – Automatic Generation, Verification, and Implementation of Security Protocols. In *Proceedings of the 13th CAV conference*, 2001.
18. D. X. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.
19. Sun microsystems. *JavaTM Cryptography Architecture API Specification and Reference*. `http://java.sun.com`.
20. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? In *1998 IEEE Symposium on Security and Privacy*, 1998.
21. T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, January 1992.
22. T. Y. C. Woo and S. S. Lam. A Lesson on Authentication Protocol Design. *Operating Systems Review*, pages 24–37, 1994.

The Role Compatibility Security Model

Amon Ott

Compuniverse / RSBAC
Email: ao@rsbac.org, WWW: http://www.rsbac.org

Abstract. This paper presents the "Role Compatibility" access control model. It has been specially designed to address recent vulnerabilities in network servers by confining compromised services and protecting the base of the system. Furthermore, while being powerful and flexible when needed, it remains fast and easy to use for simple setups.
The model design goals, its specification and implementation outline are presented, followed by a brief comparison to the RBAC and the DTE model. Finally, a Webserver example shows how the model can be used to protect real server systems.

Keywords: Security Model, Access Control, Internet Server, Linux

1 Introduction

As a response to the increasing rate of server vulnerabilities and attacks against them, network server systems require a conceptional solution for better security. Kernel level access control with a specialized security model provides such a solution.

Since no existing model suited our requirements, the Role Compatibility (RC) model has been designed and implemented in the RSBAC framework since December 1998. It supports both a general protection of the base system and an encapsulation of all network service programs to strictly confine security compromises. The abstraction of a role-based model seemed appropriate for this task.

The RSBAC framework provides a generic infrastructure for security model implementations, including persistent list management. It groups access objects into so-called *target types*, e.g. FILE, DIR or IPC[1]. Network access is controlled through *Network Templates*, which provide persistent default attribute values for dynamic network objects.[RSBAC]

The RC model has been in stable production use since January 2000, and a lot of experience with the RSBAC framework and the RC model has been gained. The latest application benchmarks show an RC model overhead of 1.25% against an empty framework, including the Authentication Enforcement (AUTH) module, which is outside the scope of this paper.

[1] System V Inter Process Communication Object, e.g. Shared Memory.

As an example, the typical configuration as given in section 6 effectively confines the Apache Webserver and thus prevents an infection by recent Linux malware like the OpenSSL Slapper worm family. The RC model has also been used to secure several Linux firewall configurations, which has been a common DTE model application for some time.[2]

2 Design Goals

The RC model design had to meet most access control requirements on modern Linux based server systems. In detail, the following design goals were accomplished:

Role based model: The abstraction of users to roles and objects to types leads to administration on a functional level and avoids the complexity of per-object control.

Single roles: Each process must have only one current role at a time. Each user ID must have only one default role, which can be assigned to the process when the user ID is acquired.

Program roles: Different programs run by the same user must be able to have different roles. Program roles must override user default roles.

Changing of roles: A process must be able to actively change its current role, if allowed by administration.

Single types: Each object must have only one type.

Granularity: Every role and type combination must have an individual set of allowed accesses.

Separation of administration duty: The model must support separation of duty for administration.

Full configurability: No hard-wired settings should be enforced.

Functional default settings: When unconfigured, the model must allow the system to work as expected.

No changes to existing applications: All applications that are not aware of the model must work as expected, unless when they have insufficient privileges.

Adaptive complexity: Model administration should be only as complex as necessary to meet the actual requirements. Using default settings, the model must behave as a simple role model. All special behaviour must be optional.

3 Specification

3.1 Basic Definitions

Within the RC model specification, the active entities (subjects) are processes working on behalf of users and executing one program file with a set of dynamic libraries at a time.

[2] Section 5 contains a comparison with RBAC and DTE models.

Objects are grouped into the RSBAC framework target types, but different groupings of objects would not change the model significantly.

Access rights are the standard framework request types plus some model specific rights. Like modified object groupings, a different set of standard access rights would not affect the model itself.

The following terms will be used:

- owner(p:process):user := owner of process p
- parent(p:process):process := parent process of process p
- program(p:process):file := program file currently executed by process p
- parent(f:filesystem object):filesystem object := parent object of filesystem object f
- $attributename_{tn}$(o:object):valuetype := value of attribute *attributename* of object o at time n

Processes as subjects can perform some model relevant actions:

- changeowner_{tn}(p:process, u:user) := change owner of process p to u at time n
- clone_{tn}(p_1:process, p_2:process) := creation of process p_2 by parent process p_1 at time n
- execute_{tn}(p:process, f:file) := start execution of program file f in process p at time n
- createfs_{tn}(p:process, f:filesystem object) := creation of filesystem object f by process p at time n
- createipc_{tn}(p:process, i:IPC object) := creation of IPC object i by process p at time n

Three types of rules will be specified:

1. Invariants define rules, which must always be met. Here the effective values of inheritable filesystem object attributes are determined.
2. Transitions define the next state of an attribute after a certain action.
3. Constraints define the conditions to be met when an action is performed.

3.2 Roles and Types

Roles Roles and types are identified by a non-negative integer index. Every process has one current role, and every user has one default role.

- currentrole_{tn}(p:process):role := current role of process p at time n
- defrole_{tn}(u:user):role := default role of user u at time n

The first system process 0 gets the default role of user 0.

$$\text{currentrole}_{t0}(0) := \text{defrole}_{t0}(0) \tag{1}$$

In the default setting, i.e., if not specified differently by the *Initial Role* or the *Forced Role* attributes of the program file or the process (see subsection 3.5), the following implicit role transitions are performed:

– All subprocesses inherit the current role of their parent process.

$$\text{clone}_{tn}(\text{p}_1, \text{p}_2) \Rightarrow \ \text{currentrole}_{tn+1}(\text{p}_2) := \text{currentrole}_{tn}(\text{p}_1) \tag{2}$$

– On every change of the process owner, the process current role is set to the new owner's default role.

$$\text{changeowner}_{tn}(\text{p}, \text{u}) \Rightarrow \ \text{currentrole}_{tn+1}(\text{p}) := \text{defrole}_{tn}(\text{u}) \tag{3}$$

– When another program is executed, the current role of the process is kept.

$$\text{execute}_{tn}(\text{p}, \text{f}) \Rightarrow \ \text{currentrole}_{tn+1}(\text{p}) := \text{currentrole}_{tn}(\text{p}) \tag{4}$$

Thus, the default scheme follows the expected behaviour of a role based model.

Types Every object has an RC type. Hierarchically organized objects of RSBAC target types FILE, DIR, FIFO and SYMLINK[3] can have a special type value *inherit parent*, in which case the parent object's type is used. If there is no parent, the default value 0 is applied.

Whenever values may be inherited, the term *effective value* is used to denote the final value. Inheritance greatly reduces the number of attribute values to be stored and follows the usual way of grouping objects in hierarchies.

– type_{tn}(o:object):type := type of object o at time n
– efftype_{tn}(f:filesystem object):type := effective type of filesystem object f at time n, including inheritance

The effective type is derived as follows:

$$\text{efftype}_{tn}(\text{f}) := \begin{cases} \text{if } \text{type}_{tn}(\text{f}) = \text{inherit\_parent} \wedge \exists \text{parent}(\text{f}) : \text{efftype}_{tn}(\text{parent}(\text{f})) \\ \text{if } \text{type}_{tn}(\text{f}) = \text{inherit\_parent} \wedge \nexists \text{parent}(\text{f}) : 0 \\ \text{if } \text{type}_{tn}(\text{f}) \neq \text{inherit\_parent} : \text{type}_{tn}(\text{f}) \end{cases} \tag{5}$$

When a new filesystem object is created, its type is set to the value of the role attribute *Default fd create type* of the current role of the creating process, which can also be the special value *inherit parent* mentioned above.

$$\text{createfs}_{tn}(\text{p}, \text{f}) \Rightarrow \ \text{type}_{tn+1}(\text{f}) := \text{default\_fd\_create\_type}_{tn}(\text{currentrole}_{tn}(\text{p})) \tag{6}$$

When a new process object is created, its type is set depending on the value of the role attribute *Default process create type* of the current role of the creating process. The special and default value *inherit parent* sets the type value to that of the creating process. Be

$$\text{pct}_{tn}(\text{p}_1) := \text{default\_process\_create\_type}_{tn}(\text{currentrole}_{tn}(\text{p}_1)) \tag{7}$$

[3] Objects of these target types are also referenced as *filesystem objects*.

$$\text{clone}_{tn}(\text{p}_1, \text{p}_2) \Rightarrow \ \text{type}_{tn+1}(\text{p}_2) := \begin{cases} \text{if } \text{pct}_{tn}(\text{p}_1) = \text{inherit\_parent} : \text{type}_{tn}(\text{p}_1) \\ \text{if } \text{pct}_{tn}(\text{p}_1) \neq \text{inherit\_parent} : \text{pct}_{tn}(\text{p}_1) \end{cases} \tag{8}$$

On execution of a new program file, the process type is set according to the value of the role attribute *Default process execute type* of the current role of the process. The special and default value *inherit parent* leaves the process type unchanged. Be

$$\text{pet}_{tn}(\text{p}) := \text{default\_process\_execute\_type}_{tn}(\text{currentrole}_{tn}(\text{p})) \tag{9}$$

$$\text{execute}_{tn}(\text{p}, \text{f}) \Rightarrow \ \text{type}_{tn+1}(\text{p}) := \begin{cases} \text{if } \text{pet}_{tn}(\text{p}) = \text{inherit\_parent} : \text{type}_{tn}(\text{p}) \\ \text{if } \text{pet}_{tn}(\text{p}) \neq \text{inherit\_parent} : \text{pet}_{tn}(\text{p}) \end{cases} \tag{10}$$

Changing the owner of a process leads to the process type being set to the value of the role attribute *Default process chown type* of the current role of the process. The special and default value *inherit parent* leaves the process type unchanged. The other valid special value *use new role def create* uses the value of the role attribute *Default process create type* of the new current role of the process (see Roles). Be

$$\text{pot}_{tn}(\text{p}) := \text{default\_process\_chown\_type}_{tn}(\text{currentrole}_{tn}(\text{p})) \tag{11}$$

$$\text{pct}_{tn+1}(\text{p}) := \text{default\_process\_create\_type}_{tn}(\text{currentrole}_{tn+1}(\text{p})) \tag{12}$$

$$\text{changeowner}_{tn}(\text{p}, \text{u}) \Rightarrow$$
$$\text{type}_{tn+1}(\text{p}) := \begin{cases} \text{if } \text{pot}_{tn}(\text{p}) = \text{inherit\_parent} : \text{type}_{tn}(\text{p}) \\ \text{if } \text{pot}_{tn}(\text{p}) = \text{use\_new\_role\_def\_create} : \text{pct}_{tn+1}(\text{p}) \\ \textit{else} : \text{pot}_{tn}(\text{p}) \end{cases} \tag{13}$$

Finally, the types of newly created IPC objects can be influenced by the value of the role attribute *Default ipc create type* of the current role of the process.

$$\text{createipc}_{tn}(\text{p}, \text{i}) \Rightarrow \ \text{type}_{tn+1}(\text{i}) := \text{default\_ipc\_create\_type}_{tn}(\text{currentrole}_{tn}(\text{p})) \tag{14}$$

The types of all newly created network objects are derived from their templates[4] and cannot be preset through role attributes.

Default type values provide a mandatory way to keep new objects suitable for the roles that created them, while completely avoiding discretionary elements for type selection and the necessity of making applications aware of the access control model.

3.3 Role Compatibility

While most changes of the current role of a process are implicit with certain actions, processes can also actively change their current role. This is specially

[4] RSBAC Network Templates, see documentation at [RSBAC].

useful for short term administration tasks and for server programs, whose subprocesses have to act in several roles without changing their user ID.[5] In both cases, the original role should not be regained.

- changerole$_{tn}$(p:process, r:role) := process p actively changes its current role to r at time n

The right to do so is called *Role Compatibility*: A process may change its current role r_1 to role r_2, if role r_2 is in the set of compatible roles of role r_1.

- comproles(r:role):set of roles := set of compatible roles for role t

$$\text{changerole}_{tn}(\text{p}, \text{r}) \Rightarrow \ r \in \text{comproles}_{tn}(\text{currentrole}_{tn}(\text{p})) \tag{15}$$

3.4 Type Compatibility

Accesses by processes performing a current role to objects of certain types are controlled through the *Type Compatibility* settings.

- getaccess$_{tn}$(p:process, o:object, a:access_type) := process p gets access to object o with access type (request type) a at time t

A process with current role r may access objects of type t with accesses of access type (request type) a, if role r is marked as *Type Compatible* with type t for access type a.

- compatible$_{tn}$(r:role, t:type, a:access_type) := role r is marked as compatible with type t for access type (request type) a at time n

$$\text{getaccess}_{tn}(\text{p}, \text{o}, \text{a}) \Rightarrow \ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{efftype}_{tn}(\text{o}), \text{a}) \tag{16}$$

Type compatibility sets are kept separately for the different RSBAC target types.

3.5 Program Based Roles with Initial and Forced Roles

There are two ways to assign roles to programs: initial and forced roles. Both program role settings are kept as file attributes.

Initial Roles If an initial role has been assigned to a program file, it is set as current role of every process that executes this program. However, the role can be changed at any time by all implicit or explicit mechanisms mentioned above, e.g. by changing the process owner. Initial roles are typically used for login programs, which need special privileges for authentication, but have to switch to a new owner's default role afterwards.

Two special initial role values affect implicit role transitions:

[5] See Webserver example in section 6.

role_inherit_parent (default value): Get initial role setting from filesystem parent object. If there is no parent object, use root dir default value role_use_forced_role. This default value allows to set an initial role for whole directory trees.

role_use_forced_role (root dir default value): Only use the forced role setting.

As usual, the inheritance implies the notion of effective values:

- initialrole_{tn}(f:file):role := initial role value of file f at time t
- $\text{effinitialrole}_{tn}$(f:file):role := effective initial role of file f at time n, including inheritance from parent filesystem objects

The effective initial role is derived as follows:

$$\text{effinitialrole}_{tn}(\text{f}) := \begin{cases} \text{if initialrole}_{tn}(\text{f}) = \text{inherit\_parent} \ \wedge \exists \text{parent(f)} : \text{effinitialrole}_{tn}(\text{parent(f)}) \\ \text{if initialrole}_{tn}(\text{f}) = \text{inherit\_parent} \wedge \nexists \text{parent(f)} : \text{role\_use\_forced\_role} \\ \text{if initialrole}_{tn}(\text{f}) \neq \text{inherit\_parent} : \text{initialrole}_{tn}(\text{f}) \end{cases} \quad (17)$$

Initial roles for program files change the implicit role transition on execution from rule 4 as follows:

$$\text{execute}_{tn}(\text{p},\text{f}) \Rightarrow \ \text{currentrole}_{tn+1}(\text{p}) := \begin{cases} \text{if effinitialrole}_{tn}(\text{f}) = \text{role\_use\_forced\_role} : \textit{follow rule } 20 \\ \text{if effinitialrole}_{tn}(\text{f}) \neq \text{role\_use\_forced\_role} : \text{effinitialrole}_{tn}(\text{f}) \end{cases} \quad (18)$$

Forced Roles While initial roles are only set as temporary current roles, forced roles are kept until either another program with initial or forced role is executed or the process actively changes to a compatible role. Certainly, it has to be kept in a process attribute for later use.

All other implicit mechanisms, e.g. when changing the process owner, do not affect the current role while a forced role is set.

Forced roles are useful for those server program encapsulation cases, where a server program must always run with the same privileges for all process owners.

There are several special forced role values which affect implicit role transitions:

role_inherit_user: Always set the (new) process owner's default role as current role when executing this program or when changing process owner while this program is executed. This can be used for login shells to make sure that the user's default role is used.

role_inherit_process: Keep the current role when executing this program or changing process owner while this program is executed. This value lets subprograms keep the forced role of their parents in all cases.

role_inherit_parent (default value): Get forced role setting from filesystem parent object. If there is no parent object, use root dir default value role_inherit_up_mixed. This default value allows to set a forced role for whole directory trees.

role_inherit_up_mixed (root dir default value): Keep the current role when executing this program, but set it to new owner's default role when changing the process owner. This is the standard role model behaviour as mentioned above.

The forced role default settings make all programs run with the process owner's default role, which is the desired behaviour in most cases.

- forcedrole_{tn}(f:file):role := forced role value set for file f at time n
- $\text{effforcedrole}_{tn}$(f:file):role := effective forced role of file f at time n, including inheritance from parent filesystem objects

The effective forced role is derived as follows:

$$\text{effforcedrole}_{tn}(\text{f}) := \begin{cases} \text{if forcedrole}_{tn}(\text{f}) = \text{inherit\_parent} \ \wedge \exists \text{parent(f)} : \text{effforcedrole}_{tn}(\text{parent(f)}) \\ \text{if forcedrole}_{tn}(\text{f}) = \text{inherit\_parent} \wedge \nexists \text{parent(f)} : \text{role\_inherit\_up\_mixed} \\ \text{if forcedrole}_{tn}(\text{f}) \neq \text{inherit\_parent} : \text{forcedrole}_{tn}(\text{f}) \end{cases} \quad (19)$$

Forced roles for program files extend the implicit role transition on execution from rule 18 as follows:[6]

$$\text{execute}_{tn}(\text{p},\text{f}) \Rightarrow \ \text{currentrole}_{tn+1}(\text{p}) := \begin{cases} \text{if effforcedrole}_{tn}(\text{f}) = \text{role\_inherit\_user} : \text{defrole}_{tn}(\text{owner(p)}) \\ \text{if effforcedrole}_{tn}(\text{f}) = \text{role\_inherit\_process} : \text{currentrole}_{tn}(\text{p}) \\ \text{if effforcedrole}_{tn}(\text{f}) = \text{role\_inherit\_up\_mixed} : \text{currentrole}_{tn}(\text{p}) \\ else : \text{effforcedrole}_{tn}(\text{f}) \end{cases} \quad (20)$$

The effective forced role value from the executed file is copied to the respective process attribute.

$$\text{execute}_{tn}(\text{p},\text{f}) \Rightarrow \ \text{forcedrole}_{tn+1}(\text{p}) := \text{effforcedrole}_{tn}(\text{f}) \quad (21)$$

The implicit role transition on process owner changes from rule 3 is modified as well:

$$\text{changeowner}_{tn}(\text{p},\text{u}) \Rightarrow \ \text{currentrole}_{tn+1}(\text{p}) := \begin{cases} \text{if forcedrole}_{tn}(\text{p}) = \text{role\_inherit\_user} : \text{defrole}_{tn}(\text{u}) \\ \text{if forcedrole}_{tn}(\text{p}) = \text{role\_inherit\_process} : \text{currentrole}_{tn}(\text{p}) \\ \text{if forcedrole}_{tn}(\text{p}) = \text{role\_inherit\_up\_mixed} : \text{defrole}_{tn}(\text{u}) \\ else : \text{forcedrole}_{tn}(\text{p}) \end{cases} \quad (22)$$

3.6 Standard Administration

In the standard case, all RC model administration is done through one or more roles, which have their *Admin Type* attribute set to *Role Admin*. This value gives

[6] If the initial role is set to the default value role_use_forced_role.

full administrative privileges, overriding the separation scheme presented in the next subsection, but no access rights to objects.

Administration tasks are definition of roles and types, specification of compatibilities, assignment of default, initial and forced roles to users and program files and assignment of types to objects.

3.7 Separation of Administration Duty

Security administration should best be separated into several tasks, performed by several different administrators, which have to cooperate to provide additional privileges.

The Role Compatibility Model contains a separation of administration duty scheme, which allows to generate limited workgroups as well as enforce cooperation of two or more roles for most administration tasks. However, the *Admin Type* role attribute makes the separation scheme completely optional.

As the separation of duty related settings can only be changed by roles with *Admin Type* set to *Role Admin*, removing these roles or resetting their *Admin Type* value fixes the separation for future use.

Admin Roles Every role definition contains a set of roles, called *Admin Roles*, which processes performing this role are allowed to administrate. For many settings, e.g. the compatibility sets, additional privileges are required, which are explained below.

- adminroles(r:role):set of roles := set of administrated roles for role r
- $\text{administraterole}_{tn}$(p:process, r:role) := process p administrates settings of role r at time n

$$\text{administraterole}_{tn}(\text{p},\text{r}) \Rightarrow \; r \in \text{adminroles}_{tn}(\text{currentrole}_{tn}(\text{p})) \qquad (23)$$

The *Admin Roles* set of any role can only be changed by roles with *Admin Type* value *Role Admin*.

- $\text{changeadminroles}_{tn}$(p:process, r:role) := process p changes the set of admin roles of role r at time n

$$\text{changeadminroles}_{tn}(\text{p},\text{r}) \Rightarrow \; \text{admintype}_{tn}(\text{currentrole}_{tn}(\text{p})) = \text{role\_admin} \quad (24)$$

Assign Roles Another set of roles contained in all role definitions is called *Assign Roles*. It defines, which roles processes running this certain role are allowed to assign as compatible role to roles, as default role to users or as initial or forced role to program files or processes.

- assignroles(r:role):set of roles := set of assignable roles for role r
- addcomprole_{tn}(p:process, r_1:role, r_2:role) := process p adds role r_1 to the set of compatible roles of role r_2 at time n

- $\text{assigndefrole}_{tn}$(p:process, r:role, u:user) := process p assigns default role r to user u at time n
- $\text{assigninitialrole}_{tn}$(p:process, r:role, f:file) := process p assigns initial role r to program file f at time n
- $\text{assignforcedrole}_{tn}$(p:process, r:role, f:file) := process p assigns forced role r to program file f at time n
- $\text{assignforcedrole}_{tn}$($p_1$:process, r:role, p_2:process) := process p_1 assigns forced role r to process p_2 at time n

$$\begin{aligned}\text{addcomprole}_{tn}(\text{p}, \text{r}_1, \text{r}_2) \Rightarrow\ & \text{r}_1 \in \text{assignroles}_{tn}(\text{currentrole}_{tn}(\text{p})) \\ & \wedge\ \text{r}_2 \in \text{adminroles}_{tn}(\text{currentrole}_{tn}(\text{p}))\end{aligned} \tag{25}$$

Default roles can only be assigned to users, if both the old and the new role are in the set of *Assign Roles*. This restriction, together with the sets of compatible roles, creates a range of reachable roles, which easily forms a workgroup.

$$\text{assigndefrole}_{tn}(\text{p}, \text{r}, \text{u}) \Rightarrow\ \text{r}, \text{defrole}_{tn}(\text{u}) \in \text{assignroles}_{tn}(\text{currentrole}_{tn}(\text{p})) \tag{26}$$

To set an initial or forced role for a program file or process object, the additional right MODIFY_ATTRIBUTE to the type of the object is needed.

$$\begin{aligned}&\text{assigninitialrole}_{tn}(\text{p}, \text{r}, \text{f}) \Rightarrow \\ &\quad r \in \text{assignroles}_{tn}(\text{currentrole}_{tn}(\text{p})) \\ &\wedge\ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{efftype}_{tn}(\text{f}), \text{MODIFY\_ATTRIBUTE})\end{aligned} \tag{27}$$

$$\begin{aligned}&\text{assignforcedrole}_{tn}(\text{p}, \text{r}, \text{f}) \Rightarrow \\ &\quad \text{r} \in \text{assignroles}_{tn}(\text{currentrole}_{tn}(\text{p})) \\ &\wedge\ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{efftype}_{tn}(\text{f}), \text{MODIFY\_ATTRIBUTE})\end{aligned} \tag{28}$$

$$\begin{aligned}&\text{assignforcedrole}_{tn}(\text{p}_1, \text{r}, \text{p}_2) \Rightarrow \\ &\quad \text{r} \in \text{assignroles}_{tn}(\text{currentrole}_{tn}(\text{p}_1)) \\ &\wedge\ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}_1), \text{type}_{tn}(\text{p}_2), \text{MODIFY\_ATTRIBUTE})\end{aligned} \tag{29}$$

Changes to the *Assign Roles* set of any role are restricted to roles with *Admin Type* value *Role Admin*.

- $\text{changeassignroles}_{tn}$(p:process, r:role) := process p changes the set of assign roles of role r at time n

$$\text{changeassignroles}_{tn}(\text{p}, \text{r}) \Rightarrow\ \text{admintype}_{tn}(\text{currentrole}_{tn}(\text{p})) = \text{role\_admin} \tag{30}$$

Special Rights Some special rights to types have been defined:

ADMIN: Administrate this type, i.e., change type name or remove type.
ASSIGN: Assign this type to objects. Additionally, MODIFY_ATTRIBUTE to the previous type of the object is needed.
ACCESS_CONTROL: Change type compatibility settings for this type and all requests, which are no special rights.
SUPERVISOR: Change type compatibility settings for this type for all special rights. If no role has SUPERVISOR right or *Admin Type* set to *Role Admin*, the special right settings can no longer be changed.

- specialrights := {ADMIN, ASSIGN, ACCESS_CONTROL, SUPERVISOR}
- $\text{administratetype}_{tn}$(p:process, t:type) := process p administrates type t at time n
- assigntype_{tn}(p:process, t:type, o:object) := process p assigns the type t to object o at time n
- $\text{changetypecomp}_{tn}$(p:process, r:role, t:type, a:access_type) := process p adds or removes access type a to or from the type compatibility set of role r to type t at time n

$$\text{administratetype}_{tn}(\text{p}, \text{t}) \Rightarrow \\ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{t}, \text{ADMIN}) \tag{31}$$

$$\text{assigntype}_{tn}(\text{p}, \text{t}, \text{o}) \Rightarrow \\ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{t}, \text{ASSIGN}) \\ \wedge \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{efftype}_{tn}(\text{o}), \text{MODIFY\_ATTRIBUTE}) \tag{32}$$

$$\text{changetypecomp}_{tn}(\text{p}, \text{r}, \text{t}, \text{a}) \wedge \ \text{a} \notin \text{specialrights} \Rightarrow \\ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{t}, \text{ACCESS\_CONTROL}) \\ \wedge \text{r} \in \ \text{adminroles}(\text{currentrole}_{tn}(\text{p})) \tag{33}$$

$$\text{changetypecomp}_{tn}(\text{p}, \text{r}, \text{t}, \text{a}) \wedge \ \text{a} \in \text{specialrights} \Rightarrow \\ \text{compatible}_{tn}(\text{currentrole}_{tn}(\text{p}), \text{t}, \text{SUPERVISOR}) \\ \wedge \text{r} \in \ \text{adminroles}(\text{currentrole}_{tn}(\text{p})) \tag{34}$$

3.8 Lifetime Limits

All compatibility, *Admin Roles* and *Assign Roles* settings have an optional time-to-live parameter. After the given time, the set of requests for a type compatibility setting gets cleared by the system, while compatible, admin or assign roles get removed from their set.

With lifetime limits, temporary additional rights do not require manual action to be revoked and thus avoid this typical situation of unnecessary permanent rights. Certainly, time-to-live settings rely on the correct system time to be always maintained.

4 Implementation

The Role Compatibility model has been implemented as a decision module for the RSBAC framework[RSBAC] and makes extensive use of its infrastructure.

The Rule Set Based Access Control (RSBAC) system is an open source security extension to current Linux kernels, which has been continuously developed by the author for several years.

RSBAC was designed according to the Generalized Framework for Access Control (GFAC)[Abrams+90] to overcome the deficiencies of access control in standard Linux systems, and to make a flexible combination of security models as well as proper access logging possible.

Only smaller RC changes and adaptions to changes of the framework have been made from November 1999 till November 2001, like initial roles or the extension for new target types. From November 2001, the RC model implementation has been moved to generic RSBAC lists and the original limit of 64 roles and 64 RC types per target type has been removed. Also, the new network target types and time limits have been included.

4.1 Roles and Types

Role and type definitions are registered as persistent generic lists with their index number as list index.

Role data includes a role name for human use and the simple attributes *Admin Type*, *Default fd create type*, *Default process create type*, *Default process chown type*, *Default process execute type* and *Default IPC create type*.

Type data only includes the type name for human use.

4.2 Role Compatibility, Admin and Assign Roles

Persistent generic lists of lists without data are registered for Role Compatibility, Admin and Assign Roles. The first level index is the role number of the set owner, the second level index the role number of the set member.

Set membership is tested by existence of the second level entry.

4.3 Type Compatibility

For each RSBAC target type one persistent list of lists is registered. The first level index is the role number, the second level index is the type number. Only second level data is used, it contains the set of allowed requests coded as a 64 bit integer used as bit set.

Absence of an item is interpreted as the default value of an empty set.

4.4 Program Based Access Control with Initial and Forced Roles

Unlike the above items, initial and forced role settings are implemented as attributes of file objects, the forced role also as attribute of process objects. They are kept and provided by the RSBAC General Data Structures component.

4.5 Access Control Decision and Notification

RSBAC request decisions and respective automatic attribute updates are performed in the decision function rsbac_adf_request_rc and the notification function rsbac_adf_set_attr_rc, which are called from the ADF dispatcher functions.[7]

Administration and role changing decisions are made in the respective individual functions, which have been implemented as additional system calls.

For most requests, the decision function only takes the process current role, the object type and the request and matches them against the type compatibility settings.

The notification function performs all implicit role and type changes for existing or newly created processes and objects as specified above.

The time values *tn* and *tn+1* used in the specification are interpreted as *at the time of the decision request call* and *directly after the notification call*. Since all attribute changes from the specification rules are either for the requesting process only, or for a newly created object, which cannot be accessed by any process before notification has completed, race conditions can only occur with active administration.

5 Comparison with RBAC and DTE Models

5.1 Role Based Access Control (RBAC)

Model Description The RBAC access control model as described in [FerKuh92] defines subjects, roles and transactions. A transaction is defined as a transformation procedure plus its necessary data accesses. All subject activities in a system are performed through transactions, but not the system tasks like identification or authentication.

The RBAC model defines three basic rules:

1. Role assignment: A subject can execute a transaction only if the subject has selected or been assigned a role.
2. Role authorization: A subject's active role must be authorized for the subject.
3. Transaction authorization: A subject can execute a transaction only, if the transaction is authorized for the subject's active role.

Additionally, transformation procedures, objects and access modes can be separated, and an access function can define, which role executing which transaction may access which objects with which access modes.

In [FeCuKu95], the term *operation* is introduced, which denotes an access with a certain mode to a set of objects. Roles are then authorized for operations and no longer for transactions or transaction procedures. Also, users are distinguished from subjects. A subject is an active entity, performing operations on

[7] See e.g. [RSBAC,Ott2001,Ott2001a] for RSBAC structure.

behalf of one user at a time, and has a set of active roles, for which the user must be authorized.

Roles may be members of other roles, so that membership in a subrole implies the membership in all parent roles, including all their authorizations. The possible membership in several roles requires the definition of *mutual exclusion* to preserve separation of duty, i.e., pairs of roles which may not share the same member or, in the revised model, which may not be activated at the same time by the same subject.

Finally, the RBAC model defines static and dynamic *capacities* of roles, the first being the maximum number of members, the latter the maximum number of subjects having the role activated.

In [Ferraiolo+2001], a NIST standard for RBAC models has been proposed. It adds the notion of *user sessions*, which allow to selectively activate or deactivate roles within a session. All RBAC features are grouped into *Core RBAC*, which contains the basic functionality, *Hierarchical RBAC* to define role hierarchies and *Constrained RBAC* with *Static* and *Dynamic Separation of Duty Relations*. All RBAC separation of duty relates to what roles from the assigned set of roles can be used by a single user at the same time. Of this, mutual exclusion is only a subset.

Comparison to RC Model Similar to the RBAC model, RC defines subjects as processes, the active entities within a system, working on behalf of users with a current role and performing accesses to objects. However, in RC model each process can only have one active role at a time, avoiding the complex scheme of mutual exclusion. This means that in some cases several roles with overlapping rights may have to be defined.

The RBAC set of authorized roles of users is covered by the RC set of compatible roles, which are reachable from the user's default role. Even more, after changing into a role there might not be a way back to the original role. This can effectively avoid uncontrolled flow of information through process memory by switching to another role with higher privileges and then back to the original role.

The RC model can even simulate the transaction concept from the first RBAC version through program based roles and separate types for the program files: Transaction authorization is mapped as EXECUTE right on the program object type, while operations allowed for a transaction can be assigned as compatibilities to the program's assigned role.

The RBAC model does not have an equivalent for RC type abstraction, program based roles and separation of administration duty. Time limits can only be simulated through dynamic mutual exclusion.

RC only lacks role capacities, which were not considered as useful.

5.2 Domain and Type Enforcement (DTE)

Model Description As stated in [Badger+95], Domain and Type Enforcement is based on an enhanced version of Type Enforcement (TE). The main additions

to the original model are a high level policy specification language and a human readable format of attribute values in the runtime policy database.

Type Enforcement is a table based access control model. Active entities, the subjects, have an attribute *Domain*, while passive entities, the objects, have a *type* attribute. Possible accesses by subjects to objects are grouped into the *access modes* read, write, execute and traverse.

A global *Domain Definition Table (DDT)* contains the allowed interactions, where domains and types form rows and columns, and each cell holds a set of access modes.

Subject-to-subject access control is based on a global *Domain Interaction Table (DIT)* with subjects as both descriptors and, again, a set of access modes, e.g. signal, create or destroy, in the cells.

In contrast to the original TE model, DTE supports *implicit* attribute maintenance. This means that values may be only kept on a higher level of the directory and file hierarchy, but are used for all levels below as well. Also, the specification language allows to specify types by lookup path prefixes.

The first process on a system, the init process, gets a predefined initial domain assigned. Each process can enter another domain by executing a program bound to it, a so-called *entry point*. An entry point may be executed to explicitely enter one of its associated domains, if the subject's current domain has *exec* right on the target domain. The *auto* access right to a domain automatically selects this domain, if one of its entry points gets executed.

The user-domain relationship is entirely built on entry points like command shells etc. However, a DTE aware login program can select from all domains associated with an entry point to avoid individual copies for each domain.

Comparison to RC Model While the RC model makes role assignments based on users and programs, both represented by processes, the DTE model itself avoids the concept of users and only focuses on programs. User representation and role assignment are placed under the discretion of unspecific DTE aware applications outside the scope of the model.

Another DTE drawback is that roles can only be changed through entry point programs, while the RC model allows to dynamically switch to compatible roles within one single application and to default roles on every change of the process owner. Dynamic role changes are specially useful for user based server programs.

Finally, DTE administration concepts were not mentioned in [Badger+95] and thus remain unclear.

6 Application Example

This section describes how the RC model is used to secure a typical server system. The approach given here can easily be adapted to many other types of services.

6.1 Base Protection

Those parts of the RSBAC and RC access control setup, which are applicable for all types of systems, are called *Base Protection*. Objects to be protected include the basic directory structure, executables, libraries, configuration files, kernel objects and boot loaders, raw devices, account and authentication data, log files, home directories etc.

Generally, for each of these object categories one RC type gets defined and assigned to the individual objects, and all existing roles get appropiate type compatibility settings to these types.

As an example, the type *Executables* is assigned to the directories /bin and /usr/bin, which contain executable files. All of these then inherit the effective type *Executables*. As soon as the type compatibilities of all roles to this type are set accordingly, executables are fully protected. Furthermore, after setting *all* desired executables to this type, one can safely remove the execute right to all other types and thus avoid any execution of unprotected and possibly malicious files.

6.2 Service Encapsulation

While the Base Protection secures the base system, the different services are additionally encapsulated to restrict them to the absolutely necessary. Here program based roles are most useful.

A typical example of good RC usage is a virtual Webserver system with an arbitrary number of customers, who want to use their own CGI scripts with private data.

We use a forced role *Webserver*, which gets assigned to the Webserver binary. This role may not access any of the base protection types except the mapping of libraries. The general Webserver logging type *Webserver Log* can be accessed to create and append to log files.

Each customer *C* gets a separate directory tree, three RC types, called *Web-Data-C*, *CGI-Program-C* and *Private-Data-C*, and three roles, called *Webserver-C*, *Upload-C* and *CGI-C*.

The general Webserver role may not access any customer data. Instead, a serving process changes to the compatible role Webserver-C when serving content for customer C. The role handling can e.g. be implemented in a simple Apache module.

Role Webserver-C may read Web-Data-C and execute CGI-Program-C. The CGI folder for customer C has a forced role setting of CGI-C, which gets inherited to all programs in it. Thus, when one of C's CGI programs is run, it uses role CGI-C and gets limited access to all of C's data, specially Private-Data-C.

Finally, the upload account for customer C gets the default role Upload-C, which has read and write, but no execute access to all three types. Access to any other type is denied.

6.3 Further Refinement

The setup presented here makes use of several RC features like compatible roles and program based roles to protect the system and the customers from each other and from client systems.

It can easily be extended by network access control to prevent unwanted connections by customer CGIs or a compromised server program, e.g. to avoid the spreading of worms.

For a complete setup, every single service can be encapsulated with individual roles and types. The focus should certainly lie on the network services.

7 Conclusion

Practical experience with server systems using the Role Compatibility model for access control shows that base protection and service encapsulation are possible without drawbacks in usability. All protection requirements of these systems could be solved by proper RC configuration, while the well-known RBAC and DTE models each show several deficiencies.

The RC model as presented in this paper proved to be easy to use in simple setups, but also very flexible and powerful in complex environments. Combined with the RSBAC concept of Network Templates, even access to and from remote systems can be effectively controlled.

References

[Abrams+90] Abrams, M. D., Eggers, K. W., La Padula, L. J., Olson, I. M., A Generalized Framework for Access Control: An Informal Description, Proceedings of the 13th National Computer Security Conference, Oktober 1990

[Badger+95] Badger, L., Sterne, D. F., Sherman, D. L., Walker, K. M., Haghighat, S. A., Practical Domain and Type Enforcement for UNIX, 1995 IEEE Symposium on Security and Privacy

[FerKuh92] Ferraiolo, D., Kuhn, R., Role-Based Access Control, Proceedings of the 15th National Computer Security Conference, 1992

[FeCuKu95] Ferraiolo, D. F., Cugini, J. A., Kuhn, D. R., Role-Based Access Control (RBAC): Features and Motivations, Proceedings of the Computer Security Applications Conference 1995

[Ferraiolo+2001] Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., Chandramouli, R., Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and Systems Security, Vol. 4, No. 3, August 2001

[FiHueOtt98] Fischer-Hübner, S., Ott, A., From a Formal Privacy Model to its Implementation, Proceedings of the 21st National Information Systems Security Conference (NISSC '98), Arlington, VA, 1998, http://www.rsbac.org/niss98.htm

[Jansen98] Jansen, W. A., A Revised Model for Role-Based Access Control, NIST IR 6192, 1998

[Ott97] Ott, A., Regelsatz-basierte Zugriffskontrolle nach dem “Generalized Framework for Access Control”-Ansatz am Beispiel Linux, Diplomarbeit, Fachbereich Informatik, Universität Hamburg, 10. November 1997, http://www.rsbac.org/dipl-ps.zip

[Ott2001] Ott, A., Rule Set Based Access Control (RSBAC), Paper for the Snow Linux Event / Unix.nl congress ”Reliable Internet”, Waardenburg, 14th of September 2001, http://www.rsbac.org/unix-nl

[Ott2001a] Ott, A., The Rule Set Based Access Control (RSBAC) Linux Kernel Security Extension, Paper for the 8th International Linux Kongress, Enschede, 28th to 30th of November 2001, http://www.rsbac.org/linux-kongress

[RSBAC] Ott, A., RSBAC Homepage, http://www.rsbac.org

[Sherman+95] Sherman, D. L., Sterne, D. F., Badger, L., Murphy, S. L., Walker, K. M., Haghighat, S. A., Controlling Network Communication with Domain and Type Enforcement, TIS Technical Report TISR 523, 1995

Using the Java Sandbox for Resource Control

Almut Herzog and Nahid Shahmehri

Department of Computer and Information Science, Linköpings universitet,
SE-581 83 Linköping, Sweden
{almhe, nahsh}@ida.liu.se

Abstract. Java's security architecture is well known for not taking the security aspect of availability into account. This has been recognised and addressed by a number of researchers and communities. However, in their suggested resource-aware Java environments, policies for resource control have so far been stated in proprietary, sometimes hard-coded, or undocumented ways. We set out to investigate if standard Java permission syntax can be used to formulate policies for resource management of high-level resources and if the enforcement of resource policies can successfully be done by the standard Java access controller. Such a solution would neatly fit in the existing Java security architecture.
We have implemented resource control for the serial port and for the file system by using the Java permission syntax for stating policies and the standard Java access controller as the enforcement mechanism. The implementation was straightforward and resulted in an API useful also for control of other high-level resources than the serial port and file system. A performance test showed that such resource management easily leads to excessive invocations of the access controller and that optimisation steps are necessary to prevent performance penalties.

Keywords. Java; Availability; Resource Management; Policy; Performance.

1 Introduction

In comparison to other programming languages, Java has good architectural support for access control to resources. While the CPU is under the hard-coded control of the Java virtual machine thread scheduler, and memory is allocated through object allocation and deallocated by the garbage collector, initial access to higher-level resources is granted by an optional security manager. Higher-level resources are e.g. the file system, I/O device APIs (application programmer interface), threads, sockets or properties. The security manager can be switched on for applications and provides a so-called sandbox that consults a policy—based on the contents of system policy files and user policy files—prior to granting access to resources.

Despite the support for initial access control, the Java sandbox lacks control over the extent to which a resource can be used. This has been recognised in multiple sources, e.g. [1][2][3]. Once access to a resource has been granted to a piece of code, the code is free to use it to any extent. For example, if a Java application is allowed to

write in /tmp, there is no control over how many bytes it writes or how many files it creates in /tmp. However, such control is needed to guarantee availability of a resource and, consequently, to counter denial-of-service attacks. Otherwise, writing excessive amounts of data to a temporary directory fills up the disk and denies other applications the creation of files in that file system. On Unix systems, creating an excessive amount of files results in the exceeding of kernel limits and effects also other user processes.

This problem has been addressed by a number of contributions (cf. §*4 Related Work*) that result in resource-aware Java environments. However, little is mentioned on the syntax used for resource management policies. Yet other work ([4][5][6], cf. §*4 Related Work*) has been experimenting with new permission languages that allow advanced policies such as time-based, context-based, user-based or hierarchical permissions. However, these languages are no longer based on Java's permission syntax nor are they enforced by the Java access controller.

Our work sets out to prove that resource management of higher-level resources is possible using the syntax of Java permissions and the standard Java access controller. The advantage of this solution is that resource control is integrated in the standard Java security architecture and implemented within the existing sandbox.

In this work, we do not deal with resource control of low-level resources such as CPU (central processing unit) or memory. Low-level resources are not mediated by the Java virtual machine, i.e. there is no way to ask the Java virtual machine for the CPU or a piece of memory, instead thread priorities must be manipulated or objects allocated. Thus, these resources cannot well be handled by the high-level access controller, which itself is implemented in Java. Access control for CPU and memory is preferably addressed within the core Java classes through modification of the Java thread scheduler, the heap manager and the garbage collector. Such work has been done for real-time Java and is described in e.g. [7][8][9].

The rest of the paper is organised as follows. Section 2 recalls existing Java permissions, their syntax and the existing enforcement mechanism. Section 3 describes our implementation of resource control using Java permissions and the Java access controller. It also describes our performance test. Section 4 supplies detailed references to related work. Section 5 concludes this paper and gives an outlook to future work.

2 Existing Permission Syntax and Enforcement

Java's current access control model for high-level resources, i.e. resources that are mediated by the Java virtual machine, is built of a policy and an enforcement mechanism.

The policy normally resides in files. There is a *system policy file*, usually within the installation directory of the Java virtual machine, which sets a system-wide policy. A *user policy file*, usually residing in the home directory of the user, can set a stricter policy for this user. These policy files can be overridden by a command line option to the Java virtual machine. Also, the policy can be set from within the application (if the effective policy allows the application to do so).

A policy consists of sets of positive permissions. A set of permissions is assigned to a given code base and/or a signer. The example policy file of Fig. 1 contains Chris' policy. Chris trusts code that was signed by Alice with full permissions. Chris allows Bob's tools only to read the file /tmp/a. Code in the jar-file app.jar is only allowed to read from /tmp/b.

```
keystore "file:/home/user/chris/.keystore";

grant signedBy "alice" {
  permission java.security.AllPermission;
};

grant signedBy "bob", codeBase "file:/sw/lib/tool.jar" {
  permission java.io.FilePermission "/tmp/a", "read";
};

grant codeBase "file:/sw/lib/app.jar" {
  permission java.io.FilePermission "/tmp/b", "read";
};
```

Fig. 1: Example policy file

A Java permission is identified by its class name, a target string and an action string. File permissions are e.g. identified by their class name java.io.FilePermission. Their target string identifies the file(s) to which the permission applies, and the action string denotes which actions are allowed for the given file(s) (cf. Fig. 2).

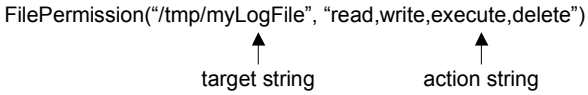

Fig. 2: File permission syntax

Even though the two strings that initialise a permission are referred to as target string and action string, the semantics of the two strings are up to the implementing sub-class. A simpler subclass of permission is *basic permission* which only considers the target string and ignores the action string. This is e.g. used for runtime permissions—e.g. RuntimePermission("setPolicy").

The policy is enforced by Java's security manager, or rather by the Java access controller that is wrapped by the security manager (lines 9-12 of Fig. 3). Whenever a resource that is subject to access control is to be accessed, the access controller is invoked to check that a positive permission for this resource exists or that the permission is implied by another permission. For example, the permission to read all files *implies* the permission to read /tmp/myLogFile.

The typical way of doing access control is shown in Fig. 3. Prior to the actual access of the resource—in this case the policy object (line 7), the needed access

permission is constructed and submitted to the access controller (line 5). This access control is only performed if a security manager is installed (lines 3-4). By default, Java applications run *without* a security manager. The access controller checks this permission against the effective policy (line 11). If the permission is implied by the current policy for every piece of code within the call chain (i.e. on the call *stack*), the access control successfully returns and access to the resource is permitted (line 7). If the permission is not granted, an access control exception is thrown.

```
from SUN's J2SDK 1.3.1: java.security.Policy.Java (our comments)
 1    public static void setPolicy(Policy policy) {
 2      // Do access control only if a security manager is installed
 3      SecurityManager sm = System.getSecurityManager();
 4      if (sm != null)
 5            sm.checkPermission(new SecurityPermission("setPolicy"));
 6      // The access check was successful, access resource now
 7      Policy.policy = policy;
 8    }

  from SUN's J2SDK 1.3.1: java.lang.SecurityManager (our comments)
 9    public void checkPermission(Permission perm) {
10      // Delegate access control to AccessController
11      java.security.AccessController.checkPermission(perm);
12    }
```

Fig. 3: The setting of a policy is subject to typical Java access control, example from SUN's J2SDK source code

Fig. 3 illustrates also our research problem for this paper. The access check is performed before the resource is accessed. Once the access check has successfully completed, the resource is accessed without further restrictions.

For complete descriptions of the Java security architecture refer to [10][1][2][11].

3 Resource Control with the Java Sandbox

Within this project, we identify three typical cases of resource control (cf. Table 1) that can be used for any resource. The cases can be roughly grouped into *irrevocable* and *revocable* resource control. By definition, the two groups are mutually exclusive. However, in our solution, parts of a resource can be granted irrevocably, while other parts of a resource can be granted in a revocable fashion. For instance, an application may be allowed to open a file for an arbitrary length of time (irrevocable) but is only allowed to write a maximum of 500k to it (revocable).

Irrevocable resource control (described in the first column of Table 1) is the existing, standard way of making use of Java permissions. The permission is constructed for the needed resource and the action that is to be performed on it. No accounting data is needed for the permission check; and access is granted without further limits.

The two revocable types are implemented by us and make use of accounting data.

In the first of these two revocable cases—shown in the second column of Table 1—a permission is checked at regular intervals through a timer thread. *This is needed in cases where a piece of code holds a lock on a resource* and the lock shall be revoked when a certain condition is met. For example, an open file shall be released after 2 seconds, or a database connection is to be terminated after 60 minutes.

The second case of using accounting data—shown in the third column of Table 1—is more advanced. Not only is accounting data needed for constructing the permission for the first resource access, it is also needed for any subsequent access and accounting data is updated depending on the outcome of the actual access. The typical example is the writing of a file where one wants to limit the number of bytes that can be written to disk —either in total or for one specific file. Accounting data must be updated depending on if the disk write succeeded or failed. *This second type of revocable resource control is needed for fine-grained,* continuous *monitoring of access to a resource.*

Table 1: Different types of access control to resources

Irrevocable access control	**Revocable access control**	
Construct permission, check	*Expiration* of the access to a resource	Construct permission with accounting data, check, perform checked action, *update accounting data based on outcome*
Examples		
Typical Java permissions as described in Fig. 3	Limiting the time a file can be open	Limiting the number of bytes written to a file
Accounting data		
–	Timestamp when the file was opened	Bytes written per file

In order to achieve resource control as shown in columns two and three of Table 1, the following issues need to be worked with.

Table 2. Steps needed for making the Java virtual machine resource-aware.

1. A syntax for expressing a resource policy as Java permissions.
2. An enforcement mechanism for the permissions, which in our case is given with the standard Java access controller.
3. A decision of which code base is accountable for the resource use.

4. A data structure for keeping accounting data.
5. An identification of the places in the code where accounting data needs to be updated.
6. An identification of the places in the code where the enforcement mechanism must be invoked.

These issues are dealt with in the following sections. Section 3.1 describes the syntax we use for Java permissions for resource control. Section 3.2 deals with issues of permission enforcement and the timely update of accounting data. Section 3.3 provides performance data for our solution.

3.1 Permissions for Resource Control

In this section, we describe how we made use of the syntax for standard Java permissions to integrate resource control (cf. Table 2, item 1).

We integrate resource limits in the target and action strings of permissions. Following our approach described in Table 1, we use one syntax for *expiration* of access for the whole resource and another for fine-grained, *continuous* access control. The target string can be extended with a target-specific resource limit

```
<target>[:<global_target_limit>]
```

that restricts the access to the resource as a whole. The action string—if present—can receive an action-specific resource limit:

```
<action>[:<action_limit>]
```

For convenient handling, we build an abstract permission class[1] called ResPermission. This abstract class can then be subclassed for specific permissions. The ResPermission class extends java.security.Permission and supports e.g. the parsing of target and action string for resource limits. The implemented method implies() returns true when the resource limit of the implied permission is lower than the resource limit of the premiss. Targets and actions must be a full match. E.g.:

```
new ResPermission("/dev/term/b:1000").implies(
                        new ResPermission("/dev/term/b:20"))
```
returns true

```
new ResPermission("/dev/*:1000").implies(new
                        new ResPermission("/dev/term/b:20"))
```
returns false

because wildcard-matching is not implemented. This must be done in a suitable subclass.

Below are some instantiations of subclasses to ResPermission. They further illustrate the use of resource limits within target and action string.

Example: The serial port on /dev/term/a shall only be accessible for 50 seconds.

```
almhe.sec.ResCommPortPermission("/dev/term/a:50000")
```

[1] An abstract class is a class that cannot be instantiated, only subclassed. It consists of method signatures but can also contain fully implemented methods.

Example: The file /tmp/a shall only be open for reading or writing for 20 seconds.

```
almhe.sec.ResFilePermission("/tmp/a:20000", "read, write")
```

Example: The file /tmp/b can be open for an unlimited amount of time. A maximum of 1K may be written to the file.

```
almhe.sec.ResFilePermission("/tmp/b", "write:1024");
```

Example: The code executing under the following permission is only allowed to create a total of 500k in all the files it creates.

```
almhe.sec.ResFilePermission("<<TOTAL>>", "write:512000");
```

3.1.1 Extended Resource Control

So far we have only been dealing with one dimension of resource access, i.e. at the moment only one limit is allowed per target and action. However, further dimensions can be thought of e.g. not only the *length of time* a resource is accessed but also the *number of times* a resource can be accessed, created, modified or deleted. This could be solved using the same syntax but iterating the resource limit, preferably under the presence of an explanatory tag, e.g.

```
<target>[:<<tag_name>><global_target_limit>]*.
```

Following such an extended syntax, the following permission `ResCommPortPermission("/dev/term/a:<length_ms>50000:<num_times>10")` would allow the access to /dev/term/a for a maximum of 5000ms but no more than 10 times during the life of the Java virtual machine. This syntax could also be used for the action string limit. This extended syntax has not been implemented yet, it simply shows the way our proposed syntax could develop to comprise even finer-grained resource control.

3.2 Enforcement of Resource Control

The enforcement mechanism for our approach was given with the Java access controller. However, the access controller must be invoked at the correct position in the code with the correct permission that is to be checked.

To keep a clean design, we subclass or—where subclassing is not possible—wrap the concerned classes, i.e., java.io.FileOutputStream, java.io.FileInputStream, and javax.comm.CommPortIdentifier, and make all resource-specific changes in these subclasses or wrappers. We are aware that such a solution is not secure, because programmers can simply avoid using these classes. However, we set out to prove that resource control is possible using the standard Java security architecture. How to securely deploy the solution is left to future work.

Resource-accounting and policy-checking code has to be added when the resource is initially seized, e.g. in the constructor of the streams and the open()-method of the serial port, and when it is released and modified (e.g. bytes are written to or read from a file) (cf. Table 2, items 5 and 6). Unlike the policy-checking code, resource accounting must also consider the *failure* of a resource access. If e.g. no bytes were written due to a disk failure, this must also be reflected in the accounting data.

The required additions in the code are rather small and one path of future development is to find general ways of inserting resource control hooks without the need to subclass, e.g. through an event listener interface.

In order to enable our new permissions for the file system, we create a new security manager by subclassing the existing security manager and overwriting SecurityManager.checkWrite() and SecurityManager.checkRead(). The new security manager is extended by a new class field that is used for keeping accounting data.

A problem prior to the design of the class for accounting data is to decide which code—according to the code base entry of the policy file—is made responsible for resource use (cf. Table 2, item 3). It could be (1) the top caller or (2) the lowest caller that does not belong to a core Java class or (3) all callers. The Java security model would demand to make all callers accountable. But in that case, when it is time to check a permission, this permission does not remain static during the permission check but needs to be updated with accounting data for each protection domain. This is not possible with the standard access controller but would require a re-definition of the checkPermission()-method. In addition, the dynamic behaviour makes it difficult for an end user to understand why a resource permission fails because there is no way to provide an error message to the Access Controller—it only returns "Access denied" but no reason. Because of this, we make the *top protection domain* accountable (i.e. choice (1) above, cf. also Table 2, item 3) which also implements a semantics of trust between protection domains [12]. However, the top protection domain is not necessarily the protection domain that is on top of the call stack. When the resource access is done by privileged[2] code, or by code called by privileged code, then the privileged code is considered the top domain.

We store accounting data in fast and thread-safe hash tables. There is one hash table per accounted resource. The contents of the hash tables is specific per resource. However, the accountable protection domain must always be part of the accounting data. An example hash table that shows accounting data when limiting the size of files is shown in Table 3.

Table 3. Hash table with accounting data for writing files

Key	Value
Protection domain of the accountable code source + file name	Bytes written to the file
Example data	
Protection domain identifier 1: /tmp/a	2000
Protection domain identifier 2: /tmp/b	10
Protection domain identifier 2: /tmp/c	50000

We comprise all accounting data in one class, the ResourceManager. Thus, one can use the resource manager when information is needed about current resource use. By

[2] Just as the Unix operating system supports privileged code with the setuid feature, Java allows the marking of certain code within a class as privileged. This code ignores the permissions from its callers and only considers the permissions stated for this specific code and its called subroutines.

outputting the resource manager object, a snapshot overview over all resource usage is supplied.

3.3 Performance

In order to have a hint as to the performance penalty introduced by our resource management, we conducted a simple but deliberately stressful performance test by writing a 1MB file in chunks of 1, 10, 100, 500, and 1000 bytes. We tested the performance by letting the application perform under the Solaris utility *truss* that can be used to trace execution time. The test was run on a Sun Sparc work station running Solaris 8. The application was run with two versions of our resource-aware security manager, with the standard Java security manager and without a security manager.

Due to the fact that our resource management excessively invokes the access controller, namely three times per write due to three protection domains on the call stack, the performance penalty was high, especially in the time spent in user code. With a first version of the resource manager, there was even an invocation rate of five times per resource access. This was eliminated with a small change in the access controller that returns the top protection domain without invoking the access controller (but unfortunately makes the solution less portable). See Fig. 4 for details.

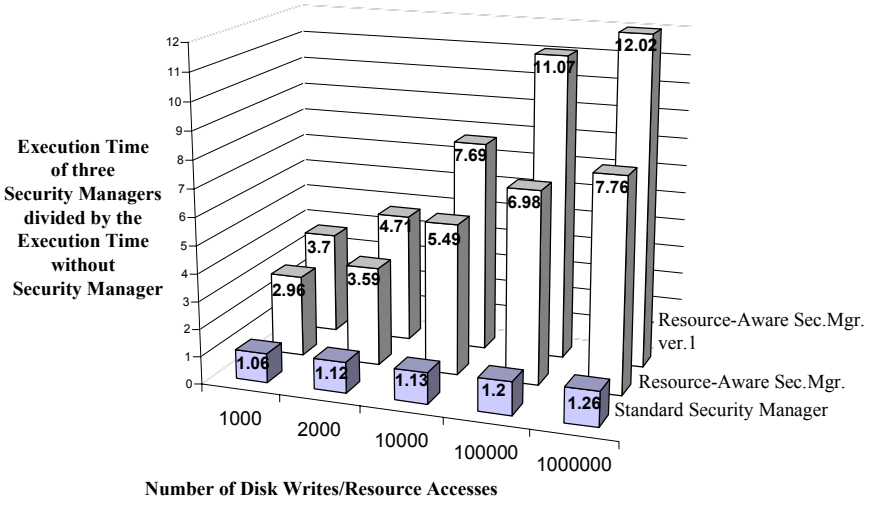

Fig. 4. Performance results when accessing the disk resource 1000, 2000, 10,000, 100,000 and 1,000,000 times. The more resource accesses, the slower perform the resource-aware security managers. In the worst case of 1 million resource accesses, the resource-aware security manager slowed the program execution down with a factor of 12. It is interesting to note that even the standard security manager is slightly affected by the number of resource accesses, although the access controller is invoked the same number of times (namely 41) in all five runs.

A conclusion drawn from the performance test is that the access controller must perform very fast if resource management with the help of the access controller shall be feasible. To unburden the access controller, additional intelligent assumptions about the outcome of an access control check could be implemented (e.g. if a small disk write is attempted within a certain time window after a successful disk write, the probability that the new write will succeed is high). However, such reasoning leads to an undeterministic behaviour of the access controller and could possibly be exploited by attacks.

We conclude this section by a critical view of our approach and give an outlook to one area of application. Our solution adds complexity to the already complex Java security architecture. Not even the standard security manager is much used for applications or servlets because often there is a trust relationship—e.g. through contracts—between code producer and platform owner. But also, because the policy management is considered cumbersome and error-prone. When there is a trust relationship, the "trusted" code is often given full permissions without much thought. Our solution may suffer from the same problem.

However, new platforms are emerging that run untrusted code, possibly without a trust relationship. In such cases, resource control is an important issue for guaranteeing the availability of the platform. An example for such new architectures are residential gateways or service gateways [13] that run services from multiple vendors as threads within one Java virtual machine with dynamic policies. Security problems in such a platform and their possible solutions have been dealt with in greater detail in [14].

4 Related Work

As mentioned in the introduction, much work has been done to bring resource-awareness to Java. The focus of previous work is rather on the goal of arriving at a resource-aware Java environment than on the integration of the solution in the existing architecture, which was our prime goal. Solutions are often used for real-time applications, i.e. for quality-of-service and not so much for availability as part of security. Also, research has been conducted in the area of formulating advanced policies for Java. We start with references to resource-aware Java virtual machine implementations.

JRes [15], a resource-accounting interface for Java, adds resource accounting and limiting facilities to Java as a class library, using bytecode rewriting. JRes introduces a resource manager class that co-exists with the Java class loader and the security manager. The resource manager contains native code for CPU and memory accounting. Overuse callbacks are generated and throw exceptions when resource limits are exceeded. The policy is hard-coded as Java code and resource limits apply to any code base.

J-Kernel [16] addresses the problem of untrusted servlets or Java plug-ins executing in one Java virtual machine, e.g. a web server. J-Kernel implements capabilities—access tokens—that represent handles to resources, i.e. a resource can only be accessed when the requester holds a corresponding capability. Unlike Java

object references, capabilities can be revoked at any time. J-Kernel is implemented as a Java library and sets out to prove that Java language features can be used to secure communicating pieces of Java code in a Java server. The access control policy is hard coded.

Chander, Mitchell and Shin [1] introduce bytecode instrumentation to remedy Java's lack of resource control. New, safe classes and methods will automatically—through the bytecode modifier—replace the original Java code prior to execution. The approach is as follows.

* Identify the Java classes that control the resources that need further monitoring.
* Subclass this class and add additional security checks and resource control to the subclass.
* Invoke the bytecode filter on the original class so that all references to the original class are replaced with references to the new, safer class.

Handling final classes is more complex than described above—it needs method-level bytecode instrumentation—but follows the same approach.

No examples of policy statements or a description of their syntax are provided. The specification language is said to be proprietary. It allows listing all classes and methods that need to be replaced. In addition, a graphical user interface is provided to edit resource limits such as number of windows, network connections or threads.

Much work deals with CPU and/or memory management. Real-time Java (www.rtj.org) allows resource control of the CPU through installable thread schedulers. Bernadat, Lambright and Travostino [8] have implemented a Java virtual machine that controls CPU and memory according to a policy in a policy file (with undocumented syntax). Another case of undocumented policy syntax comes from the Aroma Java virtual machine [9], which allows resource control for CPU, disk and network. SOMA [7] allows control over the thread scheduler and garbage collector and makes use of the Java VM Profiler Interface (java.sun.com/j2se/1.3/docs/guide/jvmpi/). Spout [17], originally designed for applet security, allows resource control for CPU, memory, threads and windows.

The specification of policies for Java is a related field to our work. Even before Java 2's security model with permissions in a policy files was introduced, researchers have been looking at ways of expressing advanced access control policies for the Java security manager. Nimisha, Mehta and Sollins [4] propose a constraint language that allows for three different kinds of permissions.

* Subject-based permissions base the access control on caller credentials—e.g. an applet's signature or code base.
* Object-based permissions consider the resource that shall be controlled—e.g. the total number of created windows.
* History permissions express e.g. that an applet is not allowed to connect to the network after it has read a private file.

This system was implemented in Java 1.0 by extending the security manager.

Java Secure Execution Framework (JSEF) [5] introduces a syntax for negative permissions—permissions that disallow access. JSEF also comprises support for hierarchical policy files that allow more than the Java 2 system policy file and user policy file. In addition, JSEF allows the negotiation of permissions at run-time. A piece of Java code could e.g. ask the user or a policy server at run time if a specific access should be completed.

Corradi, Montanari, Lupu et al. [6] propose a new language (Ponder) for Java permissions. Ponder supports negative permissions, support for subjects and objects, time restrictions and conditional expressions. Also the instant revocation of granted privileges is discussed.

5 Conclusion and Future Work

In this paper, we have shown that resource control of high-level resources is possible by using extensions of Java permissions and an unmodified Java access controller. Thus, resource control can be done within the existing Java sandbox. This contributes to a clear design of the security architecture. The Java permission syntax is so versatile that it can be extended to also support resource limits. However, the major work lies in finding all places in the code where the resource limit should be enforced or accounting data updated.

Still, the solution we present here is far from being deployed. In this paper, we only wanted to give a proof of concept. In order to work with it, ways must be found that force programmers to adhere to resource control and to use resource-aware classes. Ways of achieving this are modified Java virtual machines or automatic class modification at runtime (so-called bytecode modification).

The performance penalty in code that excessively accesses resources is mostly due to the fact that each resource access implies a number of invocations of the access controller—the number depends on the number of protection domains on the call stack. If the performance is to be improved, the number of calls to AccessController.checkPermission() needs to be reduced, for instance by intelligently avoiding to check permissions that have only a slightly changed resource limit.

Other performance bottlenecks can be synchronisation issues on the accounting data object. If different execution threads need access to a common accounting object, only one thread gets the access at one time; others have to wait. Also the used hash tables may be bottlenecks. Other structures, for instance containers that are similar to database tables, should be explored and tested for their performance.

So far we have only been dealing with the length of time a resource can be accessed. Another dimension is the number of times a resource can be accessed, created, modified or deleted. Further development includes the expansion of this technique for other high-level resources such as windows, microphone, speaker, etc. More thorough performance tests using recognised benchmarks should be done.

However, one of the more important future tracks is an investigation of what new vulnerabilities are introduced by such high-level resource management.

References

[1] Marco Pistoia, Duane F. Reller, Deepak Gupta et al. *Java™ 2 Network Security*. 2nd Edition. Prentice-Hall. 1999.

[2] Gary McGraw, Edward Felten. *Securing Java—Getting Down to Business with Mobile Code*. Wiley & Sons. 1999.

[3] Ajay Chander, John C. Mitchell, Insik Shin. *Mobile Code Security by Java Bytecode Instrumentation*. Proceedings of the 2001 DARPA Information Survivability Conference & Exposition II, DISCEX. Vol.2. pp. 27-40. IEEE. 2001.
[4] Nimisha V. Mehta, Karen R. Sollins. *Expanding and Extending the Security Features of Java*. Proceedings of the 7th USENIX Security Symposium. January, 1998.
[5] Manfred Hauswirth, Clemens Kerer, Roman Kurmanowytsch. *A Secure Execution Framework for Java*. Proceedings of the 7th conference on computer and communications security. pp. 43-52. ACM. November, 2000.
[6] Antonio Corradi, Rebecca Montanari, Emil Lupu, Morris Sloman, Cesare Stefanelli. *A Flexible Access Control Service for Java Mobile Code*. Proceedings of the 16th Annual Conference on Computer Security Applications (ACSAC). pp. 356-365. IEEE. 2000.
[7] Paolo Bellavista, Antonio Corradi, Cesare Stefanelli. *How to monitor and control resource usage in mobile agent systems.* Proceedings of the 3rd International Symposium on Distributed Objects and Applications, DOA'01. pp. 65–75. IEEE. 2001.
[8] Philippe Bernadat, Dan Lambright, Franco Travostino. *Towards a Resource-safe Java for Service Guarantees in Uncooperative Environments*. In Proceedings of the IEEE Workshop on Programming Languages for Real-Time Industrial Applications. Madrid. IEEE. Dec. 1998.
[9] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, et al. *State Capture and Resource Control for Java: The Design and Implementation of the Aroma Virtual Machine*. Proceedings of the Java™ Virtual Machine Research and Technology Symposium. Usenix. April, 2001.
[10] Li Gong, *Inside Java™ 2 Platform Security*. The Java Series. Addison-Wesley. 1999.
[11] Scott Oaks. *Java Security*. 2nd Edition. O'Reilly. 2001.
[12] Dirk Balfanz, Drew Dean, Mike Spreitzer. A Security Infrastructure for Distributed Java Applications. Proceedings of the IEEE Symposium on Security and Privacy (S&P), 2000. Pages: 15-26. May, 2000.
[13] Kirk Chen, Li Gong. *Programming Open Service Gateways with Java Embedded Server™ Technology*. The Java Series. Addison-Wesley. 2001.
[14] Almut Herzog. *Secure Execution Environment for Java Electronic Services* (tentative). Licentiate thesis. Dept. of Computer and Information Science. Linköping university. To be presented, fall 2002.
[15] Grzegorz Czajkowski, Thomas von Eicken. *JRes: A Resource Accounting Interface for Java*. Proceedings of OOPSLA'98. Pages 21-35. ACM. 1998.
[16] Chris Hawblitzel, Chi-Chao Chang, Grzegorz Czajkowski, Deyu Hu, Thorsten von Eicken. *Implementing Multiple Protection Domains in Java*. Proceedings of the USENIX 1998 Annual Technical Conference. pp. 259-272. Usenix. 1998.
[17] Tzi-cker Chiueh, Harish Sankaran, Anindya Neogi. *Spout: A Transparent Distributed Execution Engine for Java Applets*. Proceeding of the International Conference on Distributed Computing Systems (ICDCS). pp. 394-401. IEEE. April, 2000.

Performance Evaluation of Cryptographic Algorithms for Multicast Secrecy Protection

Josep Pegueroles Vallés, Francisco Rico-Novella

Telematics Engineering Dept. Polytechnic University of Catalonia (UPC).
c/ Jordi Girona 1-3 CAMPUS NORD 08034 Barcelona. Spain
{josep.pegueroles, f.rico}@entel.upc.es

Abstract. Security has become a significant requirement in nowadays multimedia communications. Once bandwidth constraints are being overcome, secrecy arises as a new desirable property for group communications. Multimedia group communications are very time restrictive so an unreliable transport protocol such as User Datagram Protocol (UDP) is used. Furthermore, due to latency and synchronism considerations for multicast rekeying, block-cipher instead of stream-cipher algorithms are used. This scenario forces us to limit the size of plain-text data to be ciphered to the maximum transfer unit (MTU) of the transport protocol. In any other case, a packet loss will lead us to spreading errors. On top of that, group communications usually take place between heterogeneous agents. So cross-platform would be a desirable feature for group communication applications. This paper presents the characterization and performance evaluation of the most important ciphering algorithms recommended in IPSec to provide data confidentiality. This study takes account of the behavior of JAVA source code over different platforms in order to choose the better algorithm for ciphering data in such environments.

1. Introduction

When cryptographic techniques are added to advanced video services such as video-conferencing many aspects have to be considered. First of all multimedia requirements are very time-restrictive, this is why unreliable protocols such as UDP are used, so ciphering techniques cannot rely on transport protocol and they may be fitted to isolated protocol data units (PDU), in any other case, a packet loss will cause forward errors. Secondly, ciphering algorithms may not increase significantly the packet delay. If this occurs, security increase will lead to visual quality decrease or excessive transmission delay, and interactive services will not be possible.

The main purpose in this study is the characterization of the behavior of different symmetric block-cipher algorithms in order to add confidentiality to multicast multimedia communications.

A simple video server model was implemented. It uses JAVA programmed ciphering and communications modules. This choice was due to the platform-independent feature of the mentioned language that makes it especially suitable for heterogeneous agents' communication.

Java offers a wide range of cryptographic tools. Among these facilities are Cryptographic Service Providers. These refer to a package or a set of packages that supply a concrete implementation of a subset of the cryptography aspects of the Java security API. So application programmers do not need to deal with the algorithms details but only call a generic instance that does it for them. Differences in the code implementation of these providers can be critical in time restrictive applications. Measures in order to choose properly the algorithm, the provider, and the plain text block size were taken.

Particularly, we measured how long it takes to cipher a block, the ciphering speed and dependency on type of processor. We also took measures of Loss and Rejection Probability due to network or excessive delay reasons. All these parameters were considered for different block sizes, providers and algorithms.

The rest of the paper is organized as follows: Section 2 gives the overview of our video-server model. The usage of multicast communication and its security requirements are discussed in Section 3. Section 4 introduces Java tools for security, especially cryptographic service providers' properties and possibilities. A description of the developed modules is given in section 5. We examine the obtained results in section 6 and finally we present a summary in section 7.

2. Video Server Model for Multicast Communication

Typically, a multimedia server works as sketched in Fig. 1.

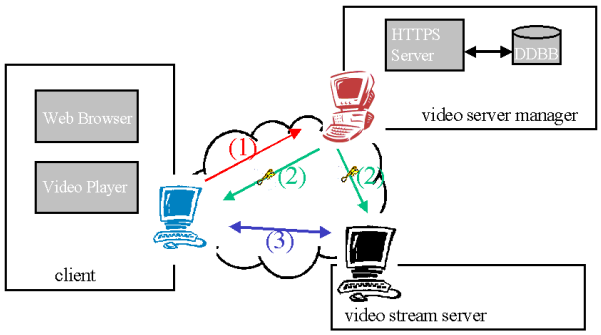

Fig. 1. Communication steps in video server architecture

When a client wants to access multimedia content in a near-on-demand video server, it accesses the server Menu via a web browser (step1 in Fig. 1) where available films and show hours are presented. After that, the client can select his own preferences (film or content to view, timetable…). Once the server knows exactly what the client wants, it asks for a payment. When this is done, the server returns to the client the data he will need in order to get the multimedia stream (step 2 in Fig. 1). It gives to the client the multicast address where the data will be delivered and the needed key in order to decipher the communication. It also sends the key to the server

host (step 2 in Fig. 1). Then the client is able to join the multicast group to get the purchased content (step 3 in Fig. 1) [1].

2.1 Vulnerabilities of the system

When using open networks, security troubles become important. In the mentioned model a very weak point is showed up: communication via Internet [2].

First, the Security Module must protect the information that the client sends to the server to order the content. Second, it must warrant that the payment will be done properly and that no participant in the transaction will be harmed. Finally, multimedia data have to be protected from malicious third parties or eavesdroppers.

According to the former paragraph, the security services required are secure access and secure payments via web, both over reliable protocols, and data secrecy over unreliable protocols.

Since these three steps are done over different transport protocols we will divide the security services in our system into two phases: secure services over reliable protocols: that is, authentication and payment; and secure services over unreliable transport protocols, that is to say, secrecy.

Next sections will discuss the assumptions we took in order to implement the security services over the unreliable phase.

3. Multicast Ciphering

As we mentioned before, data should be protected from unauthorized third parties. This is easily achieved by means of ciphering. The inclusion of ciphering algorithms in multimedia communications is critical in terms of time. Cryptographic techniques must not add significant delay to data transmission or excessive packet loss due to time constraint violation will occur.

Besides that, when multicast technology is used, additional considerations should be taken into account. When adding confidentiality to multicast communications a common secret shared by all the multicast group members is needed. The shared key provides group secrecy and source authentication. This key must be updated every time membership in the group changes, if so Forward and Backward Secrecy (FS and BS) are provided. FS means that the session key gives no meaningful information about future session keys, that is to say no leaving member can obtain information about future group communication. BS means that a session key provides no meaningful information about past session keys or that no joining member can obtain information about past group communication.

This is why video files are encrypted on-line before sending. Different keys are used for the same file, even during one session. So stored ciphered files have no sense in this scenario.

On the other hand, the Internet Engineering Task Force (IETF) defines how secure group communications should be in the Group Domain of Interpretation (GDOI) [3]. This document only specifies data-security for one security protocol, IPSec ESP. The current specification dictates that a compliant implementation must support DES in

cipher block chaining mode. A number of other algorithms have been assigned identifiers in the DOI document and could therefore easily be used for encryption; these include the following [4][5][1]:

- Three-key triple DES
- RC5
- IDEA
- Three-key triple IDEA
- CAST
- Blowfish

Finally, it is important to the client application to be as much universal as possible. This is why the client side was implemented in Java language. This allows the client to be loaded via web by using the applet technology. This choice greatly simplifies the development of the client side since Java has a wide variety of cryptographic tools.

So, the study and right election of the Java cryptographic tools of the client side for secure multicast video on demand is an important item if high quality multimedia servers want to be achieved [6].

4. Java security tools

Since its inception, Java was conceived as built for the net. So, unlike other programming languages, security mechanisms have always been an integral part of Java.

The Security API is a core API of the Java programming language, built around the java.security package. It is designed to allow developers to easily add security features to their programs. JDK 1.1 introduced the "Java Cryptography Architecture" (JCA) as a framework for accessing and developing cryptographic functionality for the Java platform. It included APIs for digital signatures and message digests. Java 2 SDK significantly extended the former security features of JDK 1.1 and introduced a new Java Security Architecture for fine-grained, highly configurable, flexible, and extensible access control [7].

The Java Cryptography Architecture (JCA) was designed around two basic principles: implementation independence and interoperability algorithm independence and extensibility [8].

Implementation independence and algorithm independence are complementary; you can use cryptographic services, such as digital signatures and message digests, without worrying about the implementation details or even the algorithms that form the basis for these concepts. When complete algorithm-independence is not possible, the JCA provides standardized, algorithm-specific APIs. When implementation-

[1] Although latest versions of SunJCE and Wedgetail offer AES algorithm for ciphering, the available versions when the study was done did not include it. Besides that, the latest Internet draft for the use of the AES Cipher Algorithm in IPSec (ESP) was published on June 2002 <draft-ietf-ipsec-ciph-aes-cbc-04.txt>, and is still a work in progress.

independence is not desirable, the JCA lets developers indicate a specific implementation.

Algorithm independence is achieved by defining types of cryptographic "engines" (services), and defining classes that provide the functionality of these cryptographic engines. These are called engine classes.

Implementation independence is achieved using a "provider"-based architecture. The term Cryptographic Service Provider (or simply provider) refers to a package or set of packages that implement one or more cryptographic services, such as digital signature algorithms, message digest algorithms, and key conversion services. A program may simply request a particular type of object implementing a particular service and get an implementation from one of the installed providers. If desired, a program may instead request an implementation from a specific provider. Providers may be updated transparently to the application, for example when faster or more secure versions are available.

Implementation interoperability means that various implementations can work with each other, use each other's keys, or verify each other's signatures. This would mean, for example, that for the same algorithms, a key generated by one provider would be usable by another, and a signature generated by one provider would be verifiable by another.

Algorithm extensibility means that new algorithms that fit in one of the supported engine classes can be added easily.

JCA includes the Java 2 SDK Security API related to cryptography. It has a "provider" architecture that allows for multiple and interoperable cryptography implementations.

The Java Cryptography Extension (JCE) extends the JCA API to include APIs for encryption, key exchange, and Message Authentication Code (MAC). JCE was previously an optional package (extension) to the Java 2 SDK, Standard Edition, versions 1.2.x and 1.3.x. JCE has now been integrated into the Java 2 SDK, v 1.4.

The multimedia server developed in this project takes advantage of the encryption utilities of JCE and uses the “provider”-architecture in order to have algorithm and implementation independence. The right election of the algorithm and provider for minimum packet loss is one of the goals of this study.

4.1 Providers

As mentioned before, JCA introduced the notion of a Cryptographic Service Provider as a package that supplies a concrete implementation of a subset of the cryptography aspects of the Security API [9].

The providers used in this project contain an implementation of one or more ciphering and key generation algorithms.

As previously noted, our multimedia server application may simply request a particular type of object (cipher engine) for a particular service (such as encryption) and get an implementation from one of the installed providers. The program can request the objects from a specific provider as each provider has a unique name used to refer to it.

Next we will introduce the tested providers we have studied in the current work and the block ciphers that provides each one.

ABA/OpenJCE
ABA stands for Australian Business Access, the name of the company that starts the project. In 1999 it was renamed to eSec Ltd [10], and the provider becomes the open source OpenJCE Project. It is a clean room implementation of the JCE API as defined by Sun, plus a provider of underlying crypto algorithms. The source code of the Provider can be found in [11].

Currently the project is defunct and no maintenance is provided [12].

DSTC
DSTC means Distributed Systems Technology Center and it is the name of the company that starts the JCSI (Java Crypto and Security Implementation) provider.

JCSI Provider 2.2 is a provider for JCA/JCE implementing industry standard algorithms for public and symmetric key cryptography. It is a "signed JCE provider", meaning it can be used with the standard JCE 1.2.1 framework and with JDK 1.4. Although the project was initiated by DSTC it is currently maintained by Wedgetail communications. It is not open source but you can ask for an evaluation copy at [13].

SunJCE
SunJCE is the provider that is included and automatically registered in the java.security security properties file included with the Java 2 SDK, v 1.4 [14].

From all the features that these providers include, this work has centered its efforts in studying the performance evaluation of symmetric ciphers. In Table 1 are represented the studied cipher engines included in each package.

Table 1. Ciphering algorithms for different providers.

	DES	DESede	Blowfish	IDEA	RC4
ABA	✓	✓	✓	✓	✓
DSTC	✓	✓	✓	✓	✓
SunJCE	✓	✓	✓	×	×

5. Developed Modules

In order to take the desired measures for real implementation, a simple video client-server framework was developed. The experimental platform was Unicast in nature. In spite of that, all the obtained results referring to ciphering time and speed can also be applied to multicast ciphering. Results related with loss probability, although not directly applicable to multicast can easily be extrapolated.

In Fig 2 it is shown the architectural design.

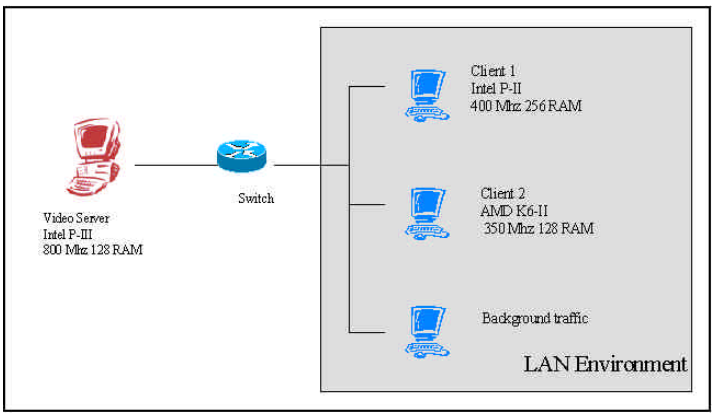

Fig. 2. Developed video server framework

Three different processor types were used for measures:

- AMD K6-II 350 Mhz 128 RAM
- Intel P-II 400 Mhz 256 RAM
- Intel P-III 800 Mhz 128 RAM

All of them acted as client and servers in order to measure ciphering time for different algorithms and providers. When taking measures for real traffic, an Intel PIII 800Mhz were used as server, since servers normally need much more processing capacity.

Real background traffic was considered by connecting a public web and ftp server in the same LAN segment. All measures were taken at the same hour and similar background traffic levels. No variations of background traffic level were considered to force packet loss. Incidence of different background traffic levels is beyond the purpose of this study and we point it as future work.

Clients were all connected to a shared 100 Mb Ethernet LAN segment. They where connected to the video server through a 100 Mb switch.

All the video sequences used in this work were VBR coded with maximum peak rates of 4Mbps. The peak rate settles a notion of the desired ciphering speed.

Fig 3. and Fig. 4 show the block diagram of the server and client Java programmed modules.

As we have mentioned above, the cipher speed and cipher time measures were taken using a simple Unicast client-server model. The server application (called UDPServer) first establishes an UDP Socket from where it waits for the client film request. When it receives the request with the corresponding parameters (java provider and algorithm to use and packet size) the server begins to read the data from the video file. After that it ciphers the video fragment using the chosen provider and algorithm, insert a sequence number in order to control packet loss, constructs the UDP packet and sends it. Marks for ciphering time traces are introduced in the cipher payload module.

On the client's side the behavior is as follows. After invoking the application with the proper parameters, the client sends a request message to the server. Then it waits

for the video stream listening to the UDP socket connection. The client application mainly receives the UDP packets, checks whether they have the desired sequence number or if they have excessive delay for accepting and processing them or rejecting. With this information, the loss probability can be calculated.

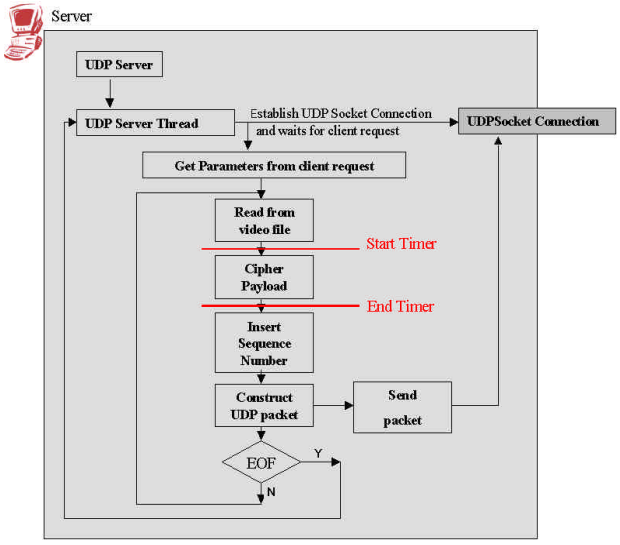

Fig. 3. Block diagram of server module.

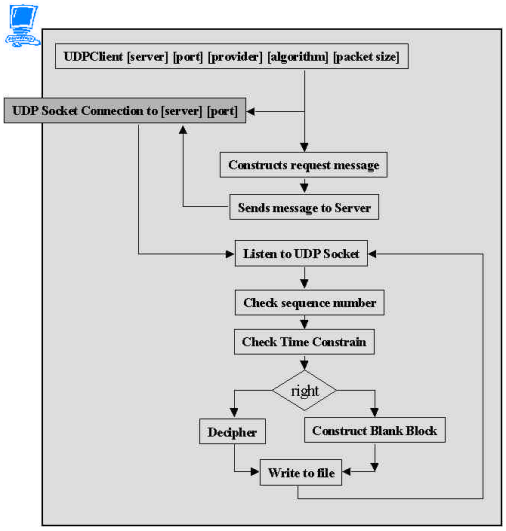

Fig. 4. Block diagram of client module.

6. Results

6.1 Different plain text block size

Measures showing the dependency on plain text block size were taken. Although mean cipher speed was the same for different packet sizes fluctuations in the cipher speed due to processor scheduling were observed. Fig 5 shows cipher speed for 300 video packets. Packets greater than 32K exhibits more stability in cipher speed than smaller ones.

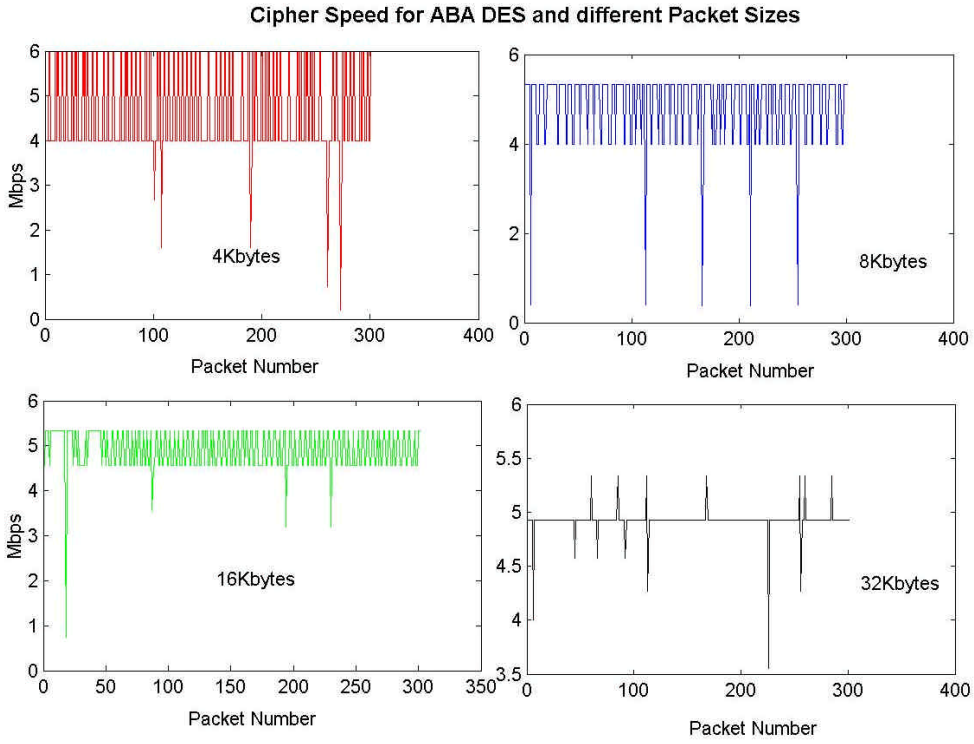

Fig. 5. Behavior for different plain text block size

This behavior can be easily explained if we consider that no process can use the processor while another process is executing an atomic instruction. If we consider the ciphering instruction as atomic, additional processes in the same computer as our server can only access the processor after a ciphering instruction. The smaller the packet size is, the more the probability that a process asks for the processor is. So greater size packets will lead us to stable ciphering time patterns. In the other hand, we must limit the plain text block size to MTU in UDP packets, if not a UDP packet

loss will affect the packets in which the plain text block size are split, cause receiver will not have the enough information in order to decipher the message.

6.2 Different algorithms and providers

Maybe the most important parameter to study was what provider and algorithm were the most suitable for our application. The ciphering speed was calculated from the ciphering time following expr. (1)

$$speed[Mbps] = \frac{\text{Packet size}[Kbytes]}{\text{Ciphering time}\,[ms]} \qquad (1)$$

As different providers can have different code, first we tested the ciphering time for the three studied providers (Fig 6 and Table 2). In almost all the tested cases ABA showed the best performance.

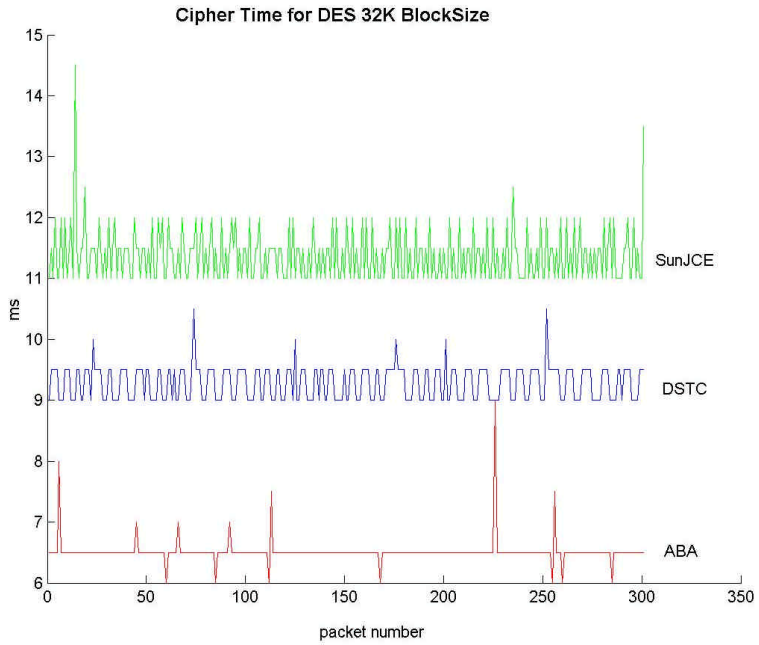

Fig. 6. Cipher Time for different providers

Respecting the cipher speed for different algorithms, DES had the best behavior.

Table 2 and Fig. 7 show the speed for different algorithms in ABA provider. It is important to note some abnormal results. First RC4 it is not the fastest, as for a stream cipher is expected. Second, Blowfish, conceived as an algorithm to be faster than DES does not reach this feature. Finally, DESede is not three times slower than DES.

It is very difficult to analyze the actual reasons that can cause these misbehaviors. Source code for different providers is not public so details of how a particular

algorithm is programmed and what optimization techniques were used cannot be studied. However, we can assume that the RC4 behavior is due to the misuse that we make of it, since we actually are ciphering blocks (UDP packets) using a stream cipher.

Respecting to DES, it is plausible that the best performance behavior was due to its popularity. DES is one of the most used ciphering algorithms and because of it, many optimization techniques at the programming level have been developed in order to achieve better performances.

Table 2. Speed for different algorithms.

Mbps	**ABA**	**DSTC**	**SunJCE**
DES	4.917	3.4487	2.8168
DESede	1.9117	1.5802	1.0516
Blowfish	3.2986	2.4321	4.9688
IDEA	3.2871	2.7357	-
RC4	3.9104	3.4164	-

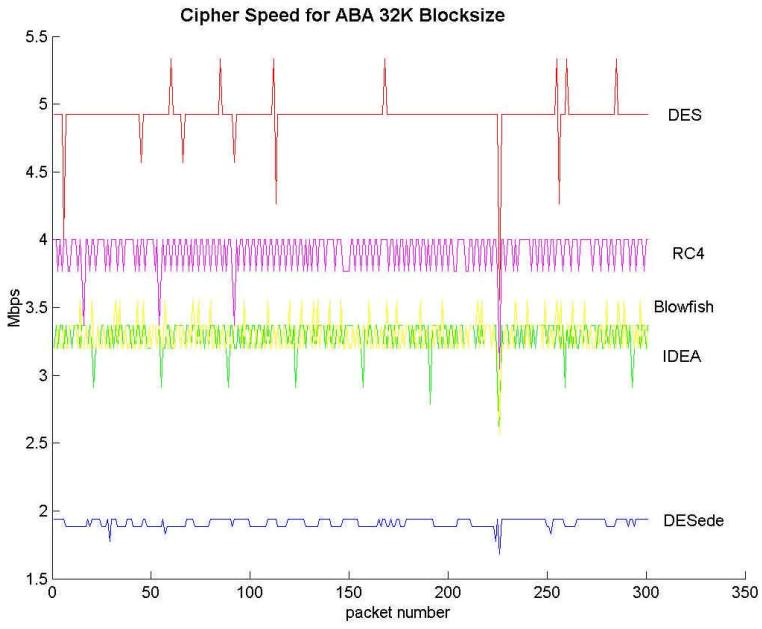

Fig. 7. Cipher Speed for different algorithms

6.3 Different processors

In order to test behavior of different processor types expr (2) was calculated. Results show that Intel Pentium family processors have the best performance. See Table 3.

$$ef[Mbps/Mhz] = \frac{\text{Ciphering speed}[Kbytes]}{\text{Processor speed}[ms]} \qquad (2)$$

Table 3. Efficiency parameter for different processor types

[Speed/Mhz]·10^{-2}	**PIII 800Mhz**	**PII 400Mhz**	**AMD K6 350 Mhz**
DES	0.6146	0.6195	0.4020
DESede	0.2389	0.2437	0.1584
Blowfish	0.4123	0.4166	0.2415
IDEA	0.4108	0.4158	0.3268
RC4	0.4888	0.5020	0.3759

6.4 Loss Probability

Finally, loss probability was measured for different providers, algorithms and packet sizes.

Figures were obtained by the comparison of the original video file and the received one. A packet loss is represented in the received file by means of a blank block, so video files do not differ in size but in content.

We did not find any regular pattern in the behavior of the Loss Probability depending on the used provider nor algorithm. Note that in Fig8 SunJCE has the best behavior for some plain text block size but it also has the worst behavior for other sizes.

Results respecting variations in client buffer size were not the purpose of this study but it can be taken into account in future works.

All the obtained results are less than 0.5% and they are not significant in relation with other loss probability causes. Results show that choosing 32K DES ABA parameters as our video server configuration does not affect Loss Probability.

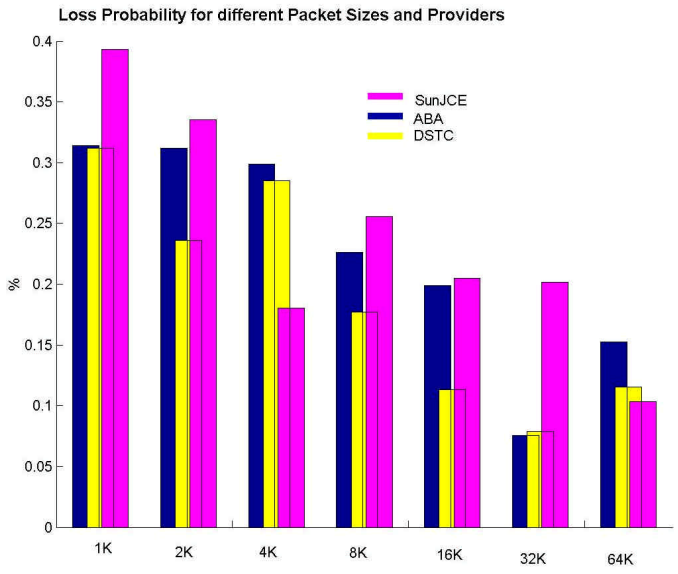

Fig. 8. Loss Probability for different packet sizes and providers using DES algorithm.

7. Conclusions

In this work we have described the design and implementation of a secure communication infrastructure for multimedia multicast applications. This infrastructure use Java tools to provide confidentiality to multimedia data streams. The election of Java was due to the inherent secure nature of this programming language and the wide variety of cryptographic facilities aiding the development of secure applications. Usage of Java language adds a new design parameter to consider: the Cryptographic Provider. Different providers show different performance behaviors so the study for the right election of the provider was also considered.

Few studies have been done in order to characterize the speed and behavior of these tools. No previous works comparing theoretical results to real implementations are known.

This characterization is critical in real-time environments such as multimedia multicast communications. Time spent in ciphering UDP packets is an important parameter when electing a certain algorithm or tool. Particularly, cipher block time, ciphering speed and dependency on processor type was measured. Loss and Rejection Probability due to network or excessive delay reasons were also measured. The measures were taken in our implementation for three different Providers: ABA, DSTC and SunJCE, different block sizes and different processors.

Results lead us to choose ABA provider for DES CBC ciphers in 32Kbytes plain text block size. This election fits with IPSec recommendation and achieves a throughput of near 5Mbps, enough for our video streams.

Peculiarities observed in the behavior due to processor scheduling or type of processor have also been discussed.

Acknowledgments

This work has been supported by the Spanish Research Council under project SSADE [CYCIT TEL 99-0822]

References

1. Szuprowicz, Bohdan O. Multimedia Networking. Harness the power of new multimedia networking technologies. Mc Graw-Hill Inc. 1995
2. Schneier, B. Secrets & Lies. Digital Security in a Networked World. John Wiley & Sons Inc. 2000.
3. Baugher M., Hardjono T., Harney H., Weis B. The Group Domain of Interpretation. I-D Draft-ietf-msec-gdoi-04.txt Feb 2002. Work in progress.
4. Stallings, W. Network Security Essentials. Applications and Standards. Prentice-Hall Inc. 2000.
5. Kent S. IP Encapsulating Security Payload (ESP). IETF RFC 2046. Nov 1998
6. Arasa, Jose L. Estudi de la integració de confidencialitat en servidors de vídeo. Aplicació a tràfic real. Master Thesis for the Telematics Engineering degree. Supervisor: Josep Pegueroles 2002.
7. Pistoia M., Reller D.F (et al) (1999), Java 2 Network Security 2nd Edition. Prentice Hall Inc.
8. Gong, Li (1998) Java Security Architecture (JDK1.2) version 1.0. Sun Microsystems Inc.
9. http://java.sun.con/products/jce/jce122_providers.html Official Java Providers web page
10. http://www.esec.com.au. ABA provider web page.
11. http://online.securityfocus.com/tools/category/21 OpenJCE download page.
12. http://www.openjce.org. OpenJCE provider defunct web page
13. http://www.wedgetail.com/jcsi/2.2/provider/index.html
14. http://Java.sun.com/products/jce/index-121.html. SunJCE provider web page.

Computational Power Borrowing Model for Mobile Security Computations

Wael Adi[1] IEEE member, Ali Mabrouk[2]

[1] Etisalat College of Engineering, United Arab Emirates.
e-mail: wael@ece.ac.ae
[2] Lufthansa Systems AS, Germany.
ali.mabrouk@lhsystems.com

Abstract. Nowadays there is an ever-growing demand in the world of telecommunication, that highly complex arithmetic operations are to be performed online. The recent technological achievements in communication systems allow collaboration between mobile equipment and fixed network entities. Such collaboration could be very helpful if we consider the fact that mobile terminals possess mostly low computation power, which leads often to serious processing bottlenecks in mobile security functions. Flexible allocation of computational power for security processing is becoming increasingly essential to achieve affordable modern security schemes. Such schemes require increasingly tremendous computational power. Computational power borrowing mechanisms for mobile security computations are needed to cope with the ever growing demand on security for mobile applications in highly complex and heterogeneous IT environment having different processing power and storage constraints. Wireless LANs seem to be a promising technology providing new features to the world of telecommunications, which could be adapted to achieve the previously stated demand. Wireless LAN often connect a mobile user to a local area network (LAN). Some wireless LANs are implemented at strategic points (so called traffic hot spots) like airports, railway stations and hospitals to provide access to nearby "roaming" users. However migrating security relevant computations to any collaborating host may lead to the loss of secrecy or authentication. Thus it is essential to involve secret hiding arrangements during such migrations to avoid any security threats on sensitive information. We propose a so called "fuzzy computation schemes" which allow powerful network nodes to perform the most complex security computations on behalf of weak terminals with minor security loss. In a previous paper[1], we outlined a fundamental technique known as "fuzzy modular arithmetic". Possible use in well-known public-key cryptographic scenarios[2] for mobile applications was also introduced. The idea is generally applicable to all systems with entities of highly different computational capabilities. In this paper, we introduce a general computational power borrowing model that is based on the "Fuzzy Modular Arithmetic" and

---

1 Wael Adi, "Fuzzy Modular Arithmetic for Cryptographic Schemes with Applications for Mobile Security". IEEE/AFCEA EUROCOMM 2000

2 Wael Adi, Ali Mabrouk; "Fuzzy Public-Key Exchange Mechanism with Applications to Mobile Security" I2TS'2002, (International Information Technology Symposium Florianopolis, SC, Brasil October , 2002

mainly adapted to the mobile environment.

Index Terms—fuzzy modular arithmetic, encrypted functions, mobile agents, computational power on demand, wireless LAN.

A Holistic Information Security Management Framework

applied for electronic and mobile commerce

Albin Zuccato

Karlstad University, Universitetsgatan 1, 65 866 Karlstad

keywords: security management, information security management, systemic-holistic approach, security management workflow

Abstract. Due to shortcomings in state-of-the-art security management methodologies and special requirements for e/mCommerce, the author has - during his working period in a large Austrian bank - discovered a need for a comprehensive security management framework. Due to that need a systemic-holistic framework for security management has been developed which will be presented in this paper.
In order to create a framework, at first the generic model is going to be presented where also the framework´s components will be described briefly. The analysis will be backed up by real work experiences.

1 Introduction

Today's society becomes more and more dependent on information technology. A computer break down mostly means loss of money. For companies this raises the necessity to prevent such a situation. Given the tendency of eCommerce, where the companies business activities only can be conducted by information technology means, this matter becomes even more important.

It seems that the tendency described also raises the interest for information system security. However, most companies who are willing to engage in security, face standards or methodologies of high complexity which are sometimes heavily technology centric and do not reflect their business needs. This discourages a lot of companies and reduces their motivation to invest in security.

To counteract the complexity, this paper proposes a "Holistic Security Management Framework" (HSMF) for electronic and mobile commerce which especially considers business, technique and sociology (see figure 1). The framework is based on a systemic-holistic approach of information security as proposed by Louise Yngström [Yngström, 1996] and will therefore be based on theoretic concepts enforced by General System Theory [von Bertalanffy, 1969].

To be able to construct a framework that can be applied successfully in business life, a set of requirements supported by the framework will be defined. Based on these requirements a generic outline of the framework will be presented. Then, according to system theory, the environments, components and interactions of the framework are defined. Finally, experiences of applying this

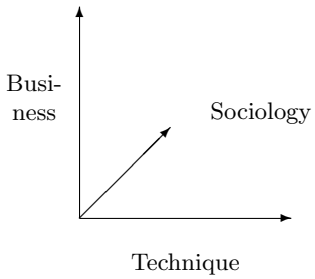

Fig. 1. Dimensions for holistic security management

framework will be described. Additionally it will be analyzed in how far the requirements previously defined, are realized in the framework.

2 Requirements

If we look to software engineering literature (see for example [Boehm, 1986], [Jacobson et al., 1999]), we can see that every development is preceded by the formulation of requirements. However, to develop a management framework is not entirely comparable with software engineering. A requirement definition, though, gives us a sound base to build on and it makes it possible to evaluate the result later on.

The framework has already been, and will also in the future be, further developed in an iterative way – by means of action research – and the requirements will be improved and completed in the same way. In the beginning, a first requirement draft was developed by brainstorming in the security group in a large Austrian bank. Also the first refinement step – a consideration of characteristics that should be supported – was defined. Before that, the requirements had been refined but stayed stable.

The current status of generic requirements that the framework has to fulfill, is that they must be:

1. Fast due to short time-to-market cycles in the e/mCommerce environment
2. Business centric to motivate security by business means
3. People centric to mitigate resistance and enhance awareness
4. Technology-oriented, but not driven, to support state-of-science security
5. Supportive in relation to the conventional software development process.

With these requirements as a base, a generic list of characteristics that the framework supports, was developed:

– Due to the fact that m/eCommerce requires short time-to-market cycles, such methods, that promise a good time behavior even for the cost of less accuracy, need to be selected with higher priority.
– A focus on existing business methods is necessary to make the framework fit better to the existing organization. In other words, you have to avoid the use of proprietary technology-oriented methods.
– Security has to be justified by economic means. However, a use of only risk analysis cannot be sufficient as it exclusively reflects an insurance perspective. To use business modelling instead helps finding security requirements that act as business enablers[1].
– People are a crucial part of security management and need support in their roles as users, developers and decision makers.
– Privacy is an important and recommended feature but should be optional in the framework due to different requirements in B2C and B2B eCommerce[2].
– Security is not an add-on approach but needs to be conducted together with the system development.
– An iterative approach has to be built on a life-cycle model that supports the current software engineering trends.
– Security must be observed during a system´s whole lifetime.

3 The Framework

Based on the requirements and the scientific concepts of general system theory enhanced by the systemic-holistic findings of Yngström [Yngström, 1996], a generic security management framework for e/mCommerce was developed. Scientifically the model is as complete as it possibly could be, but in the same time it is also applicable with justifiable resources. Special attendance has been payed to the balance between practicability concerns and state-of-the-science knowledge.

We have seen a lot of approaches that, according to the criteria developed in [Siponen, 2001], are acceptable from a scientific perspective but much too complicated or outdated. So for example Automated Secure System Development (ASSDM) by [Booysen and Eloff, 1995] or Life Cycle Model in Organizations (LCMO) by [Badenhorst and Eloff, 1989] which follow old Software Engineering methodologies[3] which are not up-to-date with the development of Object Oriented systems – which eCommerce systems depend on. Other approaches like Virtual Methodology (VM) by [Hitchings, 1995] or Structures of Responsibility by [Backhouse and Dhilon, 1996], which focus on social relationships to determine security needs, and in that way neglect economy and technic as driving factors for security. This means that this approaches do not reflect the mutlidimensional requirement structure which emerges with eCommerce. For us this

[1] A business enabler is a crucial function. Without its existence business would be impossible.
[2] B2C means Business-to-Consumer, B2B mean Business-to-Business
[3] ASSDM follows the spiral model and LCMO follows a waterfall approach

means that current scientific methods are either base on data that are not – or only at an unaffordable price – available in business, or neglect the fact that "people" have to implement and work with the system or simple do not cope with business and/or technology as driving factors for eCommerce systems. On the other hand, there are also "practical models" that are incomplete as they do not generate the expected effects and benefits.

When the first theoretical framework had been built (at that time by considering the environmental factors in eBanking), practical improvements were carried through in an iterative way. By the use of action research methods, 3 iterations of applying and improving the framework were conducted. During the application cycles some deficits were realized and some minor corrections were required and undertaken at runtime. In the analysis phases, between the iterations, the framework was consolidated and improved. As an outcome the generic model was generated – see figure 2.

In the following part we will briefly describe every component in the model. With this in mind, we will follow the system theoretic structure of models that is built out of components, interactions of components and environment(s) [Schoderbek et al., 1985].

3.1 Components

Components or objects are the elements of a system. Together they form the system. From a functional perspective they represent the basic functions of a system and are mostly grouped into input, output and process components [Schoderbek et al., 1985].

In the framework the workflow and the support components are of process type. The input and output of the processes are due to space restrictions not listed in this paper. However, because of their special role the environmental input components are listed here too.

Workflow components

These components form the main foundation for the implementation and assurance of security. Together they constitute the core workflow of the system as described below. The goal of the following description is to briefly show what they include.

Business modelling First of all the security enhanced business model has to be developed. In this part the company states its business intends, and how they can be fulfilled from a strategic and tactic viewpoint.
Here "security enhanced" means that security is introduced into the business model at the very beginning. This is the building block by which future activities are legitimated.

Project Planning In this step, plans are made how to incorporate security into the operative business. This task brings with it that a lot of daily business needs have to be considered. In difference to the business model – where an

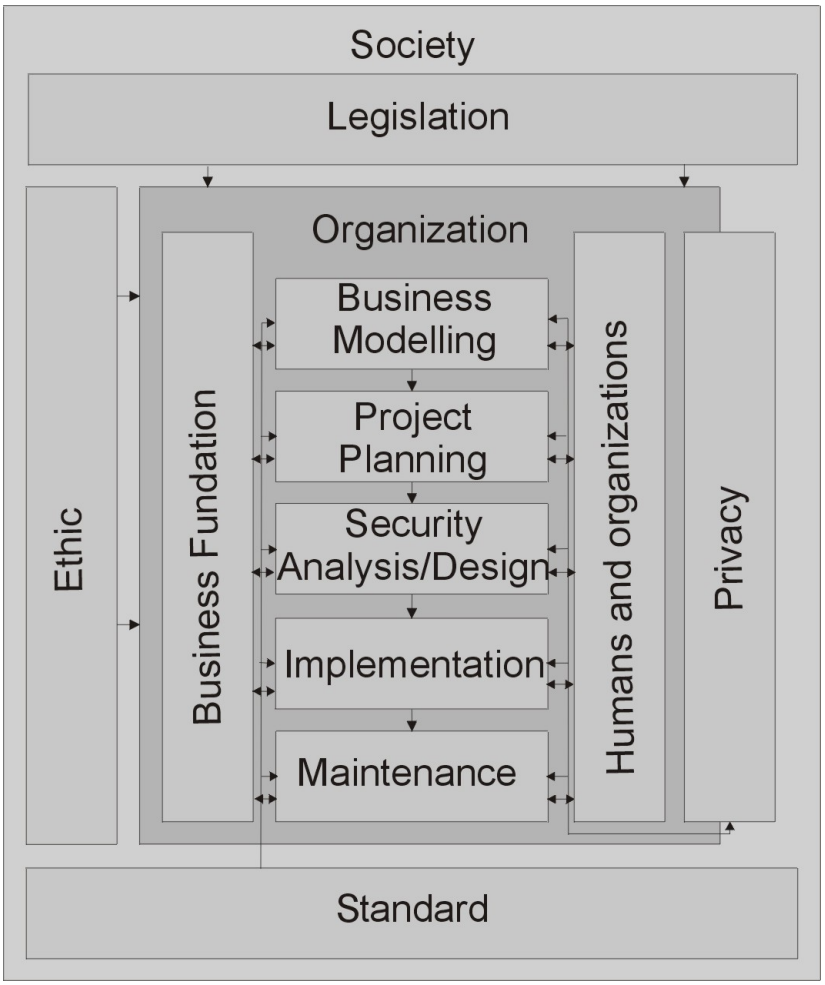

Fig. 2. Component interaction of the holistic security management framework

overall security position is taken – here the short time orientation enforces a focus to security needs emerging from operation (e.g. defining specialized product policies as a refinement of an overall policy).

Additionally, a project plan how to implement security efficiently, is developed and the project is launched.

Security Analysis and Design When the security needs have been defined, the concrete functions have to be identified in the security analysis.

It is not sufficient to look at the services that are available. It is also important to consider if the services are necessary for the company at all. In the part where security is considered as an insurance, a balance between costs and risks needs to be achieved. Earlier we have proposed a tool (A modified mean value approach – [Zuccato, 2002]), that can be helpful and probably have a good time behavior. Additionally the security requirements, that were developed during business modelling and project planning, need to be analyzed for there feasibility and profitability.

During the design the actual or the planned system concept (or if available, information system design) is enhanced with security functions.

Implementation of security functions When the planning of the implementation of security has been finished, an implementation project has to be launched where the required actions are undertaken.

This part is related to software engineering enhanced by special security needs as a higher requirement for specification and additional forms of quality assurance, like security review and penetration testing.

Maintenance When the system has started to run it must be kept running and every error has to be corrected. To ensure that the system stays on a high security level permanent improvements have to be conducted. This should be enforced by the workflow component.

Privacy Planning The component "Privacy Planning" plays a special role. It can be seen as a part of all three component groups, but does at the same time not fit 100% to any category. It was decided that it had to be included in the main workflow. The reason is that it is strongly coupled with the activities in that workflow.

It has, however, not become a built-in feature as eCommerce aims at private customers and business costumers. In the first case European legislation requires that privacy is enforced, whereas for the business costumer case privacy consideration it is not required by legislation.

But why is then privacy included at all? The answer is that we agree with [Fischer-Hübner, 2001] about its strong relations to security, and believe therefore that it is important in this context. An additional benefit is from our point of view that privacy can become a business enabler.

We would also like to distinguish privacy from data protection [4], which is a legal concept, because we agree with [Fischer-Hübner, 2001] in concern

[4] We think that data protection and privacy overlap eachother but are not the same, as privacy is related to a physical person whereas data protection deals with every data subject independent of its shape.

of a much broader definition of privacy. This distinction also explains why privacy is not included in the environmental component legislation, even if some reasons indicating that could be found.

Support components

The idea of support components was derived from business process modelling (see [Hammer and Champy, 1994] and [Vetschera, 2000]). Here the support process enables a carrying through of the main work, but does not directly contribute to the final product's function.

Humans and Organization During the whole workflow process, people have to be convinced that this makes sense and brings advantages for the company as well as for themselves. Additional tasks conducted are awareness raising and human relations management.

Business Foundation Without a proper business foundation, the management probably refuses to carry through the expensive implementation of security. Therefore the component must be a part of the process from the very beginning. Experience has shown that this support process should accompany the workflow over its whole lifetime to justify the investment.

Environmental components

As already mentioned, the environmental components hold a specific role. They are not really parts of the system but are still mentioned because of their extensive influence. It is important not to forget that they must be controlled too.

Legislation The legal framework provided by authorities gives not only a guidance of what is allowed in the implementation process, but it also gives certain obligations that have to be fulfilled.

Standards Security standards (and others) can be used as guidelines but can also, if used correctly, help to develop power by protecting the company from liability claims.

Ethics Ethics is the manifestation of the social expectations. There are no direct financial benefits in following ethics, but a nonchalance attitude may raise resistance and boycott.

3.2 Environment

Everything that is not part of the system can be considered as a part of the environment [Schoderbek et al., 1985].

It is possible to build an hierarchy of environments, which means that an environment can be part of an higher level environment. We are now going to look at the environments that are important for considering security on an eCommerce system level.

Society The society is the environment under which the whole system is running. It might look like if a consideration of the society is unnecessary. This is, however, not true. That is namely the source of moral obligations and the customer´s expectations. This phenomenon develops to a very complex task if the framework has to be built for an international acting corporation – as it can be assumed in e/mCommerce.

Organization The organization is the environment where the security management has to be implemented. As usual in system engineering, the restrictions and constrains have their source in the organizational environment. An additional factor is that an organization´s culture and behavior is very important for security management. To deal with that problem the framework methods from organizational behavior science should be used. The effect´s source is located out of the systems scope and therefore not entitled to be changed in short terms[5].

For the model, we want to establish, that the organization has a direct influence on the security management activities. The reasons are that security management focuses on the organization. The society on the other hand provides the frame for the organization, and influences in that way the organization attitudes.

3.3 Relations

A relationship bonds the objects together. Each relationship can be seen as unique [Schoderbek et al., 1985].

For the framework we will distinguish between flow dependent connections of the objects which we will call workflow. Additionally we also see a type of relation which is based on the timely order of the components. We have defined earlier, as a characteristic of the framework, the iterative nature of it and therefore we call these relation properties “iterations”.

Workflow

In a workflow, procedural rules about the order of tasks in a process, are formulated (after [WMC, 1999]). In this framework the term workflow will be used in a more relaxed way. An optimal flow of steps will be presented, which probably will not be carried through in that way in reality. The definition of workflow requires that each process step must be finished before the next starts, and that all steps are conducted. Here, both of these requirements will be used in a more “relaxed” way. The steps are likely to flow into each other – as figure 3 illustrates where the arrows moving into each other represents an overlap in time. The second major difference to the conventional understanding is that it should be possible to apply necessary components, and in that connection define the size and/or complexity of the system under consideration.

[5] Schoderbek et. all [Schoderbek et al., 1985] argues that environmental effects only can be changed in the long run .

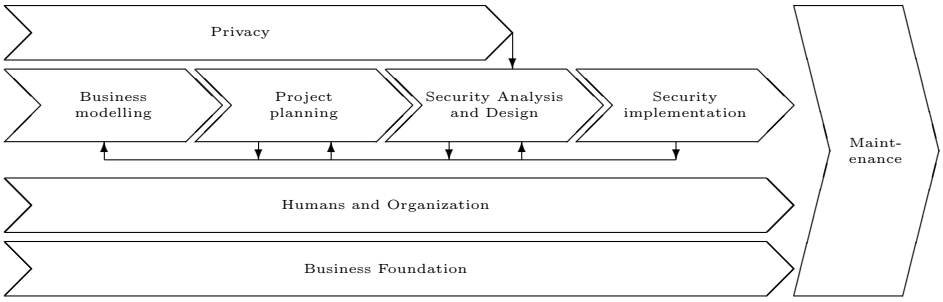

Fig. 3. HSMF workflow - note that parallel tasks always interact

As seen in the overall framework, some components have the purpose of support and others represent actual working steps. In general the steps should be conducted as presented in the order in figure 3. The support process has to be available during the whole development in the life cycle process.

It can be argued that the support processes should be integral parts of the main workflow components. However, for the framework this is considered as wrong because the ordering in the figure shows that the support processes have to be conducted continuously whereas the main workflow elements are required to end in finite time. The activities of the support process instead should be conducted without a break during the whole workflow life cycle.

Commonly the construction project is finished when the system has been launched. The activities in the workflow in the security management framework also change shape. It is reasonable to assume that after a construction, the support activities and the main activities are conducted together in the maintenance component. When an improvement or a reconsideration of security measurements becomes necessary, usually a new development life cycle is established. However, if the problem is small enough, the support process will only be conducted to the necessary extend.

Hence, this workflow gives a guidance and represents a kind of best practices. However, it should not be seen as a limiting factor and the borders can be considered as soft.

Iterations

We have argued earlier that the support of a state-of-the-art iterative development should be a characteristic of the proposed framework. By combining the life-cycle model of a project, and the knowledge of iterations-planning in the "unified software development process" in software engineering presented by [Jacobson et al., 1999], we conclude that 5 big phases are required:

Inception Where the preparatory work is conducted and the business centric preparation is carried out.

Elaboration Where the operative business conception is carried trough, as well as where an analysis and a rudimentary design are conducted.
Construction Where a security analysis is finished and the security design and implementation[6] are conducted.
Transition Where the security implementation has been finished.
Operation Where the security is maintained.

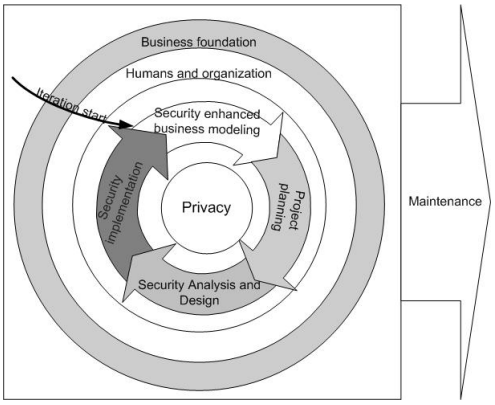

Fig. 4. Workflow for iterations transition

Except for the first phase – that means the inception phase where only one iteration is proposed – a number of iterations[7] can be performed in each phase. For each iteration the flow of activities can be see in figure 4. The block symbolizes that after each iteration the result transits to a limited maintenance, whereas the core maintenance activity starts when the developments are finished.

The effort for the conducted workflow activities depends on the phase. Table 1 lists the expected work amount[8] of activities in the different phases on a 5 steps scale (from ++ to −− – see in the description of the table). Unfortunately the expected amount cannot be determined (mostly due to lack of data) for all activities. These activities are marked with a question mark (?).

We have the feeling that some of presented efforts are not immediately understandable, and would therefore like to clarify how they come about. We would like to start with the inception phase where we see minor activities in analysis, design and implementation. These activities are conducted to develop a prototype to improve the understanding of the business model. Additionally, it has

[6] Note that this also includes documentation and quality assurance.
[7] For estimates of the number of iterations we would like to refer to [Jacobson et al., 1999] where, depending on the project size, such numbers are given.
[8] Note that available data sample cannot be considered as statistically sufficient to prove that distribution has been presented, but gives a good feeling and should therefore be understood as qualitative.

Table 1. Work amount HSMF phases

	Inception 1 Iter.	Elaboration n Iter.	Construction n Iter.	Transition n Iter.	Operation ?
Business modelling	++	+	−	−−	?
Security planning	+	++	∼	−	−
Security analysis/design	−	++	+	∼	?
Security implementation	−	∼	++	+	−
Maintenance	−	−	−	∼	++
Privacy	∼	∼	∼	∼	?
Humans and organization	++	−	+	+	+
Business foundation	++	+	−	∼	+

++ a lot of work; + some work; ∼ normal work; − less work; −− almost no work; ? unknown

turned out that a prototype supports the argumentation for necessity and sense of security, especially when seeking approval for the security enhanced business model. In the elaboration phase we see that analysis and design incorporates a lot of effort. This is due to the fact that at a later iteration in elaboration an analysis and design is used to investigate promising solutions and to generate a security centric software architecture.

The table also demonstrates the permanent presence of the support processes.

4 Experiences

In the introduction we mentioned that the framework development was conducted according to action research methods. For each iteration a security management project was launched. Two iterations of the three planned ones have already been finished and the results were introduced in the framework. The third project is ongoing and current experiences are described as well.

Due to space restrictions we will not present the whole project specification, but only a short description, some key data and the results will be presented. All of the cases were conducted in an Austrian Internet-banking.

4.1 Case 1 - Authentication redesign

The first case dealt with the redesign of authentication mechanisms. Due to the redesign purpose the business model was left unchanged. The project was of

medium size with 1 security architect, 2 developers, 1 quality assurance person and 2 external reviewers[9].

The following results followed and were introduced into the framework:

- The security design should have a sub-activity for considering external security products.
- Management support for implementing information security is crucial.
- The efficiency of security review methods for quality assurance is important.
- A broader support from top- and middle management is necessary and the internal control (revision) is well allied.

4.2 Case 2 - Security policy

The second case dealt with the formulation of security policies for e-banking. The main tasks were focused on the first components (business modelling and security planning) in the framework. For this case only one part-time assigned security architect was assigned. However, during the realization a lot of people from all over the organization (bank) worked with the project.

Following insights were gained:

- Use of security enhanced business modelling to motivate security is crucial.
- The scalability of the framework is an important feature.
- The support from business people is a crucial criteria for the first phases.
- The results from the first case were manifested.

4.3 Case 3 - Security management re-conception

This case is currently conducted. The status at the moment is that the case is in the middle of the elaboration phase.

The current insight is the importance of awareness building measures. The framework does already incorporate such activities into the support process concerning "humans and organizations", but the methods are now further evaluated.

5 Review

In the last section we mentioned that the proposed framework can be successfully applied in an eBanking environment. We consider this experience transferable to other electronic commerce areas, which, from our point of view should all have similar properties.

However, from a scientific viewpoint we also want to objectively review the framework. It seems therefore suitable to apply quality assurance methods from software engineering - namely review against requirements.

[9] Note that the role definition is not a part of the framework. However, given role names are deliberately chosen to improve the understanding.

We should therefore start with the question in how far every component contributes to one of our requirements. We will use a matrix for this purpose where we put the requirements and the components that have to be verified. To be able to distinguish between different support rates of a requirement we will use the same scale as in table 2 except for one thing: an empty field means that the requirement can not be linked to an activity. These values represent our personal opinion and are supported only by the practical experience presented in the last section.

Table 2. Requirement evaluation of framework and components

	Speed	Business centric	People centric	Technology oriented	Software development
Business modelling	+	++			+
Project planning	++	++			
Security analysis/design	+	∼	+	++	++
Security implementation	+		−	++	++
Maintenance		+	+		
Privacy		∼			
Business foundation	∼	++	+		
Humans and organization	∼		++		+

++ very well supported; + well supported; ∼ supported; − reduced effect; −− eliminated effect

By analyzing the framework we can see that every requirement is well supported by one or more components. We therefore conclude that the overall framework supports our requirements.

6 Conclusion

Given the requirement´s evaluation and the experience of application we believe that this framework can make its contribution to enhance a security management process in e/mCommerce.

In an ongoing research study we describe the detailed activities, workflows and artifacts for each component. Additionally we proposed techniques and tools to conduct the activities which should be suitable to further support the defined requirements.

Acknowledgments
Part of this work has been funded by the HumanIT research programm at Karlstad University. We therefore want to thank HumanIT for their support. We also want to

thank Prof. Simone Fischer-Hübner for her helpful comments and Linda Martinson for her support.

References

[WMC, 1999] (1999). *Terminology and Glossary, Document Number WFMC-TC-1011, Document Status - Issue 3.0*. Workflow Management Coalition.

[Backhouse and Dhilon, 1996] Backhouse, J. and Dhilon, G. (1996). Structures of responsibilities and security of information system. *European Journal of Information Systems*.

[Badenhorst and Eloff, 1989] Badenhorst, K. P. and Eloff, J. H. P. (1989). Framework of a methodology for the life cycle of computer security in organization. *Computer & Security*, 8:433–442.

[Boehm, 1986] Boehm, B. (1986). A spiral model of software development and enhancment. *ACM SIGSOFT, Software Engineering Notes*, 4(11).

[Booysen and Eloff, 1995] Booysen, H. and Eloff, J. (1995). A methodology for the development of secure application systems.

[Fischer-Hübner, 2001] Fischer-Hübner, S. (2001). *IT-Security and Privacy: design and use of privacy-enhancing security mechanims*. Lecture Notes in Computer Science. Springer.

[Hammer and Champy, 1994] Hammer, M. and Champy, J. (1994). *Reengineering the corparation: A manifesto for business revolution*. HarperBusiness.

[Hitchings, 1995] Hitchings, J. (1995). *Achieving an integrated design: the way forward for information security*.

[Jacobson et al., 1999] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process*. Object technology series. Addison Wesley Longman.

[Schoderbek et al., 1985] Schoderbek, P., Schoderbek, C., and Kefalas, A. (1985). *Management Systems: Conceptual Considerations*. Business Publications Inc., 3rd edition.

[Siponen, 2001] Siponen, M. (2001). *An Analysis of the Recent IS Security Developement Approaches*, volume Information Security Management: Global Challenges in the New Millennium, chapter VIII, pages 101 – 124. Idea Group Publishing.

[Vetschera, 2000] Vetschera, R. (2000). BBWL - Organisation und Plannung 2. Universität Wien, *http://www.bwl.univie.ac.at/bwl/org/Service/Skripten/Orga2.pdf*.

[von Bertalanffy, 1969] von Bertalanffy, L. (1969). *General System Theory*. George Braziller.

[Yngström, 1996] Yngström, L. (1996). *A systemic-holistic approach to academic programmes in IT security*. PhD thesis, Stockholm University.

[Zuccato, 2002] Zuccato, A. (2002). A modified mean value approach to assess security risks. In *ISSA-2 Proceedings*. Sout African Computer society.

Security Countermeasure Selection Method: A Fuzzy Approach to Uncertain Environments

Sungbaek Cho and Zbigniew Ciechanowicz

Information Security Group, Royal Holloway College,
University of London, Egham, Surrey, TW20 0EX, UK
{Sungbaek.Cho, Z.Ciechanowicz}@rhul.ac.uk

Abstract. During the risk management process, analysts or management should select the most appropriate countermeasures that meet security requirements under given constraints such as budget and other factors. However, such decision-making often becomes quite difficult due to a lack of precision in decision variables. This paper briefly overviews basic approaches to the countermeasure selection problem and proposes a linear programming approach that maximises the expected benefits of countermeasures under a budget constraint. This approach is based on fuzzy theory so that it can be used in a situation where information required for decision-making is incomplete.

1 Introduction

Risk management is one of the most important processes in managing IT security within organisations. Its objective is to protect IT assets such as data, hardware, software, personnel and facilities from all external (e.g. natural disasters) and internal (e.g. technical failures, sabotage, unauthorised access) threats so that the cost of losses resulting from the realisation of such threats are minimised [9]. However, it should be performed cost effectively as countermeasures (also called safeguards) require organisational resources even if they are merely in form of the management/operational procedures. For every organisation, there is some combination of effective loss prevention and a reasonable cost and therefore the purpose of risk management is to find such a balanced combination [17].

However, in general, the risk management process suffers from the absence of appropriate methodologies that aid analysts to select countermeasures under given constraints such as budget and other factors. Automated risk management tools are obviously very helpful and some of them provide a sound and structured approach to risk management. However, the decisions on countermeasure selection are left to management and/or analysts since organisational, technological, cultural and financial constraints they encounter may vary from one organisation to another.

Although there are a number of other important constraints, the budget constraint is particularly important. According to [3], only 27% of organisations (39% for large businesses) spend more than 1% of their IT budget on information

security and the expenditure on it is still seen as an overhead rather than as an investment. Therefore, there are specific needs for methodologies that address the issue of achieving maximum security under given budget constraints; that is the aim of this paper. This paper overviews basic principles and approaches to general countermeasure selection problems and then proposes a simple fuzzy linear programming model to provide a decision aid on countermeasure selection under budget constraints.

2 Overview of Countermeasure Selection

2.1 Risk Management Basics

Risk analysis and management (see [4] for details) starts with the identification of risks that may affect organisations. It is achieved by risk analysis, which includes the identification and assessment of triplets of asset, threat and vulnerability. This assessment can be either qualitative or quantitative. In qualitative assessment, the value of asset, frequency of threat and severity of vulnerability are expressed in terms of qualitative measures (usually linguistic expressions such as 'high', 'medium' and 'low'). In contrast, in quantitative assessment, they are expressed in numeric terms such as monetary values, frequencies and probabilities.

After measures (or levels) of risk for each triplet have been identified, the identification and selection of appropriate countermeasures can be made. Risk reduction can be achieved in various ways such as avoiding risks, transferring risks (e.g. insurance), reducing threats, vulnerabilities or possible impacts, and detecting/reacting/recovering from risks [4]. Therefore, organisations should take into account various ways of achieving risk reduction to ensure the identification/selection of the most appropriate set of countermeasures. In addition, it is important to note that there are a number of constraints which must be taken into consideration, such as time, financial, technical, sociological, environmental and legal constraints [4].

One important criterion for countermeasure selection is cost effectiveness since it would be inappropriate to recommend countermeasures which are more expensive to implement and maintain than the value of assets they are designed to protect [4]. Once appropriate and cost effective countermeasures have been identified and selected, they then need to be examined whether they provide an acceptable level of security. As no system can be made absolutely secure, there are always residual risks even if countermeasures are in place. If these residual risks cannot be tolerated, additional countermeasures may be considered and in this case financial constraints such as budget limits may need to be adjusted. Implementation of countermeasures and other relevant follow-up activities take place after the final decision on countermeasure selection and risk acceptance is completed.

2.2 Cost/Benefit Analysis

ALE (Annualised Loss Expectancy) is one of the most popular risk measures in quantitative risk analysis/management methods. In ALE-based approaches, risks are expressed in terms of an expected monetary loss, which can be calculated by multiplying a monetary loss by an annual rate of occurrence. As the monetary risk expression is focus of the ALE-based approaches, other variables required for making countermeasure selection decisions are also expressed in monetary terms.

Once countermeasure alternatives that meet security requirements have been identified, their cost effectiveness should be assessed. This is normally achieved by countermeasure cost/benefit analysis. In order to perform the cost/benefit analysis, all the relevant monetary data for a specific countermeasure (e.g. its cost and monetary loss reduction) must be provided.

In general, the cost of any countermeasure can be classified into two categories; implementation cost (including purchase cost) and running cost. In the cost/benefit analysis, the implementation cost is written off during the expected life time of the countermeasure. The running cost is an annual average of costs required for running/operating/maintaining the countermeasure. The benefit of the countermeasure is usually expressed as a monetary amount of loss reduction, which can be obtained by taking the ALE before countermeasure implementation and subtracting from this the ALE after countermeasure implementation. Then, the net benefit, which is a cost effectiveness measure, is simply calculated from the benefit less the countermeasure cost and those countermeasures for which net benefits are positive are regarded cost effective. Besides the net benefit, the return on investment (ROI) is also widely used for measuring the cost effectiveness. In RiskWatch [8], the ROI is defined as ROI=(Avoided Loss)+(Recovered Amount)/(Cost of Countermeasure), where (Avoided Loss)=(Loss before Countermeasure)−(Loss after Countermeasure).

In general investment project analyses, the running cost and benefit for each year of the project's life need to be estimated separately since they would vary from one year to the next. In addition, these values should be discounted by an appropriate discount rate, which reflects both interest rates and uncertainties, to measure overall performance as a net present value. However, a single estimate rather than different estimates for each year is often preferred in information security risk management due to difficulties in making detailed estimates.

The problem with applying the quantitative approach to countermeasure selection (i.e. the cost/benefit analysis) is that the monetary estimates do not normally have a reasonable level of accuracy. Therefore, the estimation is often based on subjective opinions held by analysts. For example, in RiskWatch [8], the avoided loss is calculated from the 'perceived' loss reduction (as a percentage). When there are n-to-n mappings between countermeasures and the threats they act against, estimation becomes much more difficult since analysts should estimate not only individual benefits but also collective benefits. Cost estimation itself is also very difficult although it seems to be relatively easier than the benefit estimation. When it is difficult to provide the cost estimates with rea-

sonable accuracy, it is better to overstate the costs in order to ensure that the net benefits of specific countermeasures are not overstated [16].

2.3 Qualitative Approach and Baseline Protection

The baseline approach [4] is one of the simplest ways of selecting countermeasures. In the baseline approach, standard countermeasures (those in standard materials such as BS7799 [1], GMITS-4 [5] and the IT-Baseline Protection Manual [6]) are applied to each identified triplet of asset, threat and vulnerability to achieve a baseline level of protection. In general, cost/benefit analysis or other similar evaluations are not performed in this approach since countermeasures identified in this approach are regarded so essential and/or mandatory that most (if not all) of them should be in place. The baseline approach provides reasonable protection at a minimum cost level.

However, it is difficult to say that it always provides cost effectiveness since it does not include a detailed analysis of security requirements. Therefore, if the level of baseline protection is set too high or too low when compared with the actual security requirements/risks, it will result in inadequate countermeasure selection. A medium or high level baseline may be sensible for many organisations in order to achieve sufficient protection, reliable security throughout the organisation, and reduction of organisational overhead [5]; for instance, the IT-Baseline Protection Manual [6] is intended to provide a medium level of protection.

2.4 Qualitative Approach: CRAMM

The difficulty with monetary estimates accounts for the move towards qualitative risk management. The qualitative approach looks more attractive than the quantitative one in that monetary estimates are only required at a minimum level. Another advantage of the qualitative approach is that it may provide more flexibility in establishing selection criteria than the quantitative approach since the latter approach often tends to focus on monetary values only. During the countermeasure identification/selection process, it is often required to consider not only cost effectiveness but also factors such as (1) ease of use of the safeguard, (2) transparency to the user, (3) the help provided to the users to perform their function, (4) the relative strength of the safeguard, and (4) the types of functions performed (e.g. prevention, deterrence, detection, recovery, correction, monitoring and awareness) [4].

Let us briefly overview the risk management stage in CRAMM [2], which is one of the most rigorous qualitative methods and often regarded as a defacto standard. It has been developed by the UK government and therefore is the preferred method within UK government departments.

Countermeasure Identification The CRAMM software contains a very large countermeasure library, so that it can automatically identify and recommend the countermeasures that meet the identified security requirements (i.e. measures

of risks calculated from the risk analysis stage). The countermeasures which fulfil a broadly similar purpose are collected together in countermeasure groups. Each group is further divided into sub-groups and each sub-group has a set of countermeasures that perform a common function. In addition, CRAMM has a pre-defined threat/countermeasure group table and therefore this table is used to identify appropriate countermeasures. Each countermeasure is also marked with a security level (or range of security levels) so that CRAMM can select appropriate countermeasures by comparing the measures of risks against the security levels assigned to individual countermeasures; countermeasures in the library will be selected for recommendation if the measure of risk falls within the range of security levels provided by the countermeasure.

Countermeasure Prioritisation The recommended countermeasures need to be reviewed and compared against those that are already in place to identify weaknesses or areas of over-provision. For this purpose, analysts can enter the status of each recommended countermeasure. For example, the status 'installed' is used to indicate that an existing or planned countermeasure fully meets the requirements laid by a recommended countermeasure, and the status 'accept level of risk' is used to indicate that the recommended countermeasure will not be implemented and the risk will be accepted. Once the analysts have chosen countermeasures for implementation after comparing what is recommended and what is in place, CRAMM then generates the countermeasure priority report, which provides a 'first pass' at prioritisation. Prioritisation of countermeasures is required as there are inevitable constraints such as budget and time, in many cases.

To calculate the priority score of each countermeasure, CRAMM considers the following factors: (1) cost rating, (2) effectiveness rating, (3) additional score for each threat that the countermeasure combats (4) type of protection provided by the countermeasure, (5) the highest measure of risk that led to the countermeasure being recommended for that asset, and (6) whether there are no alternative countermeasures already installed. To determine ratings for these factors, CRAMM uses pre-established information provided for each countermeasure. That is, developers of CRAMM have provided each countermeasure with required information such as its cost, supportive type, effectiveness, alternative countermeasure(s), and applicable asset(s).

The countermeasure cost may differ between different environments and therefore the pre-established ratings for the countermeasure cost is a rough estimate based on general fictitious systems. Moreover, estimated capital expenditure on the countermeasure implementation is assumed to be written off over five years and running cost was estimated at a £250 per diem rate. The total cost, expressed in monetary terms, is then converted to a corresponding linguistic rating ('low' if $\leq$£500, 'medium' if $>$ £500 and $\leq$£2,000 and 'high' if $>$£2,000). The effectiveness factor in CRAMM is defined as 'the degree to which a countermeasure meets the objectives of the sub-group that it is contained in' and the possible ratings are 'low', 'medium' or 'high'. As for the ratings against

Factor	Possible Value : Ratings
Cost Rating	L(Low): 10, M(Medium): 6, H(High): 2
Effectiveness Rating	L(Low): 2, M(Medium): 6, H(High): 10
No. of Threats Affected	Add 2 Per Each
Type of Countermeasure	RT(Reduce Threat): 10, RV(Reduce Vulnerability): 8
	RI(Reduce Impact): 6, D(Detect): 4, R(Recover): 2
Highest Risk Measure Associated	1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14

Table 1. CRAMM Priority Scoring System [2]

the type of protection, CRAMM gives higher ratings to countermeasures that prevent an incident from occurring than those that detect or facilitate recovery from an incident (however, it is important to implement a balanced set of various types of countermeasures). The overall priority rating is then reduced by 50% if an alternative countermeasure is already in place. Table 1 shows how the overall priority scores are calculated.

Countermeasure Selection Based on, but not bound by, the priority scores of countermeasures, the selection of countermeasures is to be made. To make decisions on the countermeasure selection, the analysts are required to estimate the cost of each countermeasure to be implemented. Since it is often difficult to estimate the cost accurately, bands (ranges) are normally used to record the costs in CRAMM. Although all the risk management processes have been guided by the CRAMM methodology (and software) so far, the decision-making on the actual selection and prioritisation of countermeasures is made by the analysts. As shown in the case of CRAMM, in the qualitative risk management approach, only cost factors are estimated in monetary terms and therefore the cost/benefit analysis is not performed explicitly. However, cost-effectiveness is still considered in the qualitative approach.

3 Requirements in Countermeasure Selection

3.1 Problem Definition

In general, the term, 'countermeasure selection' has a broad meaning, which varies from the prioritisation of implementation plans (the narrowest concept) to the identification and selection of required countermeasures (the widest concept). Therefore, before we go further, we define the scope of our approach presented in this paper.

As mentioned earlier, the proposed approach is to support decision-making in countermeasure selection under budget constraints. However, it does not include the identification of countermeasures and evaluation of existing countermeasures. Therefore, the countermeasures dealt with in our approach are limited to those countermeasures that have been screened initially. The initially screened countermeasures imply that (1) they meet the security requirements, (2) the needs

for them have been somehow verified, and (3) their feasibility/compatibility with the organisation's culture/skill have been assessed.

Then, the remaining problem is to select them for implementation according to the available budget. If there is no budget limit, the countermeasure selection problem becomes a typical prioritisation problem, which can be solved by CRAMM's priority scoring system or other similar approaches. Ideally, all the countermeasures with high priorities should be implemented immediately. Therefore, the increase in the budget should be provided if the total implementation cost exceeds the existing budget. However, it may not be always possible and as a result, only some of them, within the budget limit, may be approved by senior management (the rest of them are then to be considered in next year's budget). The proposed approach in this paper aims at providing a decision aid regarding this problem.

3.2 0-1 Selection Problem

The problem to be handled by our approach is a typical binary (0-1) selection problem (i.e. to select or not to select under the budget constraint), which can be modelled by traditional linear programming (LP) techniques, more specifically integer programming (IP) with binary variables. The integer programming model in Equation (1) is for optimal selection of countermeasures that maximises the potential benefits under a given budget.

$$\text{Maximise } \sum_i b_i x_i \text{ subject to } \sum_i c_i x_i \leq l \text{ and } x_i \text{ is 0 or 1 for all } i, \qquad (1)$$

where x_i is a decision variable (1 if countermeasure i is selected and 0 otherwise), b_i is the benefit (or loss reduction) of countermeasure i, c_i is the cost of countermeasure i, and l is the budget limit on capital expenditures.

If there are other constraints to be considered, they can be added to Equation (1) in form of additional constraints. For example, senior management might have imposed a limit (l_2) on the total annual running cost; this can be modelled as $a_i x_i \leq l_2$, where a_i is the annual running cost of countermeasure i. Other examples are (1) $x_1 + x_4 + x_6 = 1$ if countermeasure 1, 4 and 6 are exclusive of each other, and (2) $x_1 + x_2 + x_6 \geq 1$ if at least one of the countermeasure 1, 2, and 6 must be selected.

In the ALE-oriented approaches, the benefit factor b_i would usually be expressed as the monetary amount of loss reduction. On the other hand, b_i would be the priority rating or other similar measures in qualitative approaches. However, it must be noticed that Equation (1) may not be compatible with monetary benefit models. The above LP model is valid only when the overall benefit can be expressed as a linear sum of benefits of each countermeasure.

When several countermeasures are used to combat one common threat, the overall benefit may be different from the linear sum of marginal benefits of individual countermeasures. For example, suppose that the annual monetary benefits

of the system-generated password system and token-based authentication system against the masquerading threat are estimated to be £300 and £500, respectively. However, it does not imply that the total benefit will be £800 when both are implemented. This can be due to the fact that these two countermeasures are interrelated. Therefore, the benefits of every applicable combination of countermeasures should be estimated, but it will certainly add huge amounts of burdens to analysts even if it is possible to estimate these benefits. Due to this reason, we are in favour of qualitative benefit measures.

3.3 Benefit Scoring System

To obtain qualitative measures of countermeasure benefits, a simple additive scoring system is used in the proposed approach and it can be expressed as:

$$\text{Benefit Score of Countermeasure } i \ (b_i) = \sum_j r_{ij} w_j, \tag{2}$$

where r_{ij} is the rating of countermeasure i with respect to benefit evaluation criterion j and w_j is the weight of criterion j.

This benefit scoring system is very similar to the CRAMM priority scoring system (shown in Table 1). However, its major difference to the CRAMM priority score is that the cost factor is not considered since it will be considered as a constraint in the LP model. Although all the criteria and weights in Table 1, except the cost rating, could be used, we will consider only three of them in this paper for demonstration purposes. They are (1) CR_1: number of threats, (2) CR_2: effectiveness and (3) CR_3: highest risk. The purpose of this paper is to propose a decision aid framework, rather than a complete solution. Therefore, these criteria are not intended to be complete. Moreover, a variety of other criteria may be added or substituted as there is no absolute answer that can be uniformly applied to all situations. If discounting is required to give lower priorities to countermeasures for which alternatives are in place, it can be achieved by multiplying an appropriate discount rate, as in the CRAMM priority scoring system.

The first criterion, the 'number of threats', corresponds to CRAMM's criterion 'number of threats affected'. If an automated risk analysis tool such as CRAMM is used, it is relatively easy to obtain the number of threats that the countermeasure combats. However, it could be difficult to obtain that number if such tools are not available or different methodologies are used. Regarding this situation, ratings against this criterion can be qualitative ratings ranging from 0 (lowest) to 10 (highest). The same assumptions are given to the second and third criteria.

If criteria have different rating scales, a transformation of scales should be applied. The linear scale transformation is one of the most popular scale transformation methods and is defined as $r'_{ij} = r_{ij}/r^*_j$ for beneficial criteria and $r'_{ij} = r^-_j/r_{ij}$ for cost criteria, where $r^*_j = \max_i r_{ij}$ and $r^-_j = \min_i r_{ij}$. Therefore, if different rating scales are used, r'_{ij} should be used instead of r_{ij} in Equation (2).

3.4 Need for Fuzzy Approach

Risk analysis/management must often rely on uncertain inputs/estimates that have been obtained from speculation, best guesses, incomplete data, and many unproven assumptions [7]. These uncertain values should be addressed properly in risk analysis/management since they will be regarded as definite values otherwise. In Equation (2), ratings against some criteria may not be determined definitively due to uncertainties and as a result, analysts may prefer to assign bands of ratings rather than single ones. For example, the effectiveness rating of a specific countermeasure may be expressed as '[4,6]' rather than '5'.

To address the uncertainties found in decision-making problems, extensive research has been conducted and the probabilistic approaches and fuzzy approaches have been the most popular ones for decades. In general, the former approach is regarded better if there is sufficient information to build accurate probabilistic models of uncertainties and the latter approach is regarded better if little information is available [11].

In risk analysis/management, sufficient information for probabilistic models is not normally available and therefore, the fuzzy approach looks more promising; for example, [18] has proposed a risk analysis based on fuzzy logics at a conceptual level. To deal with situations where analysts prefer to specify vague values rather than precise ones due to uncertainties, we have applied a fuzzy number approach to the countermeasure selection problem.

4 Fuzzy Binary Selection Model

4.1 Definitions and Notations

In the following, we briefly review some basic definitions and notations used in fuzzy set theory as found in [15, 20].

Fuzzy Set A fuzzy set $\tilde{A}$ of the universe discourse X (i.e. the range of all possible values) is a set of ordered pairs $\{(x_1, \mu_{\tilde{A}}(x_1)),(x_2, \mu_{\tilde{A}}(x_2)),\ldots,(x_n, \mu_{\tilde{A}}(x_n))\}$, where $\mu_{\tilde{A}}$ ($\mu_{\tilde{A}} : X \rightarrow [0,1]$) is the membership function of $\tilde{A}$, and $\mu_{\tilde{A}}$ indicates the grade of membership[1] of x_i in $\tilde{A}$.

A fuzzy set $\tilde{A}$ of the universe discourse X is called convex if and only if $\mu_{\tilde{A}}(\lambda x_1 + (1-\lambda)x_2) \geq Min(\mu_{\tilde{A}}(x_1), \mu_{\tilde{A}}(x_2))$ for all x_1, x_2 in X where $\lambda \in [0,1]$, and it is called normal implying that $\exists x_i \in X$, $\mu_{\tilde{A}}(x_i) = 1$.

Fuzzy Number A fuzzy number $\tilde{n}$ is defined as a fuzzy subset that is convex and normal, characterised by an interval of real numbers, each of which has a grade of membership between 0 and 1.

[1] As an illustration, let us consider the statement 'John is old'. We might think of assigning a value ranging from 0 (absolutely false) to 1 (absolutely true) to the statement to indicate to what extent we support it. For instance, we might assign 0.8 ($\mu_{old}(John) = 0.8$) if we know that John's age is 75.

	Low	Medium	High
Countermeasure Rating	(0,2,4)	(4,6,7)	(7,9,10)
Criterion Weight	(0,0.2,0.4)	(0.4,0.6,0.7)	(0.7,0.9,0.1)

Table 2. Linguistic Expression and Fuzzy Numbers

Triangular Fuzzy Number A triangular fuzzy number $\tilde{n}$ is the fuzzy number that can be defined by three parameters, expressed as (n_1, n_2, n_3). Then its membership function $\mu_{\tilde{n}}(x)$ is defined as $\mu_{\tilde{n}}(x) = (x - n_1)/(n_2 - n_1)$ if $n_1 \leq x \leq n_2$, $\mu_{\tilde{n}}(x) = (x - n_3)/(n_2 - n_3)$ if $n_2 \leq x \leq n_3$, and $\mu_{\tilde{n}}(x) = 0$ otherwise.

α-cut The α-cut of a fuzzy number $\tilde{n}$ is defined as $\tilde{n}^\alpha = \{x_i : \mu_{\tilde{n}}(x) \geq \alpha, x_i \in X\}$ where $\alpha \in [0, 1]$. $\tilde{n}^\alpha$ is a non-empty bounded closed interval contained in X and it can be denoted by $\tilde{n}^\alpha = [n_l^\alpha, n_u^\alpha]$ where n_l^α, n_u^α are the lower and upper bounds of the closed interval respectively.

If $\tilde{n}$ is a fuzzy number and $n_l^\alpha > 0$ for $\alpha \in [0, 1]$, $\tilde{n}$ is then called a positive fuzzy number. Given any two positive fuzzy numbers $\tilde{m}$, $\tilde{n}$ and a positive real number r, the α-cut of two fuzzy numbers are $[m_l^\alpha, m_u^\alpha]$, $[n_l^\alpha, n_u^\alpha]$, respectively. Then, some important operations on these positive fuzzy numbers can be expressed as follows:

- $(\tilde{m} + \tilde{n})^\alpha = [m_l^\alpha + n_l^\alpha, m_u^\alpha + n_u^\alpha]$, $(\tilde{m} - \tilde{n})^\alpha = [m_l^\alpha - n_l^\alpha, m_u^\alpha - n_u^\alpha]$,
- $(\tilde{m}(\cdot)\tilde{n})^\alpha = [m_l^\alpha \cdot n_l^\alpha, m_u^\alpha \cdot n_u^\alpha]$, $(\tilde{m}(:)\tilde{n})^\alpha = [m_l^\alpha / n_l^\alpha, m_u^\alpha / n_u^\alpha]$,
- $(\tilde{m}^\alpha)^{-1} = [1/m_u^\alpha, 1/m_l^\alpha]$, $(\tilde{m}(\cdot)r)^\alpha = [m_l^\alpha \cdot r, m_u^\alpha \cdot r]$.

4.2 Fuzzy Numbers in Benefit Scores

Intuitively, triangular fuzzy numbers are relatively easy to use in expressing an agent's subjective assessment (as only three numbers are required to express the vagueness of specific data) and therefore they have been applied to a number of applications. In our countermeasure selection approach, the ratings of countermeasures against three criteria, 'number of threats', 'effectiveness' and 'highest risk' and weights of the criteria are defined as triangular fuzzy numbers.

As mentioned earlier, the range of ratings for each criterion is $[0, 10]$. If the analysts know exact values (e.g. when objective criteria are used), they may assign definite ratings (i.e. crisp values) to each countermeasure against a specific criterion. However, it is not always possible for them to provide such crisp rating values and therefore bands of ratings or linguistic expressions (e.g., 'high', 'medium' and 'low') may often be preferred. We assume that the analysts will provide linguistic expressions for the ratings. Then, these linguistic expressions can be expressed as triangular fuzzy numbers as shown in Table 2. For example, if an analyst provides the linguistic expression 'low' for a specific rating, then it will be interpreted as 'the rating will be somewhere between 0 and 4 and the most likely value is 2'. The weights of criteria will be determined similarly, but on a smaller scale as shown in Table 2.

The analyst can also provide rating bands instead of linguistic expressions. For example, if the rating band $[a, c]$ is provided, it will be interpreted as (a, b, c), where b is the mean of a and c[2]. Moreover, crisp values can be used if it is possible for the analysts to provide these values (e.g. the exact number of threats that the countermeasure combats is known); however, in this case, a linear scale transformation should be applied to all the ratings. A crisp value a can be interpreted as (a, a, a). Then, the benefit score in Equation (2) can be rewritten as:

$$\text{Benefit Score of Countermeasure } i \ (\tilde{b}_i) = \sum_j \tilde{r}_{ij}\tilde{w}_j \tag{3}$$

The benefit score of countermeasure i, $\tilde{b}_i$ can be obtained through fuzzy number addition and multiplication operations, which are defined as $(a_1, a_2, a_3) \bigoplus (b_1, b_2, b_3) = (a_1 + b_1, a_2 + b_2, a_3 + b_3)$ and $(a_1, a_2, a_3) \bigotimes (b_1, b_2, b_3) = (a_1 \times b_1, a_2 \times b_2, a_3 \times b_3)$, respectively.

As mentioned above, the linear scale transformation should be performed if different rating scales are used. In this case, the $\tilde{r}_{ij}$ in Equation (3) should be replaced by $\tilde{r}'_{ij}$. The linear scale transformation for triangular fuzzy numbers is defined as follows [10]: $\tilde{r}'_{ij} = (r_{ij1}/r^*_j, r_{ij2}/r^*_j, r_{ij3}/r^*_j)$ for beneficial criteria, and $\tilde{r}'_{ij} = (r^-_j/r_{ij3}, r^-_j/r_{ij2}, r^-_j/r_{ij1})$ for cost criteria, where $\tilde{r}_{ij} = (r_{ij1}, r_{ij2}, r_{ij3})$, $r^*_j = \max_i r_{ij3}$ and $r^-_j = \min_i r_{ij1}$.

4.3 Fuzzy 0-1 Selection Model

In our proposed approach, the benefit scores of each countermeasures are expressed by triangular fuzzy numbers. In contrast, we initially assume that crisp values are used for costs and other constraints. Then, Equation (1) can be rewritten as follows:

$$\text{Maximise} \quad \sum_i \tilde{b}_i x_i \text{ subject to } \sum_i c_i x_i \leq l \text{ and } x_i \text{ is 0 or 1 for all } i \tag{4}$$

A solution to Equation (4), $x' = [x'_1, x'_{,2}, \ldots, x'_n]^T$, is called optimal if the fuzzy number $g(x')$ is the greatest in the set $A = \{g(x) | x \in X\}$, where $g(x) = \sum_i \tilde{b}_i x_i$ (abbreviated to $\tilde{b}x$,) and X is the set of feasible solutions. However, there is no absolute way of comparing the fuzzy numbers and identifying the greatest one.

Ranking Function There are two main approaches for comparing a pair of fuzzy numbers. One is based on possibility theory and the other is based on ranking functions [14]. The possibility theory-based comparison in [13] and its variations, compare fuzzy numbers using four dominance indices (possibility of

[2] For more detailed evaluation, trapezoidal fuzzy numbers or more general L-R fuzzy numbers may be required. However, we use triangular fuzzy numbers in this paper for the sake of simplicity.

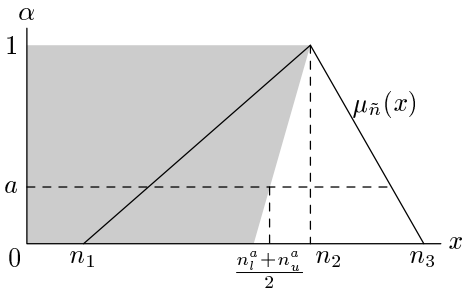

Fig. 1. Area Defined by Fuzzy Number

dominance, possibility of strict dominance, necessity of dominance and necessity of strict dominance). However, the main drawback is that it may produce non-consistency in dominance indices and counter-intuitive results cannot be prevented [14].

In order to find an optimal solution for Equation (4), we use the ranking function approach, as shown in Equation (5). The ranking function approach uses the ranking function f to map a fuzzy number onto a real number so as to rank fuzzy numbers according to an ordering on corresponding real numbers.

$$f(\tilde{b}_i) = \int_0^1 \frac{1}{2}(b_i{}_l^{\alpha} + b_i{}_u^{\alpha})d\alpha \tag{5}$$

The above ranking function provides intuitive rankings in that it considers the sizes of areas defined by fuzzy numbers (the shaded area in Fig. 1); it is based on the concept of area compensation [14] and is also equivalent to the third index of Yager [19] applied to normal fuzzy numbers. Since $\tilde{b}_i$ is a triangular fuzzy number, it can be rewritten as $f(\tilde{b}_i) = (b_{i1} + 2b_{i2} + b_{i3})/4$. Another reason for using this ranking function is that it preserves the linearity of the objective function whereas some other ranking functions result in nonlinear objective functions. The defuzzified version of the objective function $f(\tilde{b}x)$ now substitutes $\tilde{b}x$ and therefore the goal is to find a solution that maximises:

$$f(\tilde{b}x) = \sum_i \frac{1}{4}(b_{i1} + 2b_{i2} + b_{i3})x_i, \text{ subject to } \sum_i c_i x_i \leq l \tag{6}$$

Fuzzy Coefficients in Constraints It is not always possible for analysts to provide crisp values for countermeasure costs although we have assumed crisp constraints so far. Costing of countermeasures is a particularly difficult process; costs related to development, installation and operating are difficult to determine even if the hardware cost is easy to estimate. Furthermore, costs related

to procedural countermeasures are difficult (if not impossible) to estimate accurately [12]. Therefore, imprecision in cost estimates should also be considered in the countermeasure selection.

Fuzzy constraints can be considered within Equation (6), instead of crisp constraints. When constraints with fuzzy coefficients are considered, a specific approach is required since the satisfaction of a particular constraint cannot be determined in a deterministic way such as 'yes' or 'no'. For example, how can we tell that the solution x' satisfies the constraint $\tilde{c}x' \leq \tilde{l}$?; as the fuzzy number $\tilde{c}x'$ varies from its lower bound to upper bound, its value may be within or outside the range of the budget limit. The extent to which the fuzzy constraints are satisfied should therefore be taken into account rather than just 'yes' or 'no'. For this purpose, we adopt the comparison index in [14], which comes from the concept of area compensation.

$$I(\tilde{c}x \leq \tilde{l}) \geq \sigma \iff \sum_i [\sigma E^*(\tilde{c}_i) + (1-\sigma)E_*(\tilde{c}_i)]x_i \leq \sigma E^*(\tilde{l}) + (1-\sigma)E_*(\tilde{l}), \quad (7)$$

where $E^*(\tilde{c}_i) = \int_0^1 c_{iu}^{\alpha} d\alpha$, $E_*(\tilde{c}_i) = \int_0^1 c_{il}^{\alpha} d\alpha$, $E^*(\tilde{l}) = \int_0^1 l_u^{\alpha} d\alpha$, and $E_*(\tilde{l}) = \int_0^1 l_l^{\alpha} d\alpha$, respectively.

The comparison index $I(\tilde{c}x \leq \tilde{l})$ represents the degree to which $\tilde{c}x \leq \tilde{l}$ is satisfied and σ denotes the satisfaction requirement ($\sigma \in [0,1]$). Instead of the constraint $\sum_i c_i x_i \leq l$, the defuzzified version of the fuzzy constraint, Equation (8), will be used when fuzzy costs are used. The level of satisfaction requirement, σ, will be determined by decision makers (analysts or senior management). Setting $\sigma = 1$ results in very conservative selection since the upper mean value of $\tilde{c}x$ is constrained to be smaller than the lower mean value of $\tilde{l}$ and setting $\sigma = 1/2$ is typical [14]. For triangular fuzzy costs and budget limit, $E^*(\tilde{c}_i)$, $E_*(\tilde{c}_i)$, $E^*(\tilde{l})$ and $E_*(\tilde{l})$ become $(c_{i2} + c_{i3})/2$, $(c_{i1} + c_{i2})/2$, $(l_2 + l_3)/2$ and $(l_1 + l_2)/2$, respectively. When $\sigma = 1/2$, Equation (8) becomes $f(\tilde{c}x) \leq f(\tilde{l})$, i.e., $\sum_i (1/4)(c_{i1} + 2c_{i2} + c_{i3})x_i \leq (l_1 + 2l_2 + l_3)/4$.

4.4 Example

In this section, we provide a simple example with fictitious data to show how the proposed approach can be used. Suppose that there are three countermeasures that have been initially screened. These countermeasures are: (1) smart card system to provide access control to workstations, (2) firewall system to block the transmissions of all unnecessary packets, and (3) automatic fire detectors and alarms. Table 3 shows their benefit ratings and implementation costs. The weights of criteria are also shown in Table 3. In addition, assume that the budget limit is 42 (=(42,42,42)).

By Equation (3), benefit ratings of countermeasures can be calculated. For instance, the benefit rating of countermeasure 1, smart card system, is (7.2,14.4,18.9) (= (4,6,7) $\otimes$ (0.4,0.6,0.7) $\oplus$ (7,9,10) $\otimes$ (0.4,0.6,0.7) $\oplus$ (4,6,7) $\otimes$ (0.7,0.9,1)). Likewise, the benefit ratings for countermeasure 2 and 3 are (9.3,17.1,21.9) and

Criteria	CR_1	CR_2	CR_3	Cost
Weights	M; (4,6,7)	M; (4,6,7)	H: (7,9,10)	
1. Smart Card	M: (4,6,7)	H: (7,9,10)	M: (4,6,7)	(19,23,24)
2. Firewall	H: (7,9,10)	M: (4,6,7)	H: (7,9,10)	(17,18,21)
3. Fire Alarm	L: (0,2,4)	M: (4,6,7)	M: (4,6,7)	(9,12,13)

Table 3. Example Ratings and Weights

(4.4,10.2,14.7) respectively. From these benefit scores, the defuzzified values of benefit scores of each countermeasure are 13.725, 16.35 and 9.875 respectively. On the assumption that $\sigma = 1/2$, we defuzzify each fuzzy cost of countermeasures and the results are 22.5, 18.5 and 11.5, respectively. Then, the countermeasure selection problem can be formulated as follows:

$$\begin{aligned} &\text{Maximise } 13.725x_1 + 16.35x_2 + 9.875x_3 \\ &\text{subject to } 22.5x_1 + 18.5x_2 + 11.5x_3 \leq 42 \end{aligned} \tag{8}$$

The solution of this problem is $x_1 = 1$, $x_2 = 1$ and $x_3 = 0$. Therefore, we can select countermeasure 1 and 2 for implementation to maximise benefits under the given budget limit. By solving the problem at various levels of constraint satisfaction, sensitivity analysis can be conducted.

5 Conclusion

In this paper, we extended the concept of CRAMM priority rating to the actual countermeasure selection problem that considers the budget limit simultaneously. When the actual countermeasure selection decision is being made, the complete and precise information is not available in many cases. As a result, the input values for benefit scores and the cost estimates may not be presented as exact numbers and therefore imprecise data should be handled properly to achieve sounder decision-making.

However, our approach has several limitations. Firstly, the scope of this paper does not include identification of benefit evaluation criteria although qualitative cost-effectiveness measures depends heavily on the soundness, effectiveness and completeness of evaluation criteria. Secondly, this approach does not provide consideration of collective benefits of multiple countermeasures; we avoided this problem by simply not considering quantitative loss reduction. Thirdly, this paper does not cover validation of the proposed approach although fuzzy set theory looks attractive; more research should be conducted from the practical viewpoint (e.g. applicability and effectiveness to large-scale problems).

Due to these limitations, the solution obtained from the proposed approach should just be treated as a reference for decision-making, rather than as a complete and definite one. There has been relatively little effort within the area of countermeasure selection. There is a real need for intensive research on this subject.

References

1. BS7799-Part 1: Code of Practice for Information Security Management. British Standard Institution (1999)
2. CRAMM: CRAMM User Guide Ver. 2.0. UK Security Service (2001)
3. Information Security Breaches Survey 2002, UK Department of Trade and Industry. (2002)
4. ISO/IEC TR 13335-3: Guideline for Management of IT Security-Part3 (GMITS-3) - Techniques for the Management of IT Security. (1997)
5. ISO/IEC TR 13335-4: Guideline for Management of IT Security-Part4 (GMITS-4) - Selection of Safeguards. (1999)
6. IT-Baseline Protection Manual. German Information Security Agency (2001)
7. NIST Handbook: An Introduction to Computer Security. NIST Special Publication 800-12 (1995)
8. RiskWatch: RiskWatch v.8.1 Physical Security Training Manual. Risk Watch Inc. (1999)
9. Bandyopadhyay, K., Mykytyn, P.P., Mykytyn, K.: A Framework for Integrated risk management in information technology, Management Decision **37/5** (1999) 437-444
10. Chen,C.T.: A fuzzy approach to select the location of the distribution center. Fuzzy Sets and Systems **118** (2001) 65-73
11. Chen, S., Nikolaidis, E., Cudney, H. H.: Comparison of Probabilistic and Fuzzy Set Methods for Designing under Uncertainty. Proceedings of AIAA Structures, Structural Dynamics, and Materials Conference, St. Louis, MO, Apr. 12-15 (1999) 2860-2874
12. Ciechanowicz, Z.: Risk analysis: requirements, conflicts and problems. Computers & Security **16** (1997) 223-232
13. Dubois, D., Prade, H.: Ranking of fuzzy numbers in the setting of possibility theory. Information Science **30** (1983) 183-244
14. Fortemps, P.: Fuzzy Sets for Modelling and Handling Imprecision and Flexibility. PhD Thesis. Faculte Polytechque de Mons (1996)
15. Kaufmann, A., Gupta, M.M.: Introduction to Fuzzy Arithmetic Theory and Applications. Van Nostrand Reinhold, New York (1985)
16. Ozier, W.: Risk analysis and assessment. in: Krause, M., Tipton, H.F. (eds.): Handbook of Information Security Management. CRC Press (1999) 425-464
17. Rainer, R.K., Snyder, C.A., Carr, H.H.: Risk analysis for information technology. Journal of Management Information Systems **8/1** (1991) 129-147
18. de Ru, W.G., Eloff, J.H.P.: Risk analysis modelling with the use of fuzzy logic. Computers & Security **15/3** (1996) 239-248
19. Yager, R.R.: A procedure for ordering fuzzy subsets of the unit interval. Information and Science **24** (1981) 143-161
20. Zadeh, L.A.: Fuzzy sets. Information and Control **8** (1965) 338-353

Model-based Risk Analysis of Security Critical Systems

Siv Hilde Houmb[1], Trond Stølen Gustavsen[1], Ketil Stølen[2], Bjørn Axel Gran[3]

[1] Telenor R&D, Norway, {siv-hilde.houmb, trond-stolen.gustavsen}@telenor.com

[2] Sintef Telecom & Informatics, Norway, kst@sintef.no

[3] Institute for Energy Technology, bjornag@hrp.no

CORAS (www.nr.no/coras) is a EU funded R&D project (IST-2000-25031) developing a methodology and a framework for model-based risk assessment based on AS/NZS 4360 [1]. CORAS focus on security critical systems in general, with particular emphasis on IT security. IT security includes all aspects related to defining, achieving, and maintaining confidentiality, integrity, availability, non-repudiation, accountability, authenticity, and reliability of IT systems (ISO/IEC TR 13335-1:2001 [2]). The focus is on controlling risks by using well know risk analysis methods from the safety domain, such as HazOp [3], FMEA [4] and FTA [5], which have been used within for example the chemical and nuclear industry since World War II [3].

The presentation will focus on how to use UML (Unified Modeling Language) behavioural diagrams [6] as input diagrams to risk analysis. The approach will be exemplified by demonstrating how a UML sequence diagram can be used to support HazOp for risk identification. Our approach includes both guidelines on how to construct the input diagrams from existing system documentation and how to perform a risk analysis using a particular input diagram.

The presentation will conclude with summarising some of the preliminary results from the CORAS project, with emphasis on how we have and will verify the usability of the methodology developed within the project and in particular the methodology developed for using input diagrams to support risk analysis.

References

1. AS/NZS 4360:1999 Risk management (1999).
2. ISO/IEC TR 13335-1:2001: Information technology – Guidelines for the management of IT Security – Part 1: Concepts and models for IT Security.
3. Leveson, Nancy G., "SAFEWARE, System, Safety and Computers", Addison-Wesley, ISBN: 0-201-11972-2, 1995.
4. Bouti, A., Ait Kadi, D., A state-of-the-art review of FMEA/FMECA. International Journal of Reliability, Quality and Safety Engineering 1 (1994), 515-543.
5. IEC 1025:1990 Fault tree analysis (FTA) (1990).
6. Booch, Grady, Jacobson, Ivar, and Rumbaugh, James, "The Unified Modeling Language Reference Manual", Addison-Wesley Longman Inc., ISBN: 0-201-30998-X, 1999.

Mobile Applications and Multilateral Security

Kai Rannenberg
Mobile Commerce & Multilateral Security
Johann Wolfgang Goethe University Frankfurt, German
Kai.Rannenberg@whatismobile.de

Abstract. "Mobile Whatever" had become a popular buzzword and inspired a lot of more or less realistic application scenarios, especially when UMTS licenses were in high demand. Now much of the euphoria is over and many market players had to revise their optimistic business expectations. "What are the revolutionary new applications that we need all this bandwidth for?" is a question one can hear very often. However, the hangover after the hype sometimes seems to overshadow the real chances of enhanced mobile applications that lie in enhancing privacy and security of day to day communication, e.g. by using the extra interaction channels for negotiations.

Following a short description of Mobile Commerce and the work this talk will introduce the concept of "Multilateral Security" that aims at balancing the competing security requirements of the different parties. The discussion will include scenarios and examples of research on everyday communication and interaction, such as Reachability Management and access to mobile portals. They indicate that on one side the prudent use of the extra communication channels and options can enhance security and privacy while on the other side this can make mobile applications more popular and acceptable.

References

Kai Rannenberg: CamWebSim and Friends: Steps towards Personal Security Assistants; Pp. 173 - 176 in Viktor Seige et al.: The Trends and Challenges of Modern Financial Services – Proceedings of the Information Security Summit; May 29-30, 2002, Prague; Tate International; ISBN 80-902858-5-6

Kai Rannenberg: How much negotiation and detail can users handle? Pp. 37-54 in Frédéric Cuppens et al.: Computer security: Proceedings of the 6th European Symposium on Research in Computer Security; October 4-6, 2000, Toulouse, France; Lecture Notes in Computer Science 1895, Springer-Verlag; ISBN 3-540-41031-7

Kai Rannenberg: Multilateral Security – A concept and examples for balanced security; Pp. 151-162 in: Proceedings of the 9th ACM New Security Paradigms Workshop 2000, September 19-21, 2000 Cork, Ireland; ACM Press; ISBN 1-58113-260-3

Forward Secure Mixes

George Danezis

University of Cambridge Computer Laboratory
William Gates Building, JJ Thomson Avenue
Cambridge CB3 0FD, United Kingdom
George.Danezis@cl.cam.ac.uk

Abstract. New threats such as compulsion to reveal logs, secret and private keys as well as to decrypt material are studied in the context of the security of mix networks. After a comparison of this new threat model with the traditional one, a new construction is introduced, the *fs-mix*, that minimizes the impact that such powers have on the security of the network, by using forward secure communication channels and key updating operation inside the mixes. A discussion about the forward security of these new proposals and some extensions is included.

Keywords: Anonymity, mixes, forward security, traffic analysis

1 Introduction

Research concerning anonymous communications has for a long time concentrated on protecting against traditional cryptographic threat models. These include passive and active attacks against network links, and a subset of compromised mix nodes [6]. Unfortunately these models fail to adequately capture the subtlety of the attacks that mix networks might be subject to, such as compulsion to reveal keys and logs, as well as being forced to decrypt seized material. The shortcoming of the traditional threat model has led engineers to produce solutions that are weak under these, quite realistic, threats.

In section 2 we analyze the traditional threat models and argue that they are still relevant in order to protect mix networks. In section 3 we present some new threats and attempt to complement the traditional threat model. We then explore how existing mixing technology copes with the new threats and how partial solutions can be implemented to protect against them, in section 4 and 5 respectively. We then present a generic solution protecting against the new attacks and analyze its security and how it could in practice be used (sections 6, 7 and 8).

2 Traditional Threat Models

In the context of anonymous communications the objective of the attacker is to discover the patterns of traffic of users in the network. A secondary goal of the attacker might be to disrupt and discredit the anonymous network.

The traditional threat model for mix networks has been based on the general threat model used in security and cryptology research. In this model attackers are first classified as *passive* or *active*. A passive attacker has the ability to monitor all network links and record their traffic. David Chaum demonstrated the security of mix networks against such an attacker [6]. In other studies the attacker is assumed to be less powerful and is only allowed to monitor some proportion of links [12]. An active attacker not only has the ability to monitor links but can also delay, delete, replay and inject new messages into the network. Such an attacker can perform replay and tagging attacks that some modern proposals for mix networks address [20, 14].

In most threat models against mix networks it is also assumed that the attacker controls a subset of the mix nodes. These compromised nodes can either simply monitor the traffic, therefore revealing the correspondence between their inputs and outputs to the adversary, or can actively misbehave in order to extract information out of the network or decrease its reliability. Distribution of trust by using many mixes is usually used to address the issue of information leakage, while solutions based on publicly verifiable proofs of correctness [1, 8, 18, 17] or reputation systems [9, 10] have been proposed to address the reliability issues. Different mix network topologies such as mix-cascades [4] as implemented in JAP [3], have also been explored to minimize the risk that corrupt nodes break the security of the network.

3 Beyond the Traditional Threat Models

The threats addressed in the traditional threat model are increasingly relevant to the proper functioning of mix networks. The small number of active remailer nodes [19] and the progress in interception technology and legislation, makes it possible for all mix nodes to have their traffic monitored. Common allegations of wide-spread interception of international telecommunications reinforce the importance of that threat [7]. Domestic interception of the content of communications is, at least in theory, regulated and warrants need to be sought before it can take place. Obtaining them takes time and effort, and a clear need has to be demonstrated. Of course this does not apply to monitoring by entities not belonging to the Law Enforcement community.

Active attacks are slightly more difficult to mount but given the current technology deployed [13], where only weak authentication is used between remailer nodes, they are quite possible to perform. There have not been any confirmed reports of compromised mix nodes, but since it would be very difficult to detect them the current state of uncertainty remains. In particular recent court cases in the United States have ruled as legal "hacking" into machines [22] in order to investigate criminal activities or gather evidence. It is not clear if that could extend to uncooperative mix nodes.

In addition to the threats above, that are largely speculative, remailer operators have experienced a series of new threats. The most common one is a subpoena requesting logs of traffic. Such logs do not usually exist in modern

systems, although the "anon@penet.fi" [16] remailer node had to stop operating after such an incident compromised the anonymity of one of its users.

Recently introduced U.K. legislation also allows Law Enforcement Agencies to request decryption of material and even private encryption or decryption keys [11]. This introduces the threat that honest mixes will have to collaborate by giving out some partial information to trace particular communications. All currently deployed or theoretical mix architectures are vulnerable to such an attack, particularly if they are requested to decrypt a message that has gone past them, or reveal the route of a single use reply block. Even worse the seizure of their keys would mean that their functioning would be completely transparent to outsiders. Of course such requests for decryption and keys can only be targeted and limited since both their cost is substantial and the authorization procedures are strict and time consuming.

Although interception of all network links is expensive and requires proper warrants in most jurisdictions, the interception of traffic data necessary to perform traffic analysis, does not have the same degree of legal protection. Recent legislative initiatives at international and national levels [5] make interception, access to and blanket retention of traffic data possible. Although the information present in the traffic data is much poorer than in the communication itself, mix network designers needs to keep in mind the ease of access to such data, and engineer their protocols so that they can resist opponents what have access to them.

A major assumption we will make, in trying to protect systems against the threats described above, is that with the exception of blanket traffic data retention, they have to be limited in breadth and in time. For example the attacker can only request the decryption of some material for some period of time. If the degree and the time of the attacks above was to be unlimited it would be extremely difficult to engineer a secure anonymizing protocol.

In addition to the assumed limits in time and breadth of the opponent powers we assume that the opponent does not have any special information in order to perform traffic selection, and therefore has to use information in the network to choose the victims of his targeted actions. As we will see, the techniques used to protect against the above threats rely of the attacker having to intercept or decrypt exponentially many messages relative to the number they are really interested in, or the number of remailers used.

4 Existing Networks and Designs

We will review the effect that the attacks above have on a traditional mix system, that in addition to providing a forward sender-anonymous channel also provides facilities for single use reply blocks (SURBs) as first described in [6] and used in [14]. Most of the constructions using zero-knowledge proofs [1, 8, 18, 17] are concerned with verifying correctness, therefore increasing robustness. They do not include features such as SURBs and are difficult to modify while retaining

their useful properties, therefore we shall not examine them in detail, although some of the techniques we present could be adapted and used by such schemes.

The object of our study is to determine how an opponent could use the powers described by the extended threat model in order to trace a sender-anonymous communication or to find out the originator of a reply block.

In the absence of any content interception, it is extremely difficult to use any decryption powers to trace back a sender-anonymous communication. Assuming that the mix network has a majority of honest nodes and that mixing has rendered traffic data useless, the recipient of the sender anonymous message does not have in their possession any bit-string that they could ask any node of the mix network to decrypt. All the routing information has been stripped and deleted by the time the message arrives. However an opponent interested in tracing communications sent by a particular user, only needs to put this specific user under surveillance and request all intermediate nodes used by messages to reveal the destinations of the messages.

When some content interception takes place the communication can at best be traced back, from its final recipient, to the point where the message was intercepted. Often there seems to be some confusion about the exact process that an opponent would need to follow in order to trace the communication. It is not the case that the opponent can start from the last hop and work his way backwards following the path of the message backwards, by compelling mixes to decrypt the messages. Such an exercise would require the opponent to provide mixes with the material to decrypt which by default he does not have. In order to trace back a communication it is required to work forward starting at the interception point, by requiring mixes to decrypt the material intercepted hoping that the communication traced ends up being the one that had to be traced. This process means that the opponent will need to acquire exponentially many decryption in the number of hops to potentially trace a message. Good anonymizing protocols would force this effort to be as large as to require all messages present in the mix network to be decrypted.

If one can assume that near ubiquitous content surveillance is in place, the procedure above becomes much more efficient. It is only necessary for the opponent to require the last hop to decrypt all communications that were intercepted until the message to be traced is decrypted. Then the procedure is repeated on the previous hops recursively. Different mixing strategies [21] may make such an attack slightly harder but it should in general be linear on the number of mixes used by the communication (although all content has to be intercepted). There is a trade off between the number of messages that have to be intercepted and stored versus the number of decryption requests that might be requested when tracing a message. Dynamic traffic selection could be employed to select traffic according to the likelihood that it corresponds to a message sent by, or to, a subject under surveillance. Without any additional information, performing such a task is equivalent to performing traffic analysis, and will require the interception of all material. In section 5.1 we will examine how super-encrypting communi-

cations between mixes with ephemeral keys renders content surveillance of the links useless.

As stated, the above methods for tracing communications concern sender-anonymous channels. In the case that a single use reply block has to be traced back to the destination it points, much more information is available to the attacker than in the sender-anonymous case. Reply blocks contain all the routing information needed to deliver the message encrypted under the private keys of intermediate mixes. Therefore the attacker can simply request the reply block to be decrypted iteratively by all the servers. This will take some time proportional to the number of mixes that the reply block contains. Tracing reply blocks does not directly affect the anonymity of other users and is therefore likely to be considered quite proportionate if a warrant is necessary. Note that there is no need for any content or traffic data interception to trace back SURBs.

5 Partial Solutions

While they do not address the full range of attacks described above some designs propose measures against some of them. In this section we will study some of these countermeasures and assess the degree to which they protect the network. In the next section we will integrate them along with additional countermeasures to provide generic protection against attacks based on compulsion to reveal keys, logs or decryption of material.

It is worth mentioning that the proposals below address the extended threat model but no countermeasure can be effective against an adversary that can force mix nodes to keep a detailed record of their internal workings. Indeed the solutions presented rely on the honest mix nodes being able to effectively forget key material or not even being able to access it, through the use of tamper resistant cryptographic hardware. An attacker that through legislative means or hacking techniques manages to record this material will therefore be able to defeat all countermeasures.

5.1 Forward Secure Link Encryption

A cheap way to render link level surveillance and recording of content fruitless for an opponent is to use a forward secure encrypted channel between mix nodes. Technically this invloves encrypted channels established using key exchanges with ephemeral public keys, signed with long term signing keys. The ephemeral keys are discarded immediately after a session key has been established. After each message is processed the session key is updated using some hash function, some exchanged fresh nonces and possibly the message content. The old keys and the nonces are systematically deleted after the update has taken place. This makes it impossible for nodes to decrypt past messages that are presented to them, since the keys used have been deleted. It is essential that fresh nonces are used during the key updating. This forces an adversary that at some point in time seizes the keys used by the mixes to observe indefinitely the channel to keep

his knowledge of the current keys up to date. Standard secure communication protocols such as SSL [2] support such forward secure modes of operation, and special purpose protocols such as JFK [23] are also available.

This technique renders interception at the link level useless, assuming that mixes have good ways of authenticating each other. If only a small number of mixes operate, this should not be a practical problem. The task of authenticating mixes could also be performed by the creator of the anonymous message, by including the hash of their verification keys in the anonymous routing data. An honest node should check that this hash matches the verification key during the key exchange protocol before establishing the encrypted channel. It also makes it difficult to perform active attacks such as inserting, delaying, deleting or modifying messages on the network links, that can be used to flood nodes to decrease the amount of anonymity they provide as described in [21].

Messages used to compel mixes into decrypting them can still be intercepted by malicious nodes and presented to the next honest mix in the chain. In order to detect malicious nodes the name of the previous mix could be contained in the headers of messages. In that way it is impossible to hide which node performed the interception or the previous decryption.

5.2 Public Key Cycling

It is best practice to change the encryption/decryption key pairs of the mixes often, and systematically destroy the old ones. This makes the decryption of messages encrypted under the old keys impossible.

The main problem arising with frequent changes of the public encryption keys is the short life of reply blocks. When a key changes, and is destroyed, it is impossible to route the SURBs encrypted under that key any more. Advertising future public keys in advance open mixes to the risk of having their future keys seized as well.

5.3 Secure Cryptographic Hardware

In order to minimize the risk of being compelled to surrender key material or decrypt intercepted material, one could implement the core functions of the mix inside a tamper proof hardware box. The sensitive functions including the decryption of messages and the replay prevention, would need to be performed by the secure co-processor in order to avoid compelled decryption. Assuming that the cryptographic co-processor is secure it should be extremely difficult to extract the secret key material inside it.

This construction protects against compulsion to reveal keys, but does not protect against corrupt mix owners, that want to trace the correspondence between inputs and outputs in the cryptographic module. In addition to the decryption and replay prevention, the secret permutation needs to be performed inside the cryptographic module in order to protect against such attacks.

6 Generic Solution: The *fs-mix*

Forward secure link encryption makes the product of interception useless, since if intermediate nodes forget the old keys, the material transmitted on the lines cannot be decrypted, but still allows corrupt mixes to intercept messages that could be decrypted to trace the communication. A more generic approach would be to make the decryption of the message impossible by honest mixes after a certain amount of time or after a certain event. We shall call this property *forward secure anonymity* and a mix that implements it a *forward secure mix* (*fs-mix*).

6.1 Presentation of the scheme

We can achieve forward secure anonymity by introducing state into the mix nodes. This is not a radically new requirement since most techniques implementing replay prevention already force mixes to keep hashes of the packets, or parts thereof, that have been processed in the past. The difference, as we will see, is that the state we require the mixes to keep in order to implement forward security needs to be kept secret since it will be part of the keying information.

In traditional mix systems [20, 15, 14] the address of the next mix ($A_{M_{n+1}}$) and the key used to decrypt the payload to be sent ($K\text{message}$) are included in the asymmetrically encrypted header, under the public key of the mix (Pk_n).

$$M_{n-1} \rightarrow M_n : \{K\text{message}, A_{M_{n+1}}\}_{Pk_n}, \{\text{Message}\}_{K\text{message}}$$

Traditional mix systems, such as mixmaster, store a packet ID that is used along with some integrity checking to avoid messages being processed more than once by the node [20]. In the case of [14] a special public hash N_{replay} of the symmetric key K, otherwise used to decrypt parts of the message, is kept in order to avoid replays of the same message. In case another message's secret hashes to the same value it is dropped.

$$N_{replay} := H_1(K\text{message})$$

We propose keeping a second secret hash of the symmetric secret (K_i) in a table indexed by a public value N_i:

$$K_i := H_2(K\text{message}) \text{ and } N_i := H_3(K_i)$$

When a message goes though the mixing network it leaves behind it a sequence of keys and their indexes on each of the nodes it went past. Future packets can use these keys, in conjunction with secrets they carry, in order to decrypt both their addressing information and their payload. So in practice a node would first decrypt the headers of a message using its private decryption keys and then read the index of the key to be used N_j, and retrieve the appropriate secret K_j. It would then compute a secret shared key K_{final} based on both the key K_j and the secret $K\text{message}$ contained in the decrypted header:

$$K_{final} := H_4(K_{\text{message}}, K_j), K_l := H_2(K_{\text{final}}) \text{ and } N_l := H_3(K_l)$$

$$M_{n-1} \rightarrow M_n : \{S_{\text{message}}, N_j\}_{Pk_n}, \{A_{M_{n+1}}, \text{Message}\}_{K_{\text{final}}}$$

In order to make it impossible for an attacker to force decryption of this message or to gain any information by accessing the list of secrets K some key updating operations need to be performed, to replace the old keys by new values:

$$K_j := H_5(K_{\text{final}}) \text{ and } N_j := H_3(K_j)$$

As soon as the old values (N_j, K_j) are replaced by the newly generated (N_l, K_l) they must be permanently deleted.

To summarize, the H_1 function is used to extract a public tag of the message to avoid replays and is always applied to the secret contained in the message. The function H_2 is applied to the final symmetric secret in order to generate keys that can be used in future communications, and the function H_3 is applied to these keys to generate their public indexes. The function H_4 is used on the secrets stored in the mix and the packet to create final shared secrets while the function H_5 is used to update secrets. All functions $H_x()$ are secure hash functions, in particular because they have to be pre-image resistant, to avoid giving any information about the key K_i associated with an index $N_i := H_3(K_i)$ or the relation between new and old keys $K_j := H_5(K_{\text{final}})$.

There must be a special index N_0 indicating that the message does not depend on any existing key present on the server, and the message should in that case be processed in the traditional way. If messages refer to a key index that does not exist they should be dropped.

6.2 The cost of an *fs-mix*

The properties provided by *fs-mixes*, as described above, are not achieved for free. The next section will explore the security advantages of this new schemes but first we will discuss the additional expenses both in terms of storage and processing.

A traditional mix only needs to maintain a public record of the ID of all the messages it has ever processed under its current private key. Therefore the storage required after processing n messages is $\mathcal{O}(n)$. For each new message processed a lookup is performed on the public record. Assuming that the lookup table is implemented using a hash table one can expect a cost of $\mathcal{O}(\log n)$ to perform each lookup.

An fs-mix stores more state than a traditional mix. It will need to store m pairs of (N, K) values. Unlike n, the number of messages processed by the current private key of the server, m is proportional to the number of messages that have ever been processed by the mix. In particular it will be maximum if the

forward secure functionality is never used by messages, since no pairs are ever deleted. As above the cost of finding a particular element should be proportional to $\mathcal{O}(\log m)$. In addition to the lookups as many as four hash operation might need to be performed per message.

In order to minimize the state needed by an fs-mix entries in the index and key table can be made to expire, either automatically or as requested by the sender. More details about this scheme will be presented in section 8.3.

7 Security Analysis

We shall argue that the modifications presented do not make the mix any weaker while providing additional security features. It is clear that if the keys K contained in every mix were public, an adversary would be exactly in the same position as in a traditional mix architecture. He would have to compel the mix to perform decryption using its private keys or surrender it, in order to trace any material that is in his possession. Therefore in that case the new scheme is equivalent to the traditional one.

7.1 Protection Against Compulsion

An attacker that tries to trace back a message that has already gone past an fs-mix cannot decrypt it unless all the messages upon which this communication's key depends were also intercepted and decrypted. Such an exercise would require all traffic to all the mixes to be logged and all of it to be decrypted in order to work, since the attacker does not have any a-priory knowledge of the message dependencies. The above is true for both sender-anonymous communications and single use reply blocks.

In case keys K are seized the first messages referring to them can be intercepted. The attacker then needs to be intercepting all subsequent messages in order to update his knowledge of keys to maintain his decryption capabilities. For each of these messages there must be a decryption request made to the mix (unless the private keys are seized).

Since not only the address is impossible to decrypt but the whole message, there is no way an opponent could try to use the body of the message in order to find out its destination. That means that even a single honest mix in a chain that implements a *forward secure anonymous* channel, that has genuinely deleted the used keys, is sufficient to order to provide forward security for the whole communication.

7.2 Traffic Analysis

Although the cryptographic security of the messages is stronger than in the traditional mixes, there is a new requirement imposed on the selection of routes that packets need to travel through. If there is a need for state to be present on

the mix from previous packets, that means that the same mixes are chosen repetitively in a non random manner. That selection process might provide enough information for the adversary to be able to link messages between them.

Since the number of nodes applying the proposed scheme does not need to be very large, and provided that there is a small number of mixes, the traffic analysis of the route should be hard. This is the case because a particular node has a large probability of being present in two routes anyway. In the case of a large peer-to-peer mix scheme, the security of such protocols against traffic analysis has to be re-evaluated.

Additionally the intermediate mixes are aware of the link between messages, since the only party that knows the keys stored is the party that has sent the previous messages. They are also aware of two other nodes on the path that are seeded by the same messages (the one preceding them and the one after them).

It is worth noting that the messages used to distribute keys are identical to otherwise normal messages, as far as any outside observer or compromised mix is concerned. This is an essential feature, that makes the traffic selection task of extracting messages that contain key material extremely hard for the adversary. In many cases the key trail left behind normal messages could be used as key material for further messages.

7.3 Robustness

If the anonymity of a network cannot be broken, an attacker might chose to degrade its performance, hopping that it will put people off form using it. Therefore a large body of work concentrates on proving the correctness of mix networks [1, 8, 18, 17]. Unfortunately, the requirement upon fs-mix nodes to store additional state is making the transport of messages more fragile and the nodes more prone to denial of service attacks.

Given that messages can get dropped at random, due to traffic congestion or network failures, making future messages reliant on past ones could potentially make the network even less reliable overall. Fortunately one could think of strategies in order to ensure the proper delivery of special, key distribution messages, before making further messages dependent on the keys they distributed. One way of doing so could be to create a message finally addressed to oneself and check for its delivery before using the secrets distributed to route further messages. Unfortunately there are security implications, namely the risk of having the key distribution message intercepted and traced, as well as all further messages linked to it.

8 Additional Features

In addition to the mechanisms described above, having facilities to make messages dependent on keys on the nodes could be used to implement *interdependent reply blocks* and *path burning messages.*

8.1 Interdependent Reply Blocks

Without modifications to the mechanisms described above it is possible to make many reply blocks depend on each other, in such a way that once one of then has been used the other ones cannot be traced back. To do this a common mix is used in the path of all the reply blocks, that has been given a particular secret by a previous message. The SURBs are constructed in such a way that they all can only be decrypted by a single secret entry on the shared mix node. As soon as the first of the messages routed using one of the interdependent SURBs uses the key, the key updating operation takes place, and the messages using the other SURBs automatically get dropped. Furthermore it is impossible to trace any of them back.

8.2 Path Burning Messages

Much in the same way as above, it is possible to make reply blocks valid for only a limited amount of time. After a determined time period a message is constructed that uses the same intermediate secrets as the SURBs, and is fired into the mix network. This message updates the keys, and any subsequent messages depending on them will get dropped. The same techniques can be used to make reply blocks valid only after a particular time period by only providing the necessary keys at some future time.

8.3 Automatic Key Expiry Date

By adding a time stamp to each of the keys one can make sure that the intermediate nodes automatically delete the keying material. This would provide the same functionality as the Path Burning Messages without requiring the principals willing to maintain their anonymity to actively delete keys. One could consider the encryption/decryption key pair rotation that mixes should perform periodically to be providing some automatic key expiry anyway, and requiring the stored keys to expire at the same time.

Alternatively the expiry time could be specified by the user along with the key. This has the disadvantage that it could be used for traffic analysis if any logic is used to calculate this expiry date, that takes into account the current time, or other information local to the user.

9 Conclusions

We have shown in this paper that the traditional threat model of active and passive attackers and corrupt nodes does not fully capture the variety of attacks that todays mixes could come under. In particular compulsion to reveal keys or decrypt material could be used to trace communications going through them.

By adding some additional state in the nodes we manage to limit in many cases the value that an opponent can extract using her compulsion powers, and allow the network to regain its security in the absence of continuous and ubiquitous surveillance.

Acknowledgments I would like to thank my supervisor Dr R. Anderson and my colleagues Richard Clayton and Andrei Serjantov for reading the manuscript and providing valuable feedback. This paper has also been improved by the comments of the anonymous referees.

References

1. Masayuki Abe. Universally verifiable MIX with verification work independent of the number of MIX servers. In *Advances in Cryptology - EUROCRYPT 1998, LNCS Vol. 1403*. Springer-Verlag, 1998.
2. Paul C. Kocher Alan O. Freier, Philip Karlton. The SSL Protocol Version 3.0. `http://home.netscape.com/eng/ssl3/draft302.txt`.
3. Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A system for anonymous and unobservable Internet access. In *Designing Privacy Enhancing Technologies, LNCS Vol. 2009*, pages 115–129. Springer-Verlag, 2000.
4. Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In *Designing Privacy Enhancing Technologies, LNCS Vol. 2009*, pages 30–45. Springer-Verlag, 2000. `http://www.tik.ee.ethz.ch/~weiler/lehre/netsec/Unterlagen/anon/disadvantages_berthold.pdf`.
5. Electronic Privacy Information Center. Data retention. Internet Web Resource. `http://www.epic.org/privacy/intl/data_retention.html`.
6. David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1982. `http://www.eskimo.com/~weidai/mix-net.txt`.
7. Temporary committee on the ECHELON interception system. Report on the echelon interception system. European Parliament Report A5-0264/2001 PAR1. `http://www.europarl.eu.int/committees/echelon_home.htm`.
8. Yvo Desmedt and Kaoru Kurosawa. How to break a practical MIX and design a new one. In *Advances in Cryptology - EUROCRYPT 2000, LNCS Vol. 1803*. Springer-Verlag, 2000. `http://citeseer.nj.nec.com/447709.html`.
9. Roger Dingledine, Michael J. Freedman, David Hopwood, and David Molnar. A Reputation System to Increase MIX-net Reliability. Proceedings of the Information Hiding Workshop 2001. `http://www.freehaven.net/papers.html`.
10. Roger Dingledine and Paul Syverson. Reliable MIX Cascade Networks through Reputation. Proceedings of Financial Cryptography 2002. `http://www.freehaven.net/papers.html`.
11. Foundation for Information Policy Research. Regulation of Investigatory Powers Information Centre. `http://www.fipr.org/rip/`.
12. Michael J. Freedman, Emil Sit, Josh Cates, and Robert Morris. Introducing Tarzan, a Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02)*, Cambridge, MA, March 2002.
13. Electronic Frontiers Georgia (EFGA). Anonymous remailer information. `http://anon.efga.org/Remailers/`.
14. Nick Mathewson George Danezis, Roger Dingledine and David Hopwood. Mixminion: Design of a Type III Anonymous Remailer Protocol. Manuscript. `http://seul.org/~arma/minion-design.ps`.

15. C. Gulcu and G. Tsudik. Mixing E-mail with Babel. In *Network and Distributed Security Symposium - NDSS '96*. IEEE, 1996. http://citeseer.nj.nec.com/2254.html.
16. Johan Helsingius. Anon.penet.fi is closed! http://www.penet.fi/.
17. Markus Jakobsson. Flash Mixing. In *Principles of Distributed Computing - PODC '99*. ACM, 1999. http://citeseer.nj.nec.com/jakobsson99flash.html.
18. M. Mitomo and K. Kurosawa. Attack for Flash MIX. In *Advances in Cryptology - ASIACRYPT 2000, LNCS Vol. 1976*. Springer-Verlag, 2000. http://citeseer.nj.nec.com/450148.html.
19. Christian Mock. Mixmaster stats (Austria). http://www.tahina.priv.at/~cm/stats/mlist2.html.
20. Ulf Möller and Lance Cottrell. Mixmaster Protocol — Version 2. Unfinished draft, January 2000. http://www.eskimo.com/~rowdenw/crypt/Mix/draft-moeller-mixmaster2-protocol-00.txt.
21. Andrei Serjantov, Roger Dingledine, and Paul Syverson. From a trickle to a flood: Active attacks on several mix types. Information Hiding Workshop 2002.
22. Bob Sullivan. FBI software cracks encryption wall. MSNBC. http://www.msnbc.com/news/660096.asp?cp1=1.
23. M. Blaze R. Canetti J. Ioannidis A.D. Keromytis W. Aiello, S.M. Bellovin and O. Reingold. Just Fast Keying (JFK). Network Working Group, Internet Draft, draft-ietf-ipsec-jfk-04.txt.

Continuous Opinion Polls on the Internet

Filip van Laenen
f.a.vanlaenen@ieee.org

Politiek.Net

Abstract. This paper presents a scheme to run continuous opinion polls on the internet. A continuous opinion poll is an opinion poll that allows voters to confirm or change their vote regularly within the same long term opinion poll. This makes it possible to track changes in the opinion of the voters. The proposed algorithm makes sure that on the one hand, one cannot find out who voted what, while at the other hand voters still can retrieve their last vote from the database and confirm it or change it after a freezing period.

Keywords Electronic Voting Schemes, Continuous Elections

1 Introduction

Many websites on the internet run opinion polls, ranging from traditional opinion polls about the voting intentions for the next parliamentary elections over polls about who is the most popular politician, media figure or sports player of the moment to polls about the next player that should leave some reality television show. Usually these opinion polls are discrete, that is, they open at a given point in time, and they last until they are closed. Participants are also supposed to vote only one time, so these polls don't accomodate for participants changing their minds. Some websites try to correct this by running the same poll over and over again.

The security mechanisms used by these opinion polls range from non-existing to fairly good. Some opinion polls do not even include the simplest mechanism to avoid that a voter votes two times. The average poll either places a cookie on the voter's computer to remember that a user on that computer already has participated in the poll, or they register the IP number from which somebody voted. The more professional websites require that a voter registers and logs on, and remember in their database who has already voted.

This paper presents a scheme that allows to run continuous opinion polls on the internet that has a few advantages over the opinion polls currently running on the internet. One advantage is that it can remember the votes of its participants, but because of security reasons, or rather as a precaution, it may be better to let the system forget the links between the votes and the voters when those links aren't needed any more. If a participant changes his mind, he can change his vote accordingly. However, in order to avoid too much turbulence in the results, a freezing period is used so that participants cannot change their votes too often.

Another feature of the scheme is that it allows votes to age, and become invalid after a certain period of time. Votes don't have to become invalid immediately after the freezing period, but they can loose their weight in the results gradually instead. This feature works as a smooting mechanism.

An important aspect of opinion polls is the privacy of the voters. Obviously, if certain privacy requirements aren't met, only very few people will want to register at a website and submit their opinion about political or other issues. This is why the votes should be protected such that other people than the voter himself cannot verify for whom or for which choice a voter voted. On the other hand, the scheme should also be robust enough such that voters who pretend to have forgotten their passwords cannot take advantage of the system and have a much larger impact on the results than other voters. But then again, people will forget or loose their passwords, and the scheme should allow for this to happen and be corrected in some way.

The next section will discuss the basic requirements and assumptions that lay at the basis of the voting scheme. The following section will take a look at how the data will be stored in the database. Section 4 will explain how voters can submit their votes, change their passwords, what happens when a participant forgets his password and how the votes will be counted. After that, the security of the scheme will be discussed by going through all the requirements that we put forward in section 2. Finally, a few words will be said about the implementation, including some extra security measures that are outside of the scope of this paper but nevertheless worth mentioning.

2 Requirements and Assumptions

Bruce Schneier [3] lists the following six requirements as the basic requirements for secure elections:

1. Only authorized voters should be allowed to vote.
2. No one can vote more than once.
3. No one can determine for whom anyone else voted.
4. No one can duplicate anyone else's vote.
5. No one can change anyone else's vote without being discovered.
6. Each voter can make sure that his vote has been taken into account in the final tabulation.

For our purposes, we will relax the second requirement to the following requirement:

2. There is an upper limit for how many valid votes a malicious voter can submit.

Since this is a continuous opinion, people should be allowed to change their mind over time, like Neville Holmes [1] described in his column. The concept of a freezing period during which a vote cannot be changed is copied, and it will later turn out that this helps to meet the second requirement in its relaxed form. The dual aspect of this is that votes should not be allowed to remain valid forever. This results in the following two additional requirements:

7. One has to be able to change his vote after a certain period.
8. Votes should become invalid after a certain period.

We can also make a few assumptions about the server that we will use to simplify the design of the algorithm. The assumptions are the following:

1. The server can be completely trusted.
2. The server is write-protected.
3. The server is sufficiently read-protected.

These assumptions are not so unrealistic for an opinion poll on the internet. The first assumption boils down to trusting the implementers of the opinion poll system and the people running the server. If they cannot be trusted, all the rest becomes worthless. The second assumption is a very basic one too: if the server isn't write-protected, there is general problem with the server. In the third assumption, the term "sufficiently read-protected" should be explained some more. In general, it should not be possible to read the raw data from the database. However, even if the complete database would be published, it should not be possible to break requirement 3. If it would be necessary to do so, fulfilling requirement 6 by publishing the complete database and the relevant implementation details should be an option. However, this should not be the default situation, but an exceptional one. Furthermore, we want to protect our participants against the eventuality that some entity (governmental or other) would obtain the database and therefore be able to read who voted for what. One could imagine that such an entity could obtain the database in a legal way, like for example through a court warrant, or in an illegal way by hacking the website and its database. But continuously monitoring the database should not be possible.

3 Overview over the Data in the Database

Before we describe the functionality to vote, to change passwords and to deal with lost passwords, this section will describe the data that has to be stored in the database. Notice that only the basic information that is necessary to organize a poll will be discussed here; a real implementation will probably store more information to make the system more user friendly and/or to implement additional security measures.

Table 1 shows the database table that stores the personal information of the participants. It may be necessary to store more information about the participants, like for example gender, for statistical reasons. Such information can be added to this table.

Notice that we use a hash function to store the password. We do not specify in this paper which hash function should be used to do this, nor which hash function should be used in the rest of the algorithm, but we leave it as an implementation issue.

The next table, shown in table 2, stores the information about the polls. Each poll has its own freezing and validity period, so that different polls in the same database can have different freezing and validity periods. The length of the time

Field name	Type	Comment
Id	ID	Unique identifier
Username	String	Unique user name
PasswordHash	String	Hash of the password and a salt
FirstName	String	Real first name
Surname	String	Real surname
Email	String	E-mail address

Table 1. Data stored for each voter.

frames too can be set per poll, such that more popular polls could have time frames that are shorter than less popular polls. Why we need time frames will be explained when we discuss the time stamps in the table with the votes.

Field name	Type	Comment
Id	ID	Unique identifier
Name	String	Name of the poll
Question	String	Question of the poll
Background	String	Background information about the poll
TimeFrameDays	Integer	The number of days per time frame
FreezingPeriodDays	Integer	The number of days for the freezing period
ValidityPeriodDays	Integer	The number of days for the validity period

Table 2. Data stored for each poll.

The possible choices that the voters can select for each poll are stored in another table, table 3, so that there is no limit on how many choices each poll can have. The order in which the choices will be presented to the user when he wants to vote should of course be randomized.

Field name	Type	Comment
Id	ID	Unique identifier
PollRef	IDReference	Reference to the poll
ChoiceText	String	Text for this choice

Table 3. Data stored for a choice for a poll.

When a user wants to vote for a poll, the system will create a voting letter connecting the voter and his vote with a poll. The voting letter stores only information about the fact that a voter has voted, and indicates when the last vote for this voting letter was submitted, how long the vote will be valid (if it isn't confirmed, replaced by another vote or withdrawn in the mean time), and when the freezing period will be over. A voter will not be allowed to vote for the same poll again as long as the freezing period registered on his voting letter

isn't over. The timestamps for the end of the freezing and the validity period are stored explicitly on the voting letter, and not deduced from the freezing and validity period of the poll and the timestamp of when the votes was cast, in order to avoid problems when the freezing and/or the validity period of the poll itself is changed. Table 4 describes the database table with the voting letters.

Field name	Type	Comment
Id	ID	Unique identifier
VoterRef	IDReference	Reference to the voter
PollRef	IDReference	Reference to the poll
ValidFrom	Timestamp	Timestamp indicating when the last vote was cast for this voting letter
FrozenUntil	Timestamp	Timestamp indicating the end of the freezing period for this voting letter
ValidUntil	Timestamp	Timestamp indicating the end of the validity period of the last vote cast for this voting letter

Table 4. Data stored for each voting letter.

Table 5 finally describes the data stored for each vote that is cast. Again, the timestamps for the end of the freezing and validity period are stored explicitly in this table. The choice is stored in plain text, which means that the votes can be counted with a simple database query. The connection with the voting letter and the voter is stored in the voting letter hash, which is the result of sending the user's password, the user's identifier, the poll's identifier and the timestamp of the voting letter through a hash function. Unless somebody has guessed a user's password, it should not be feasible to find out who cast this vote. Notice that this also puts restrictions on the resolution with which the timestamps will be created: it should not be possible to link a voting letter to a vote using the timestamps. The timestamps should therefore not be stored using millisecond precision, but with a precision of TimeFrameDays days, as defined for each poll. The length of one time frame depends on how many votes there will be cast per day: it should be so long that on average, "many" votes will be cast in every time frame, and in general always enough such that it becomes impossible to link votes to voting letters using the timestamps. One could also imagine a more adaptive algorithm that closes a time frame only when a sufficiently large amount of votes has been cast.

4 Functionality

This section describes the functionality of the system. In general, voters should be able to do the following things:

- Log on to the system with their user name and secret password
- Check the votes that they have submitted earlier
- Confirm, change or remove votes if the freezing period is over

Field name	Type	Comment
Id	ID	Unique identifier
Poll	IDReference	Reference to the poll for which this a vote
Choice	IDReference	Reference to the choice that was made
ValidFrom	Timestamp	Timestamp indicating when this vote was cast
FrozenUntil	Timestamp	Timestamp indicating the end of the freezing period for this vote
ValidUntil	Timestamp	Timestamp indication the end of the validity period for this vote
VotingLetterHash	String	Hash of the user's password, the user's identifier, the poll's identifier and the timestamp as it is stored in the voting letter indicating when the vote was cast

Table 5. Data stored for each vote cast.

– View the last results or the evolution of the results over time
– Join new polls.

When a vote leaves the freezing period, the system can send a reminder to the voter so that he can confirm, change or remove his vote.

The following sections describe some basic scenarios in detail. First we describe how a new vote can be submitted. Then we show how a user can change his password. The next section discusses what should be done in case a user says he has lost his password. We conclude with a short discussion about how the votes can be counted and a proposal for a weighting function for the votes.

4.1 Voting

Table 6[1] presents the algorithm to submit a vote. Now is a timestamp representing the date and time at the beginning of the current time frame. The Votes should not be overwritten so that historic data can be retrieved. As a precaution, the VotingLetterHashes in the Votes are nilled out when a vote becomes invalid, so that there aren't more hashes in the database than strictly necessary. The drawback of this is that a voter cannot retrieve his complete history from the database, but if that would be necessary, then the voting letters should be not be overwritten either.

4.2 Changing a Password

Changing a password has as a consequence that all the signatures on the votes for a user become incorrect. Therefore, when a user changes his password, all signatures on the votes should be updated. Table 7 shows how this can be done.

[1] In the description of the algorithms in this and the following tables, the symbol "|" denotes the concatenation of strings, and "$h()$" represents the hash function. The concatenation of strings is supposed to be unambiguous, which can be done e.g. by putting a special character between the arguments.

1. Input: Username, Password, PollId, ChoiceId
2. Using the Username, fetch the correct User row
3. Check for the User that h (Username | Password) = PasswordHash
4. Given the Id of the User and the Poll, fetch the correct VotingLetter row
5. Calculate from the VotingLetter row the VotingLetterHash s:

$$s \leftarrow h(\text{UserPassword}|\text{User.Id}|\text{Poll.Id}|\text{VotingLetter.ValidFrom})$$

6. Given the signature s, fetch the correct Vote row
7. If Vote.ValidUntil $>$ Now, then set the field Vote.ValidUntil to Now
8. Set the field Vote.VotingLetterHash to Nil
9. Create a new Vote with a new Id and the given PollId and ChoiceId
10. Set the field Vote.ValidFrom to Now
11. Set the field Vote.FrozenUntil to Now + Poll.FreezingPeriodDays days
12. Set the field Vote.ValidUntil to Now + Poll.ValidityPeriodDays days
13. Set the field VotingLetter.ValidFrom to Now
14. Set the field VotingLetter.FrozenUntil to Now + Poll.FreezingPeriodDays days
15. Set the field VotingLetter.ValidUntil to Now + Poll.ValidityPeriodDays days

Table 6. Algorithm to submit a vote.

1. Input: Username, OldPassword, NewPassword
2. Using the Username, fetch the correct User row
3. Check for the User that h (Username | OldPassword) = PasswordHash
4. Given the Id of the User, fetch all VotingLetter rows for this user
5. For each VotingLetter row do:
   (a) Calculate from the VotingLetter row the VotingLetterHash s_1 as follows:

$$s_1 \leftarrow h(\text{OldUserPassword}|\text{User.Id}|\text{Poll.Id}|\text{VotingLetter.ValidFrom})$$

   (b) Calculate from the VotingLetter row the new VotingLetterHash s_2 as follows:

$$s_2 \leftarrow h(\text{NewUserPassword}|\text{User.Id}|\text{Poll.Id}|\text{VotingLetter.ValidFrom})$$

   (c) Given the signature s_1, fetch the correct Vote row
   (d) Update Vote.Signature to s_2
6. Update User.PasswordHash to h (Username | NewPassword)

Table 7. Algorithm to update a password.

4.3 Lost Passwords

Since the passwords aren't stored in the database in plain text, the user cannot retrieve his password when he forgets it. However, we want to avoid that a user can pretend to have forgotten his password so that he can cast multiple votes for the same poll. The freezing period for each vote protects to a certain extent against this. In section 5 an upper limit for the number of valid votes and an upper limit of the weight of these votes will be calculated. Table 8 shows the algorithm for setting a new password when the user has forgotten his old password.

1. Input: Username, NewPassword
2. Using the Username, fetch the correct User row
3. Contact the user using his e-mail address, and send him a password change key
4. When the user contacts the system again with the correct password change key, update User.PasswordHash to h (Username | NewPassword)

Table 8. Algorithm dealing with lost passwords.

The user should be contacted before the new password is set to make sure that the user really requested his password to be changed without giving an old password.

4.4 Counting the Votes

The algorithm to count the votes is rather simple, since the votes are stored in plain text. However, since all historic votes are kept in the database, the validity period of the votes has to be considered, but it also allows to track the evolution of the opinion of a certain poll.

One can add a weighting function to the votes depending on how long ago they were cast. During the freezing period, the weight of a vote could be kept at 1, but after that the weight of a vote could gradually decay to become 0 at the end of the validity period. The following weighting function lets the weight decay linearily from 1 to 0 from the end of the freezing period (t_f) to the end of the validity period (t_v):

$$w(t) = \begin{cases} 1, & 0 \leq t \leq t_f \\ \frac{t_v - t}{t_v - t_f}, & t_f \leq t \leq t_v \\ 0, & t_v \leq t \end{cases} \quad (1)$$

As will be shown in the next section, a weighting function may reduce the influence of malicious voters in the system.

5 Security

This section will go through all the requirements from section 2 and verify to which extent they are met.

Only authorized voters should be allowed to vote This requirement is rather trivial in our scheme, since anybody can register in order to participate in the poll. If this would not be the case, an authorization scheme should be added. A few hints about this can be found in section 6 that discusses some implementation issues.

There is an upper limit for how many valid votes a malicious voter can submit As long as a user remembers his password, a user can vote only one time. There is however a problem when a user forgets his passwords, or pretends that he has forgotten his password. If the validity period is longer than the freezing period, a user can cast a second or even more more votes between the end of the freezing period and the end of the validity period. The maximum number of votes he can cast at any time is given by the number $\hat{n}$ calculated as follows:

$$\hat{n} = \left\lceil \frac{t_v}{t_f} \right\rceil$$

The weight of these votes can be reduced if a weighting function like the one defined in equation (1) is used. One can easily verify that in that case, the maximal total weight of the votes of one user is limited to $\hat{w}$:

$$\hat{w} = 1 + \frac{\hat{n}}{2}.$$

If the validity period is identical to the freezing period, only one valid vote can be cast at the same time, even if the user forgets or pretends to forget his password.

No one can determine for whom anyone else voted If the database is read-protected, this requirement is trivial for static situations. However, if the server is compromised, this requirement should still be met. Determining for whom anyone else voted is equivalent to doing one of the following things:

1. Guess from the timestamps on the votes and the voting letters which vote corresponds to which voting letter (which in its turn has a reference to the voter)
2. Guess the password of a user and retrieve his voting letter and then his vote
3. Guess the password and the user's identifier from a voting letter

The first problem can be avoided by having a large resolution for the timestamps, as was already discussed earlier. The second problem is equivalent to breaking the hash function used in the Voter table, and the third problem is equivalent to breaking the hash function used in the Vote table. It is therefore important that the hash function is a secure one.

Notice that if the server can be trusted not to log who votes for whom in another database table or a file, then not even the operators of the server would

be able to determine who voted for whom, unless they start to compare the historical states of the database with a higher time resolution than the time frames for a poll. This can in one term be described as a differential attack in the time, and this is also why we assumed that the server is "sufficiently" read-protected: it means that even though the database need not be read-protected the whole time, protection should be good enough to thwart these differential attacks by external people.

A differential attack in the time does not always require complete knowledge of the database to find out what a particular person has voted. Indeed, suppose Eve knows that Alice hasn't joined the opinion poll yet, and she wants to find out what Alice would vote for. She can start by checking the current results, and write them down. Next she sends a message to Alice inviting her to join the opinion poll. Eve could then start tracking the results of the opinion poll, and assume that the first change that occurs is the vote that Alice just submitted. Even though Eve cannot be completely sure that it really was Alice who submitted the new vote and that it was not one of the other voters changing his vote, or a visitor that happened to join the opinion poll right then, it would still be a very good guess if the opinion poll has not too many registered voters. To circumvent this problem, two things can be done:

- Present the current results with a lower than the highest precision, such that the changes of only a few votes within the current time frame cannot be observed.
- Never present the results of the current time frame, but only those from time frames that are already closed.

No one can duplicate anyone else's vote This should be enforced by the security on the server (write-protection). Notice that during a compromise of the server, votes could be duplicated with false VotingLetterHashes, and that the server will not be able to verify this unless all voters cooperate and remember their passwords.

No one can change anyone else's vote without being discovered This should be enforced by the software and by the security on the server (write-protection).

Each voter can make sure that his vote has been taken into account in the final tabulation In general, there will be no need for this requirement in a simple opinion poll on the internet, but this requirement can, if necessary, be met by publishing a snapshot of the database. Each voter can then verify that his vote has been included.

One has to be able to change his vote after a certain period A voter can change his vote when the freezing period on his voting letter is over. This has to be enforced by the software.

Votes should become invalid after a certain period The database table with the votes carries a field with the timestamp indicating the end of the validity period of the vote. However, if a voter makes a new submission, which may either be a withdrawal of his vote, a confirmation of the current vote or a change of the vote, the timestamp should be updated to the beginning of the curren time frame, so that the vote becomes immediately invalid. By doing so, each voter can have at the most only one valid vote in the database at any historic point in time.

6 Implementation

The voting scheme outlined in this paper is implemented on the political website [2], but it isn't operational yet. The website already had a user base, and each of the users will be offered to join the opinion polls. In addition to the security measures mentioned above, additional security measures had to be taken.

Functionality to avoid problems with cross-site scripting Cross-site scripting could be used to trick users that have registered on the website and that have checked the box to remember their password on their computer in a cookie to submit a particular vote without them knowing about it. It would probably be even better not to use long-term cookies at all, but this would degrade the user friendliness of the website too much.

Screening of the users that register Since all you need to join the opinion poll is an e-mail address, one could easily use multiple e-mail addresses to vote more than once. One of the countermeasures against this is for example to register for each voter which IP numbers he has used to submit a vote. Other countermeasures could involve proving your identity with a legal document, a valid social security number or a credit card number, or sending out physical tokens. How much energy is spent to screen the users depends on how important it is that no person can register twice or more.

Registration of how many times a user forgets his password If a user forgets his password too often, this could be a sign that he tries to submit more votes than he otherwise would be allowed to do.

Limiting access to the voting results Section 5 already discussed why the current results should not be presented at all or with a lower than highest precision because of security reasons. An additional limitation could be the following: Before a voter can see the results of a given poll, he has to submit a vote for that poll first. As a result, his vote will be frozen. If a voter is not forced to submit a vote first, he could take a look at the results before he submits his vote, and could therefore be influenced by these results when he later decides to submit a (new) vote. Of course, if a voter really wants to see the current results before he votes, he could always have a look at the results the last day before the freezing period of his vote is over and remember the results, but this

is probably much more trouble than what the average voter would be willing to go through to become influenced by the current results. Strictly speaking, this is not a security issue, but it's worth mentioning how the security mechanisms can be used to raise the quality of the voting results.

7 Conclusion

Originally, Neville Holmes described how legislative elections could be run in a sort of continuous mode as opposed to the current discrete mode and what the implications might be for democracy and society. This paper proposed a scheme to run continuous opinion polls on the internet that respects the privacy of the participants of the polls as long as the server is "sufficiently" read-protected. Although the scheme from this paper probably would need quite some changes before it can be used for real legislative elections, it may be very useful to start experimenting with it in the not too serious environment of a political website on the internet. Apart from that, it may also be useful in itself, because it has some clear advantages over the many opinion polls currently run on the internet. One of those advantages is that it is able to run opinion polls on a longer term while at the same time dealing with the fact that the participants may change their mind. This makes it possible to track the changes in the opinions of the participants as a group as a reaction to certain events or the policy of the government.

A key factor to the success of such a continuous opinion poll on the internet is the privacy of the participants and the security of the database. The latter is a general problem on the internet, and for the most outside the scope of this paper. A lot of the privacy of the participants, however, depends on how the votes will be stored in the database. The scheme in this paper takes care of this, while the solution remains sufficiently user-friendly.

If one day in the future a decision would be taken to go over to a more continuous election system, then more research is needed in two fields. The first field is that of the political sciences and sociology, because the transition to continous elections would represent a breakpoint with the current discrete elections. The other field is cryptography, because people will have to be able to trust the correctness of the results coming out of the database, and at the same time also trust that their privacy is still respected.

References

1. Neville Holmes, *Computer*, Representative Democracy and the Profession, IEEE, February 2002, pages 118-120
2. Dirk Laeremans and Filip van Laenen, `http://www.politiek.net`, 2002, Politiek.net
3. Bruce Schneier, *Applied Cryptography Second Edition: protocols, algorithms, and source code in C*, 1996, John Wiley & Sons

Privacy for Location Data in mobile networks

Alberto Escudero-Pascual[1], Thijs Holleboom[2], and Simone Fischer-Hübner[2]

[1] Royal Institute of Technology, IMIT, S 166 40 Stockholm, Sweden
aep@kth.se
[2] Karlstad University, Dept. Computer Science, S 651 88 Karlstad, Sweden
{Simone.Fischer-Huebner, Thijs.Holleboom}@kau.se

Abstract. The new EU Directive 2002/58/EC has introduced with its Art. 9 special protection for location data other than traffic data. In this paper, we argue that also location data within traffic data can contain sensitive information about the "relative positioning" and "co-located displacements" of mobile nodes and thus also requires special protection. After a brief introduction to how mobility is supported in IP networks, to the level of privacy protection for location data introduced in the new European Union data protection directive, and to means of protecting privacy by technology, we introduce the concept of *co-located displacements in MobileIP* and show how the home agent will be able to determine whether or not a set of mobile nodes move in a co-located fashion.
Finally, we present how privacy-enhancing technologies can be used to provide the level of privacy protection as required by Art. 9 of the EU Directive 2002/58/EC for location data other than traffic data, also for location information within traffic data.

Introduction

Location-based services (LBS) can be described as applications that exploit knowledge about where an information device (user) is located. For example, location information can be used to provide automobile drivers with optimal routes to a geographical destination or inform a group of friends when or where a friend is close in the neighborhood.

Traditionally security in computer networks include different aspects of message integrity, authentication, and confidentiality. However, in wireless networks, where users move between different networks and media types, another issue becomes equally important: location privacy.

This paper focuses on the situation where the absolute or relative position is computed in the infrastructure (i.e., home agent) in MobileIP-based networks.

The actual task of a home agent on a mobile node's home network is to tunnel datagrams for delivery to the mobile node when it is away from home (see section I.A). In the future, however, it might become more and more common that mobile nodes are also offering value-added services, such as LBS.

In these scenarios the user is not in full control of the location information associated with the mobile device. The problem arises when location information

is required in order to obtain a service and at the same time the user does not want to reveal more personal identifiable information than is strictly necessary for the provision of a concrete service. For example, a mobile user may want to inform to only a certain number of people for a certain period of time about his or her position or, to learn the position of the nearest catholic church without revealing his or her personal identity.

The paper is divided as follows:

Section 1 gives an introductory overview to mobility in IP networks, the European Union data protection directive concerning the processing of location data and to privacy-enhancing technologies useful for protecting location data, such as the Platform for Privacy Preferences Protocol (P3P) and *mix nets*.

Section 2 describes co-located displacements in MobileIP and proposes a formal method that allows a home agent to determine whether or not two mobile nodes move in a co-located fashion.

Section 3 explains how mix nets can be used to anonymise location information and how to use P3P to technically support the legal requirement of informed consent for the processing of location data within traffic data for value-added services.

1 Background

1.1 Mobility support in IP networks

The protocol operation defined for mobility in IP networks is known as MobileIP[1]. MobileIP allows a mobile node to move from one link to another in the Internet without changing the mobile node's home IP address. With MobileIP the mobile node can seamlessly roam among IP networks and media types without restarting any of the ongoing connections or associated applications. A mobile node is always addressable by its home address (HoA), an IP address assigned to the mobile node within its home subnet, i.e., with the network prefix of its home link.

MobileIP allows users to move between different networks, while maintaining an addressable static identifier (home address). This is done by associating a dynamic identifier (care-of-address, CoA) with the mobile node when it is away from home at a foreign link. All traffic to the mobile node is intercepted in the home network by a home agent (HA) that tunnels the data to the care-of-address that is in use in that moment. Packets may be routed to the mobile node using their home address regardless of the mobile node's current point of attachment to the Internet (CoA), and the mobile node may continue to communicate with other nodes after moving to a new link. With MobileIP the movement of a mobile node away from its home link is thus transparent to transport and higher-layer protocols and applications.

MobileIPv6 shares many features with MobileIPv4, but the protocol is now fully integrated into IPv6. As in MobileIPv4 the mobile mode is responsible for discovering its current location. When the mobile node is attached to its home

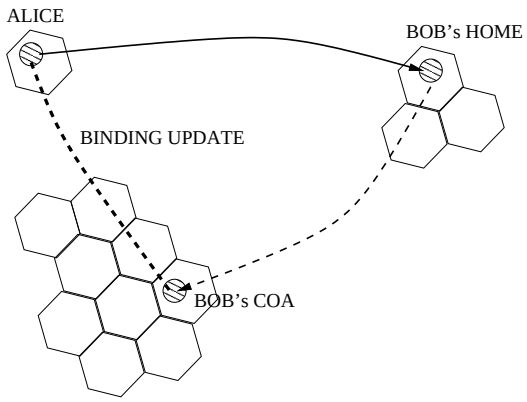

Fig. 1. Route Optimization in MobileIPv6.

link it directly receives packets and when roaming in a foreign network, it must acquire a co-located care of address and notify its home agent of this address.

MobileIPv6 on the other hand also includes the mechanisms that allows the mobile node to inform selected IPv6 correspondent nodes (CN) of its care-of-address, so packets from these correspondent hosts can be redirected straight to the mobile node instead of using the home agent as an intermediary (route optimization) [Fig .1].

1.2 European Union Directive on privacy in electronic communication

On July 12, 2002 the EU Directive 2002/58/EC concerning 'processing of personal data and protection of privacy in the electronic communication sector' [2] was officially adopted. The new Directive is part of a package of initiatives which will form the future regulatory framework for electronic communications networks and services. It aims to adapt and update the existing Data Protection Telecommunications Directive (97/66/EC) [3] to take account of technological developments.

The new EU Directive 2002/58/EC establishes a common framework for data protection in telecommunication services and networks regardless of the technology in use in electronic communication services and networks.

Whereas in the Directive 97/66/EC traffic data only refers to "calls" in so-called circuit switched connections (traditional voice telephony or plain old telephone system aka. POTS), the new EU Directive 2002/58/EC covers all traffic data in a technology neutral way including Internet traffic data.

Traditionally content data has had a high level of privacy protection, and it has been acknowledged that strict privacy requirements for location data other than for traffic data are needed, as they enable exact positioning and hence a permanent surveillance of users.

While it is questionable if the traditional classification of the data in traffic, content and location can be applied to the Internet [4], in contrast to the Directive 97/66/EC, in the EU Directive 2002/58/EC in Art.5 *(Confidentiality of the communication)*, traffic data has been added. Hence, at least according to the new Directive, traffic data is supposed to have the same level of privacy protection as content data. Thus EU Directive 2002/58/EC is thereby acknowledging that traffic data needs the same level as protection as content data.

The EU Directive 2002/58/EC differentiates between location data other than traffic data, allowing the exact positioning of a mobile user's device, and location data within traffic data, giving geographic information that is often less precise. If used for value-added services, location data other than traffic data has a higher protection. Whereas for traffic data informed consent is required (Art. 6 par. 3,4), for location data other than traffic data either anonymisation or informed consent is required (Art. 9 par.1) with the possibility for users that have given their consent to temporarily refuse the processing for each connection or transmission of a communication (Art.9 par.2).

In this paper, we argue that also traffic data can contain sensitive information about the *relative positioning* and *co-located displacements* of two mobile nodes, and thus also needs a high level of privacy protection.

According to the principle of data minimization and avoidance derived from the principle of necessity of data collection and processing, location data, no matter whether within traffic data or other than traffic data, should be anonymised if the effort involved is reasonable in relation to the desired level of protection. Also for location data within traffic data, users that have given their consent should have the possibility to "revoke" their consent for each connection or transmission.

1.3 Privacy-Enhancing Technologies

There are basically two major ways of enhancing privacy in the mobile Internet by technology.

Privacy can be protected most effectively by the first group of privacy technologies that avoid or at least minimize personal data that are exposed on the communication lines and at network sites, and are thus providing anonymity, pseudonymity, unlinkability or unobservability. Mix nets are examples for effective privacy technologies for anonymising communication.

The second way to protect privacy is by using technologies that can control that personal data are only used according to legal provisions. P3P [5] for example, is a technology which can provide technical support for implementing that personal data is only forwarded to web sites with the user's informed consent. According to data protection legislation, informed user consent is often required for the legitimacy of data processing.

Mix networks David Chaum described in [6] a technique based on public key cryptography that allows an electronic mail system to hide who a participant communicates with as well as the content of the communication.

More generally, messages are exchanged through a chain of one or more intermediaries called *"mixes"*. The purpose of a mix is to hide the correspondences between the items in its input and those in its output. The main function of a mix is to: receive and decrypt messages, buffer messages until a defined number of messages has been received, change the sequence of the received messages in a random manner and encrypt and forward the messages to the next mix or to the receiver.

Platform for Privacy Preference (P3P) Protocol The Platform for Privacy Preferences (P3P) Protocol, which has become an official W3C recommendation in April 2002, enables web sites to express their privacy policies in a machine-readable XML format that can be retrieved automatically, interpreted easily and compared with the user's privacy preferences by user agents. Thus, it enables users/ user agents to come to a semi-automated agreement with web sites about the privacy practices for personal data processing by that sites.

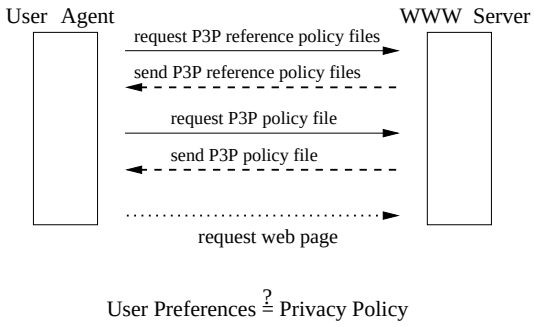

Fig. 2. P3P for informed consent

How a P3P agreement is done is described in [5] and depicted in [Fig. 2]. The P3P user-agent will typically, when an HTTP request is made, fetch a reference file, which is a site map, matching policy files with pages, parts of the site or the whole site, and is typically stored at a well-known location at a website, *"/w3c/p3p.xml"*. According to this reference file, the appropriate policy file will be retrieved, and matched against the user's preferences. If there is a match, the page will be requested, and if not, the user-agent will take some kind of action to warn the user.

2 Co-located displacements

As explained in [Sect. 1] the home agent of a mobile node keeps track of the binding between the home address (HoA) and the mobile node's care-of-address (CoA) and is fully aware of the network prefix of the link to which a mobile node is attached to. This prefix carries information about the geographical position of the mobile node. Even though a prefix cannot generally be converted into an exact geographical position it will usually confine the possible values of the geographical position to an area that is small in comparison to the area of the surface of the earth.

It also is conceivable that knowledge of the prefix confines the possible positions to a limited area without being able to exactly locate that given area. In that case, however, it will still be possible to determine for two mobile nodes whether or not they are located in the *same* limited area. In other words, for two mobile nodes that use the same home agent, that home agent is aware of possible proximity, and hence the relative positioning, of two mobile nodes.

What is more, however, is that since the care-of address is a function of time, the home agent is able to record at which instance of time a mobile node moves its point of attachment from one foreign link to a new, different, foreign link. That also means that the home agent is able to determine whether or not two mobile nodes move in a co-located fashion. We consider that two mobile nodes that have the same home agent *move in a co-located fashion* if they change to a new care-of-address in the same foreign links a number of times *simultaneously.* An example of such movement is two people traveling in the same car or train. A sketch of how the prefixes of the care-of addresses of two such nodes change as a function of time is given in [Fig. 3]. Note that two nodes generally not change prefix at *exactly* the same time due to the nature of the events that trigger the mobility handovers (signal/noise ratio, network latency etc).

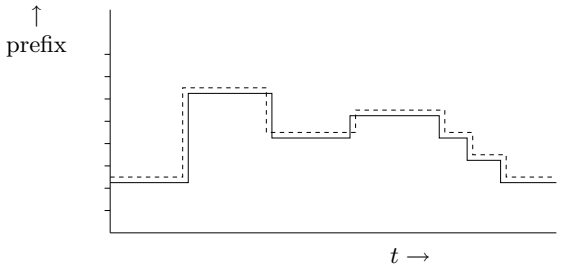

Fig. 3. *Sketch of prefixes of care-of addresses as a function of time. Values, i.e., prefixes, that fall in the same slot on the vertical axis are identical. line:* $CoA^{(i)}(t)$*, dashed line:* $CoA^{(j)}(t)$

The care-of address as such only determines the geographical position of a mobile node up to the area covered by the foreign link associated with that

specific care-of address. Movement of a mobile node while connected to the same foreign link will be undetected by the home agent. The care-of addresses of two mobile nodes will allow the home agent to determine the distance between two mobile nodes with an accuracy of approximately the size of the area covered by the foreign link. However, by studying the dynamics of the care-of addresses, it is possible to obtain a more accurate picture of the actual movements of these two mobile nodes. If two mobile nodes change their care-of addresses at almost the same time it is likely that their actual geographical distance was small at the time of change. If this happens a few times in a row it is likely that these nodes were also close at intermediate times. Hence, by studying the *dynamics* of the care-of addresses of two mobile nodes it is possible to extract information about the geographical distance of these two mobile nodes.

2.1 Analysis of co-located displacements

The care-of address, as a function of time, of a mobile node i will be denoted as $CoA^{(i)}(t)$. For two mobile nodes, i and j, consider the function

$$\Gamma(CoA^{(i)}(t), CoA^{(j)}(t)) = \begin{cases} 1 & \text{prefix } CoA^{(i)}(t) = \text{prefix } CoA^{(j)}(t) \\ 0 & \text{otherwise} \end{cases}$$

Then the integral

$$T = \int_{t_1}^{t_2} dt \Gamma(CoA^{(i)}(t), CoA^{(j)}(t)) \tag{1}$$

gives the time, within the interval $[t_1, t_2]$, that the nodes i and j were co-located. This could equivalently be expressed as a percentage $p = T*100/(t_2 - t_1)$. The measure T does provide information about the *duration* of co-location, irrespective of the *movements* of the nodes i and j. As a consequence two nodes that are connected to the same foreign link during the whole interval $[t_1, t_2]$ will produce $p = 100$, like two nodes that are not static but *move* in a co-located fashion, i.e., simultaneously change to the same new foreign link any number of times. As mentioned above, two roaming nodes will in reality never change prefix at exactly the same instance of time. Such nodes will therefore always produce a number p slightly less than 100 %, even if they move in a fully co-located fashion.

The function $\Gamma(CoA^{(i)}(t), CoA^{(j)}(t))$ implicitly depends on time through the care-of addresses and can also be denoted $\Gamma(t)$. In figure 4 this function is sketched. The shaded area gives the duration of co-location. Small periods of non-co-location that arise when two nodes change prefix only marginally affect the total area, being equivalent to the integral in equation 1.

In order to be able to distinguish co-located roaming from static co-location, i.e. non moving nodes connected to the same link, one can simply count the number of hops where two mobile nodes simultaneously change their care-of

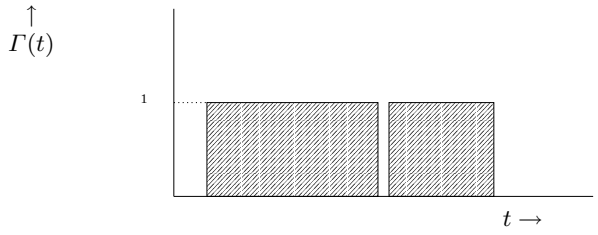

Fig. 4. *The area under $\Gamma(t)$ as a measure of co-location*

address prefixes to the same new value, again within a certain time interval $[t_1, t_2]$. This number, H, is then a functional of the care-of addresses $CoA^{(i)}(t)$, and $CoA^{(j)}(t)$, which in turn are functions of time, and has the following functional form

$$H = h\left[CoA^{(i)}(t), CoA^{(j)}(t), t_1, t_2\right] \tag{2}$$

The two explicit time arguments t_1 and t_2, indicate the boundaries of the time interval under consideration. H can easily be calculated by analyzing the functions $CoA^{(i)}(t)$ and $CoA^{(j)}(t)$, which in turn can be done by analyzing logged data at, for example, the home agent. Since, as pointed out above, two roaming nodes never change prefix at exactly the same time, it is necessary to use an interval Δt. Two nodes that change prefix of care-of-address at times t and t', where $|t - t'| < \Delta t$ are considered to have changed simultaneously. The interval Δt should be small, at least in comparison to the duration of the entire measurement $t_2 - t_1$.

In summary, the number of simultaneous hops H, of two nodes i and j, can be extracted from logged data by finding all instances where i and j change prefix at times t and t' separated by less than some predetermined amount Δt. Only hops where both the 'old', and the 'new' prefixes are the same should be counted, since otherwise there is no co-location. More formally H can be calculated as follows. If the care-of address $CoA^{(i)}(t)$ changes at times $t_k, k = 1 \cdots n$ then define

$$h_k = \begin{cases} 1 & CoA^{(j)}(t) \text{ shows identical change at } \ t = t', |t' - t_k| < \Delta t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Now the quantity H defined in equation 2 can be expressed as the sum

$$H = \sum_{k=1}^{n} h_k \tag{4}$$

3 Privacy-enhancing technologies for protecting location data

In this section, we will discuss how privacy-enhancing technologies can be applied to technically support and enforce legal privacy requirements of Art. 9 of the EU Directive 2002/58/EC for location data, no matter whether location data other than traffic data or within traffic data.

3.1 Mix nets for anonymisation of location data

The mix network concept was implemented as part of the Freedom System [7,8]. Freedom is a pseudonymous IP network that provides privacy protection by hiding the user's real IP addresses, email addresses, and other personal identifying information from communication partners and eavesdroppers.

The Freedom System could be seen as an overlay network composed of globally distributed servers that runs on top of the Internet. Freedom routers or Anonymous Internet Proxies (AIP) are the core network privacy daemons and they are in charge of passing encapsulated packets between themselves until they reach an exit node or AIP wormhole. When a certain AIP runs as an exit node, it works as a traditional network address translator.

Symmetric link encryption is applied between AIP pairs and the freedom-client and the selected AIP entry point to hide the nature and characteristics of the traffic between them. Once the route is created from the freedom client to the wormhole, the data packets travel toward the wormhole over the virtual circuit, being link decrypted, telescope unwrapped and finally link encrypted at each point. The data is routed to the next hop by use of an Anonymous Circuit Identifier mapping table. The ACIs indicate, along with a packet's implicit source address and port, the next hop in a particular route.

When a freedom client communicates with a correspondent node via a previously built virtual circuit in the Freedom System, the correspondent node sees that the traffic as coming from the wormhole IP address instead of the client's real IP address.

In [9] we introduced a set of protocol extensions to the Freedom System architecture to permit a mobile node to seamlessly roam among IP subnetworks and media types while remaining untraceable and pseudonymous. The extensions make it possible to support transparency above the IP layer, including the maintenance of active connections in the same way that MobileIP does but with the addition that the home and foreign network are unlinkable [Fig. 5].

The concept of a mix network for location based services was also introduced in [10] where a Privacy Enhanced-Location Based Services (PE-LBS) proxy can be configured to act as a "mix" by buffering and changing the sequence of the service requests. The mobile device can use a chain of PE-LBS proxies configured as a "mixing network" to forward a location based service request. The architecture allows a mobile node to request location based services via a mix-network hiding the network location of the mobile device while providing service accountability.

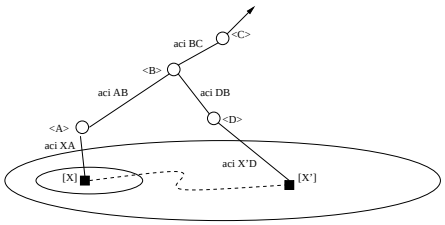

Fig. 5. Mobility extensions for the Freedom System. The virtual circuit is partially recreated during a vertical handover $[X] \rightarrow [X^{'}]$. The exit node $< C >$ is not aware of any mobility.

3.2 P3P and processing of location data in MobileIP

The Platform for Privacy Preferences (P3P) Protocol can be used as a technical means for technically supporting the privacy principle of informed consent, and also for allowing users to later revoke their consent. Although P3P is a standard for controlling the personal data processing by web servers, we will discuss how it could also be used for obtaining informed consent for the processing of location data within traffic data by home agents for value-added services, such as location based services.

In the future, more effective solutions for automated privacy agreements between mobile nodes and home agents could be based on compact privacy policy or preference information included in newly to be defined extension headers for Mobile IPv6. Such a solution, however, will first require new protocol extensions, whereas a solution based on P3P can easily be implemented already today with available technologies.

Often there is a close administrative relationship between the owner of a mobile node and the owner of that mobile node's home agent. For example, a company that provides mobile nodes for its employees is also operating home agents for those mobile nodes, or a home agent could even be operated by the mobile user. If there is a trust relation between the mobile user and the owner of the home agent, an agreement about data processing practices do not have to be automated but can as well be done off-line. However, home agents could also be owned by a service provider or other organizations to which no close trust relation exists. Besides, if we demand the same level of privacy protection for location data within traffic data as for location data other than traffic data, mobile users should still have the possibility to revoke their consent for the processing of location information within traffic data for each connection or transmission of a communication (as required by Art. 9 par. 2 for location data other than traffic data). Also for the enforcement of this requirement, an online solution as provided by P3P is required.

For enabling P3P agreements a home agent needs to have a web server interface and has to have a P3P privacy policy containing a statement describing data practices that are applied to location data. In order to be compliant with the EU Directive 2002/58/EC, location data within traffic data should only

```
<STATEMENT>
         <PURPOSE>
               <current/>
               <other-purposes required = "opt-in">
                Location-based-Services
               </other-purpose>
        </PURPOSE>
         <RECIPIENT>
              <ours/>
         </RECIPIENT>
         <RETENTION>
               <no-retention/>
         </RETENTION>
        <DATAGROUP>
               <DATA> ref="#dynamic.miscdata">
                      <CATEGORIES>
                            <location/>
                      </CATEGORIES>
               </DATA>
         </DATAGROUP>
</STATEMENT>
```

Fig. 6. P3P policy statement for a EU Directive 2002/58/EC compliant processing of location data.

be processed for the transmission of a communication (Art. 6 par. 1) or for marketing its own electronic communication services or for the provision of value-added services, such as location-based services (Art. 6 par.3). [Fig. 6] shows an example P3P policy statement for location data, allowing its processing for the current purpose of transmitting a communication and for location-based services. The opt-in requirement should be used to state that location data can only be processed for location-based services if the user explicitly requests that service and thus gives his/her consent for the use of location data for location-based services. At a policy's "opturi" link, instructions are provided for users how to decline from their request. Hence, the home agent could set up a web site that allows mobile users to fill-in forms for granting or revoking their consent for the processing of location data for the specified value-added services. This guarantees that also Art.9 par.2 can be technically supported.

Within the P3P policy statement, the RECIPIENT element should be set to <ours/>, meaning that the location data is only handled by the Home Agent or possibly entities processing the data on its behalf for the completion of the value-added service, as required by Art. 6 par.4 and Art. 9 par.3.

The RETENTION element that indicates the kind of retention policy that applies to the data should be set to <no-retention/> or <stated-purpose/> to state that the data are only processed for the duration necessary for the value-added service as required by Art. 6 par.3 and Art.9 par.1.

The mobile node has to have a P3P-compliant user agent including P3P privacy preferences defined by its user for the processing of location data. By accessing the Home Agent's web site, the mobile user can check the Home agent's privacy policy for processing of location data and can fill-in a form for requesting a value-added service, and for thereby giving his/her informed consent for the processing of traffic data for that service. A mobile user should evaluate the home agent's privacy policy at the time that she/he chooses a mobile node, and should reevaluate it before the expiry period of the policy file has passed, or in case that the mobile user has changed his/her preferences or wants to revoke his/her consent.

A problem, however, is that location information is included in all messages sent by a mobile node to its home agent. Thus, when the P3P user agent of a mobile node is fetching the P3P reference file and the policy file, it is already transferring location information with those requests, even though there has not been a successful P3P agreement with that agent yet. The home agent's web site should hence follow the so-called safe-zone practices for communications which take place as part of fetching a P3P policy or policy reference file, and thus should not collect location information that is available within the safe zone. If a user does not want to rely on the safe-zone practices, she/he should preferably initiate P3P negotiations at times that her/his node is located in its home network. If a mobile user does not succeed to select a home agent that fulfills her/his privacy preferences, she/he should have the option to use anonymous communication.

P3P has been criticized by the Art.29 Data Protection Working Party [11,12] and others, as it cannot in itself secure privacy on the Web. Hence it needs to be applied according to a regulatory framework, such as given by EU Directive 2002/58/EC. Besides, P3P cannot ensure that web sites really follow privacy policies as they claim to do. Third party monitoring can enhance control over the compliance with the privacy policies published at web sites.

4 Conclusions

In this paper, we have shown that traffic data in MobileIP-based networks can also contain sensitive information about the relative positioning and co-located displacements of two mobile nodes, and thus also needs high level of privacy protection. By studying the dynamics of the care-of addresses of two mobile nodes it is possible to extract information about the geographical distance of these two mobile nodes. The co-located displacements in MobileIP allow to the home agent to determine whether or not a set of mobile nodes move in co-located fashion.

Privacy-enhancing technologies should be applied to technically enforce legal privacy requirements of Art. 9 of the EU Directive 2002/58/EC for location data, no matter whether location data other than traffic data or within traffic data.

According to the privacy principle of data minimization and data avoidance, location data should be anonymized if the effect involved is reasonable in relation to the desired effect. Mix-nets based architectures (as presented in section 3.1)

provide an effective means for anonymising location data, and should preferably be provided to mobile users in order to fulfill the requirements of Art. 9 par. 1. We have also shown how the Platform for Privacy Preferences (P3P) Protocol can be used as an Online mechanism for obtaining informed consent of mobile users for the use of location data for value-added services, and also for allowing users to later revoke their consent, as required by Art. 9 par.1 and par.2. Hence, we have shown how existing technologies can be used to provide different levels of location privacy, even though we strongly propose that the future developments in the next generation Internet Protocol should also directly include features for location privacy.

References

1. C. Perkins, IP Mobility Support, RFC 2002, October 1996.
2. Directive 2002/58/EC of the European Parliament and of the Council concerning the processing of personal data and the protection of privacy in the electronic communications sector, Brussels, 12 July 2002, http://www.etsi.org/public-interest/Documents/Directives/Standardization/Data_Privacy_Directive.pdf
3. European Union, Directive 97/66/EC of the European Parliament and of the Council Concerning the Processing of Personal Data and the Protection of Privacy in the Telecommunications Sector of 15 December 1997. http://europa.eu.int/ISPO/infosoc/telecompolicy/en/9766en.pdf
4. A. Escudero, Location Privacy in mobile Internet in the context of the European Union Data Protection Policy. Proceedings of INET2002. Washington DC. June 2002.
5. The Platform for Privacy Preferences 1.0 (P3P1.0) Specification, W3C Recommendation, 16 April 2002, http://www.w3.org/TR/P3P/
6. D. Chaum, Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Communications of the ACM (24)2, 1981.
7. I. Goldberg, and A. Shostack, Freedom Network 1.0 Architecture and Protocols. 1999.
8. A. Escudero, M. Hedenfalk, and P. Heselius, Flying Freedom: Location Privacy in Mobile Internetworking. INET2001. Stockholm. June 2001.
9. A. Escudero, Anonymous and Untraceable Communications: Location Privacy in Mobile Internetworking. Licentiate Thesis ISSN 1403-5288. May 2001.
10. A. Escudero, and G. Q. Maguire Jr, Role(s) of proxy in Location Based Services. Proceedings of 13th IEEE International Symposium IEEE on Personal, Indoors and Mobile Radio Communications, Vol.3 pp 1252-1257, Lisbon. September 2002.
11. Working Party on the Protection of Individuals with regard to processing of Personal Data, Opinion 1/98, Platform for Privacy Preferences (P3P) and the Open Profiling Standard (OPS), adopted on 16 June 1998, http://europa.eu.int/comm/internal_market/en/dataprot/wpdocs/wp11en.htm
12. Article 29 - Data Protection Working Party, "Privacy on the Internet - An integrated EU approach to Online Data Protection", adopted on 21st November 2000, http://europa.eu.int/comm/internal_market/en/dataprot/wpdocs/wp37en.pdf

Attitudes Towards Privacy in Conjunction with Location Based Services

Christian Bratsberg Hauknes
Department of Informatics
University of Oslo
chrisbh@ifi.uio.no

Abstract.

Location Based Services (LBS), and many other new technologies and services have a privacy destroying potential. As well as protecting privacy with technology and legal means, we also need to design systems, services and interfaces that fit the user's privacy needs and attitudes. In this work, I look at how this can be achieved for new technologies and services in general and LBS in particular.

The concept of privacy is investigated, and it is argued that a broader approach to privacy is needed. A system can affect privacy in more ways than just being a new channel of information; it can also affect privacy by changing behavior, communication and expectations. This also has to be considered when designing privacy solutions and interfaces.

Privacy implications of a system are divided into three areas; User vs. Surroundings, User vs. Commercial Interest and User vs. Authorities (Surveillance). The latter, although extremely important, is not discussed here. For each area there we need to consider how a user under 'normal' circumstances regulates the flow of information, and how he protects his privacy. We then have to observe how the system might affect this, and design solutions that let the user achieve an acceptable privacy and comfort level. The importance of observing the user in context to achieve valuable data is emphasized.

In the User vs. Surroundings area a simple system delivering LBS to students was developed and tested, with a few services, notably a 'Friend Finder' service. In interviews done prior to testing the system, users expressed the need for regulating who could access their location information, and a 'blocking' function. This was found to be insufficient, as blocking was experienced as socially unacceptable and rude. The system was also found make mobility more structured, and change expectations of availability. Although mostly unproblematic, there were some specific contexts where this made users feel uncomfortable.

Being able to communicate contextual information might solve some of these problems, but there were instances where users wanted to be able to display erroneous information, and maybe blame the system. This problem and need might be even more profound in other contexts (not just a student setting).

In the User vs. Commercial Interest area, the 'normal' flow of information is quite different. Comparisons is made to Internet surfing and online purchases. An interface was designed based on P3P for a simulation of a Map-service. Privacy critics would pop up and warn the user if there was a mismatch between his privacy preferences and the service provider's privacy policy. A case study was done, communicating with the interface screenshots on sheets.

Responses to the system were very good, with users expressing a feeling of comfort, understanding and control. Follow-up interviews did however show low awareness and understanding of their own privacy behavior and the privacy critics. This in general caused little concern with the users, and did generally not change their feeling of comfort or control. It might be beneficial to discuss what we want to achieve with the privacy interfaces.

Tracing the underlying reasons for security breaches – A method based on cognitive interviews

Lars Westerdahl, Arne Vidström, Jonas Hallberg, Amund Hunstad and Niklas Hallberg

Swedish Defence Research Agency, Dept. of Systems Analysis and IT Security, Box 1165, SE-581 11 Linköping, Sweden
{lars.westerdahl, arne.vidstrom, jonas.hallberg, amund.hunstad, niklas.hallberg}@foi.se

Abstract. There are numerous strategies to improve the security of information systems. Some of those are focused on pre-operation, such as product evaluation and risk analysis, whereas others try to improve existing systems, for instance red teaming. To the recognitions of the authors, so far little or no effort has been made to discover the real reasons behind security vulnerabilities. Considering a specific security vulnerability, the important question is *what is the core reason for its occurrence?* In this paper, a method for tracing the underlying reasons for security vulnerabilities in IT systems is proposed. The focus of the method is primarily non-technical since the interest is concentrated to the production, operation and maintenance of existing systems. Thus the main focus is on the people behind the decisions that affect the correctness and level of security of the resulting systems. As a consequence, the main approach of the method is to conduct interviews in order to discover facts usually not documented in manuals or system logs. The proposed method is still in its infancy and needs to be further studied and improved. However, a literature survey discovered no similar efforts when it comes to choice of method and focus of the problem. This may be due to that most efforts are technically oriented since their aim is to patch existing systems, whereas the proposed method is focused on humans and human interactions – the actors behind important decisions.

1 Introduction

IT security breaches enabled by inadequate development, administration and maintenance of information systems are a constant nuisance and serious threat to all organizations depending on distributed information systems. To lessen the problem effective methods for alleviation of system vulnerabilities are needed. There are many ways to trace security vulnerabilities in information systems. A common strategy is to deploy red teams[1], which actively search for vulnerabilities in the systems and propose suggestions on how to deal with the discovered problems. However, all approaches focused on finding security vulnerabilities, such as read teaming, leaves

[1] The term red team refers to a group of people who are assigned the task of finding vulnerabilities in a system. The term tiger team is also used to describe the same occurrence. Throughout this paper we will use the term red team.

several important questions unanswered: Why did the vulnerability occur in the first place? Was it something that could have been avoided? If so, how and to what cost?

Considering the size and complexity of distributed information systems, it is not a bold statement to assume that security vulnerabilities exist in all of them. Some of these vulnerabilities exist because of well-balanced decisions since the security has to be weighted against the cost and flexibility of the system. However, many of the vulnerabilities present in distributed information systems remain for unspecified reasons. Thus, a difficult question appears: How to use the limited resources available for security-related activities in such a way that the potential damage is minimized?

In this paper, an ongoing effort to produce a method for tracing the reason behind security vulnerabilities in IT systems is presented. We believe that the real reasons behind security vulnerabilities are found in the processes of production, procuration, and configuration of systems. By analyzing these processes, the proposed method should reveal the corresponding weaknesses and be able to suggest ways to improve the processes. That is, the method will not automatically correct flaws in existing systems, but it will be a way to improve the processes and organizational routines affecting the systems. The goal is a long-termed solution – not to find someone to punish. The purpose of the method is to discover non-technical reasons to why security vulnerabilities occur. In doing so, the focus is on problems that might not be obvious or documented. The interest lies in or around the development process, or if the development lies elsewhere in the procuration routines, and in the configuration of the systems. In these processes, decisions are made that affect security-relevant events in all system deployment processes. Even if technical issues will be discussed during some of the phases in the method, it is still the reasoning behind each decision and the way it was carried out that in the end forms the result.

2 Background

A common method for examining information systems and to discover vulnerabilities is to deploy red teams. The purpose of red teaming is to find technical weaknesses in systems that are in operation. As a result, the discoveries made by red teams allow the owner of the systems to reconfigure or patch the systems.

Red teams are of great importance when it comes to finding vulnerabilities. Their purpose is not only to find vulnerabilities, but also to examine how to exploit the vulnerabilities. Reports from red teams includes: the found vulnerabilities, how to explore them, and, the most important part, how to alleviate them. Midian [1] describes how a red team might operate and their purpose.

Unfortunately, there is a limit to the effort of a red team. The primary interest of red teams is technical and they do not consider why the vulnerability occurred in the first place. As a result, the effort of a red team has a short expiration date. Moreover, the value of red teaming is limited by the lack of proof of completeness. That is, the fact that a certain red team cannot find vulnerabilities in a system does not mean that there is none.

By using certified products, for instance Common Criteria [2] approved products, the system owners are assured, although not proved, that their products are secure as

long as they are used according to the evaluation prerequisites. Still, two by themselves evaluated and secure products may introduce vulnerabilities when put together as a new system [3].

In order to construct secure distributed information systems, the three parameters time, knowledge, and money have to be considered. A computer system is too complicated for a complete comprehension, thus systems are most often divided into subsystems or modules. Commonly, security vulnerabilities arise where these subsystems or modules interact. Red teams will probably find those weak technical links, but if it is not determined how they occurred in the first place, new weak links may occur in the systems for the same reasons. This is where the method proposed in this paper is hoped to make a difference by finding the underlying reasons for security vulnerabilities.

3 Method development

To develop a method for systematic search for the underlying reasons for security vulnerabilities, the following steps were considered necessary:

1. A list of possible underlying reasons has to be created. This list contains the set of possible underlying reasons for security vulnerabilities and the result from each application of the proposed method will be taken from this set. Thus, the quality of the results produced by the method is to a large extent depending on the contents of the list. An additional purpose of the list is to shape a common view of the target of the method and thereby both focus the effort involved in the method development and avoid subjects or parts being left out, that is, keep a broad view of the subject. Since the completeness of the list cannot be proved, the list has to be a live document under constant development and review. An initial list was created during a brainstorm session involving three IT security specialists.
2. An initial version of the method has to be developed. This involves two steps: (1) to create an overall layout of the method dividing it into a number of steps and (2) to find or formulate the basic tools needed to perform these steps. A draft of the overall layout was created and thereafter discussed at seminars with IT security and system development specialists. A set of basic tools was found through a literature study of different techniques and their deployment in different situations. Like the overall layout of the method, these tools were analyzed and adapted during a series of seminars.
3. A way of evaluating the method in-house needs to be determined. This is important to tune the method before it is applied to organizations and systems in operation.
4. Evaluations of the method in working environments have to be conducted in order to fine-tune the method and assess its usefulness.

In this paper, the effort to formulate the list of possible underlying reasons and the systematic search method, that is step 1 and 2 in the list above, are described.

4 Categorizing the underlying reasons for security vulnerabilities

Since the aim of the method is to find the underlying reasons, the analysis is concentrated to the development and configuration processes of the system. Studying these processes the focus will be on the people affecting their execution and performance. Consequently, a graph containing "soft" elements like stress and lack of motivation, knowledge, and time was constructed. All these elements are likely to result in security vulnerabilities. The graph, inspired by Kaoru Ishikawa's fishbone diagrams [4], is not a predetermined map but rather a method of gathering thoughts and, jointly, broaden the views on what may cause a security breach. The output of the map will constitute the set of underlying reasons being part of the result of the method. At a later stage, the graph was used when formulating the questionnaires used as a basic tool in the method. A sub-tree of the graph is presented in Figure 4.1.

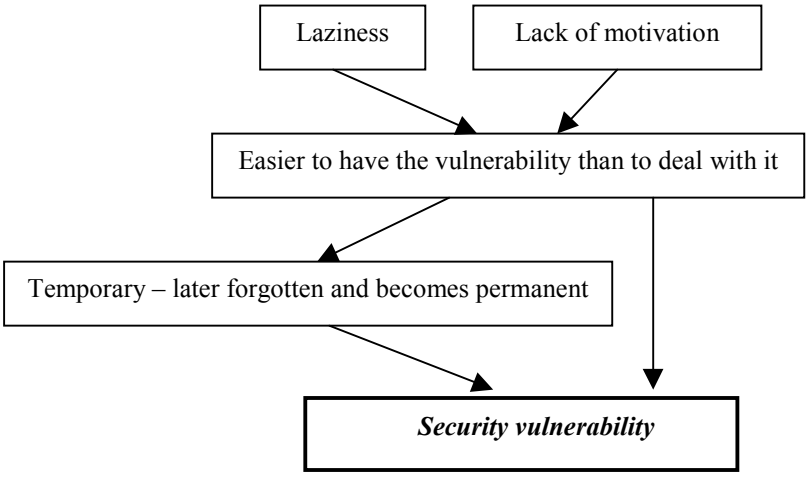

Fig. 1. A sub-tree of the graph of reasons.

By arranging the elements in the graph into groups, it was possible to get an overview of the larger picture. The naming of the groups suggests where the problems can be handled. Most of the groups span over the whole time axis of the development and configuration processes, but either one can be the dominant force at a given time. The groups are listed below.

Knowledge: Knowledge can refer to individual knowledge of group members, the ability to understand the difficulty of a problem and solving it.

It can also refer to tradition within an organization. Problems solved "the way we always have done it" may introduce or conserve well-known difficulties.

Time: Refers to the utilization of the amount of time provided for a specific task.

Difficult problems: The task itself may contain difficulties beyond the scope of the development process. It is possible that the task may include the use of functionality that is not entirely secure. In such cases the interest lays in discovering if the team was competent enough to take necessary precautions to prevent known vulnerabilities.
Difficult events: When planning a project, it is important to make a risk-analysis. Even if such an analysis is conducted, it is possible that events that are very difficult to foresee may occur. The interest in this case is to determine how such an event was handled.
Leadership/management: Leadership refers to the ability of a manager to guide and encourage the staff in order for them to perform assigned tasks.

Closely related to leadership is management, i.e. how resources are made available and handled within the scope of the responsibilities of managers.
Planning: A crucial component for a successful result is a well-set timeframe and organization of the task. This group traverses over multiple levels, from the top of the organization in a more broad view as well as on the lower levels with more hands-on duties.

The Time-group is closely related to this group.
Equipment: Even if the team is competent and properly managed, lack of efficient or specialized equipment may cause unwanted effects on the result.

Since some of the groups are not limited to a certain time or event in the process, it may be wise to differ between actual reason and the consequence of such. For instance, lack of time can be a consequence of poor planning, thus planning becomes the actual reason for the security vulnerability.

5 A method to find underlying reasons

With the graph in mind, the problem is to map security breaches into underlying reasons. To achieve this, information on the development, configuration and operation of the system has to be extracted from the organization. For this purpose a method is proposed. The method contains the following steps.

1. Select a security vulnerability.
2. Investigate technical details of the vulnerability.
3. Create a common understanding of the problem.
4. Select key persons to interview in order to extract relevant information.
5. Carry out the interviews.
6. Analyze the results of the interviews to build a common view of the underlying reasons to the existence of the vulnerability.

In the remainder of this chapter, the six steps of the method are discussed.

5.1 Selection of security vulnerability to study

A study using the proposed method is likely to start from a security incident [5] or possibly as a result of red teaming. It is important to stress that the method is first to be used when an incident has occurred, or a vulnerability has been discovered. The method is neither meant to replace red teams nor to be deployed for random searches. Thus, the first task is to derive one or more security vulnerabilities from that an incident. Vulnerabilities can, for instance, be found in the reports produced by red teams and vulnerability scanners as well as in incident reports describing actual intrusions.

For a vulnerability to be of interest to study, it has to meet some criteria. These criteria are designed so that they point at a discovered or occurred dangerous situation, which needs to be avoided in the future.

- It has to occur in an operational environment important to the organization.
- It should be possible for an attacker to create some actual damage, either by exploiting the vulnerability itself or in conjunction with other vulnerabilities.
- The vulnerability should not exist as a calculated risk in a risk analysis, i.e. it should not be a legal operation.

Naturally other properties can be addressed, but these are the basic statements. An incident meeting these criteria's is a candidate for further investigation. To confirm that these criteria are met, a deeper analysis is necessary. It is quite possible that the search has to proceed a few steps before it can be stated that all criteria are fully met. The following should be documented regarding the criteria at this step:

- A brief description of the vulnerability on a concrete level.
- The role of the vulnerability in the running environment.
- How is it possible for an attacker to utilize the vulnerability to cause real damage?
- Why it should not exist if a correct operation had been carried out, in regard to a correct risk analysis.

By documenting the vulnerability, it is possible to focus on the effort.

5.2 Investigation of technical details

To fully understand the vulnerability and in what way it can cause damage, it is necessary to learn as much as possible about the system it operates in. It is mainly technical details that are of interest at this point, considering both the system and the vulnerability.

Interesting questions that needs to be answered and documented at this stage are:

1. What is the typical usage of the system?
2. Define the role of the system in the context of other interacting systems.
3. How does the system work, and what components does it incorporate?
4. What is the technical description of the vulnerability?

5. Is there a technical solution that would render the vulnerability harmless? (Note that there can be more than one answer to this question. An adequate red team report should be able to answer this question.)

Already at this state it is possible to find some underlying reasons to the vulnerability. If the issuing organization cannot answer the questions above, it is quite possible that they do not fully understand their own system.

5.3 Selection of interviewees

An important step is to decide which roles, e.g. system administrators, project managers, system developers, and security officers, in the organization that would be of interest to interview. Key criteria for these people are their role in different processes, but most of all their ability to affect the security of the system during development and operation. The final selection of interviewees (most likely several) depends on the studied breach and the organization. A top-down approach should be suitable, starting with managers and continuing with the developers and administrators, thus moving from decision makers to implementers and maintenance. However, an alternative approach is to reverse the approach described above and make a bottom-up search for suitable candidates. Again, the decision of which approach to be used should be based on the studied vulnerability and the organization.

5.4 The interviews

Since the goal of the search is to find the non-technical reasons behind a security vulnerability, the main tool used in the method is interviews. The interviews provide a mechanism for reaching the people behind the decisions and implementations of systems. This is important since documentation produced during development and configuration usually is technically oriented.

As a preparatory step questionnaires are used before the interviews. Questionnaires provide a simple way of starting up the searching by focusing the persons to be interviewed on the precise situation and helping them to remember the role they had in the process. The interviews allows for a more unstructured follow-up of the questionnaires, and the ability to explore a certain event that may arise during the interview. Together interviews and questionnaires form a focused method of exploring the reason behind security vulnerabilities.

When setting up interviews some choices have to be made. Common for all interview methods is that they are indirect ways of gathering information, as opposed to observations and field-tests. It is the level of structure that differs between different interview strategies. Highly structured interviews are conducted with a predetermined question sheet and the interviewers are not allowed to follow up questions in other ways than to clarify questions. In short, it is basically like a verbal question sheet. The downside of structure is the lack of ability to penetrate a specific subject. By allowing a more unstructured approach, a deeper understanding of the situation is possible. [6].

For the systematic search method an unstructured interview approach was selected called the cognitive interview method [7]. The core of the method is the free report from the person that is being interviewed. Allowing the interviewees to recite freely from their memory, non-obvious aspects can be explored. Such non-obvious aspects would not be captured by a structured interview.

A cognitive interview scheme is divided into five sections:

1. Introduction
2. Spontaneous recapitulation
3. Guided recapitulation
4. Verification and completion
5. Conclusion

The introduction sets the frame for the interview. This is where the interviewer establishes the climate and explains what is going to be discussed. A good introduction makes the interviewee feel comfortable and willing to share what they know.

When the frame is set and the field of interest is defined, the interviewees are encouraged to describe how they have experienced the given situation with their own words. It is important that the interviewer only interrupts if it becomes necessary to change the perspective or to the discussed topic. The most important tool for the interviewer at this phase is silence. By using silence correctly, it is possible to make the interviewees continue their story a bit further even if they first considered it done.

Eventually the spontaneous recapitulation will end and the interviewer can start the guided recapitulation. The interviewer selects an event from the spontaneous recapitulation and asks for additional information. The questions are formulated to penetrate the issue from the outside and to the core of the event. As the guided recapitulation proceeds, the interviewees are encouraged to complement the story or continue where they stopped earlier.

During the verification and completion, the interviewer summarizes the story and asks respondent to confirm the conclusions. Even in this phase, the interviewees are encouraged to continue or start new recapitulations.

The interviewer concludes the interview by thanking the interviewee and to encourage them to take contact if something else comes into mind. This marks the end of the interview from the perspective of the interviewee, but for the interviewer the interview is still on. As long as the interviewer and the interviewee are talking the interview is continuing. It is not unusual that key information is given after the official interview is over when the interviewees are more relaxed.

An important consideration during the whole event is to make the interviewees feel comfortable. They should feel that the delivered presentation is important for the task at hand, but at the same time know that they do not defame themselves or anyone else. A positive feeling after the interview is necessary if further contact is to be possible in the future.

5.5 Analysis of the results

An analysis of the collected data will be conducted after each interview. The purpose of the analysis is to determine whether one or several underlying reasons to a security vulnerability can be deducted, if additional interviews are needed, and, if more interviews are needed, the corresponding interviewees. Obviously this constitutes a fairly blunt decision tool, but as the method matures a more precise description of the analysis criteria will evolve.

During each individual interview, the main considerations are of technical and organizational character. However, one or more common underlying reasons should be possible to detected and verify as the interviewing proceeds.

6 Discussion

In this paper, an effort to produce a method for the search for the underlying reasons for security breaches in distributed information systems is presented. The method approaches the problem by analyzing the processes of development, procurement, configuration, and operation. In order to initiate the method, a well-defined security vulnerability has to be selected. It is not assumed that the method will be a panacea solving all the problems of IT security or even the problem of prioritizing the security needs of organizations. However, the method can constitute a powerful tool for understanding the underlying mechanisms of why security vulnerabilities exist and why they remain, despite the effort put into securing the systems.

In order to make the method suitable for a variety of organizations, it has to be general enough to be able to adapt to different organizations and security breaches. Thus, the method is not a step-by-step manual to follow but more of a guideline of how to find the underlying reasons for security vulnerabilities. Consequently, it has to be adapted to the specific circumstances of the studied security vulnerability and organization. However, the adaptation should be neither difficult nor time consuming. In a cognitive interview, the main player is not the interviewer but the interviewee. Thus, few specific questions have to be altered. The same applies to the questionnaires. In the questionnaire, the questions regard how the interviewees experienced the situation and how they made, or could have made, a difference.

A potential problem is opinions about the use of such a method. It is not a tool to alleviate discovered breaches in existing systems. The effort is much more long termed. The purpose is to help an organization to avoid making the same mistakes again. It is a follow-up tool aimed at providing knowledge of where the weak links are and all in all to raise the security awareness within the organization.

How well the method will work depends on the openness of the studied organization. If the method is used internally, a higher level of openness can be assumed. A group from the outside may encounter some resistance, albeit unintentionally, mainly due to fear of unwanted exposure of their internal routines and situation. This should not constitute a major problem, since one of the goals of the project is to present a method that can be used on and by any organization.

There are of course different aspects to consider when deciding who should conduct the search. In the case of an internal investigation, the confidentiality of the information should not be a problem. On the other hand, conducting an interview is a role-play. The power balance between the actors is sensitive and should be as equal as possible. With this in mind, the interviewer cannot be a supervisor or any other position with the power to affect the future within the company for the interviewed persons. If external contractors are being used, the company should have considered the security risk when hiring them and, thus, taken the necessary legal precautions.

Most organizations do not use internally developed hardware or software. Most of the applications are commercial products or, when a unique application is needed, specially ordered products. This can make an examination of the development process hard. It is possible that an examination cannot proceed further than the acquisition of some application. This does not necessarily mean a limitation of the method for the corresponding organization. In order to conduct a purchase, a list of demands should be produced stating the needs and restrictions of the application. This process is, like the development process, both interesting and error prone and, thus, important to study.

References

1. Midian, P.: Perspectives on Penetration Testing. Computer Fraud & Security. Issue. 6. (2002) 15-17
2. Common Criteria. Available at www.commoncriteria.org. 2002-09-27.
3. Gritzalis, S., Spinellis, D.: The cascade vulnerability problem: The detection problem and a simulated annealing approach for its correction. Microprocessors and Microsystems. Vol. 21. No. 10. (1998) 621-628
4. Ishikawa, K.: Guide to quality control. Quality resources/A division. (1974)
5. Levin, C.: Net security reawakening. PC Magazine. June (1995) 31
6. Kvale, S.: Interviews: An introduction to qualitative research. Sage. (1996)
7. Christianson, S-Å., Engelberg, E., Holmberg, U.: Avancerad förhörs- och intervjumetodik. Natur och kultur. Bokförlaget. (1998)

A Proposed Taxonomy for IT Weapons

Martin Karresand

FOI
Swedish Defence Research Agency, Division of Command and Control Systems,
Department of Systems Analysis and IT Security
Box 1165, S-581 11 Linköping, Sweden
martin.karresand@foi.se

Abstract This report presents a proposal for a taxonomy of IT weapons, limited to computer software. Because of this limitation the term software weapons is used instead of IT weapons. A definition of software weapons is also formulated. No other taxonomy with the above scope is known to exist today. This taxonomy therefore offers a theoretical base for the unification of the nomenclature for classification of software weapons

The taxonomy contains 15 categories of general properties of a software weapon. It has been adapted to international standards through a connection to the CVE list (Common Vulnerabilities and Exposures), which is maintained by MITRE.

The problem of how to make unambiguous classifications of combined software weapons is discussed and a solution is proposed. Each category of the taxonomy is explained in a separate paragraph. Thereafter the taxonomy is used to classify two well known software weapons.

Keywords: computer security, information warfare, IT weapon, IW, malware, software weapon, taxonomy, virus, worm.

1 Introduction

The terminology used in the IT security area is not yet fully standardised and the situation is getting worse [1,2,3] because of the continuous influx of new members to the research community, who all have their own preferred vocabulary. Hence there is an increasing need for a standardisation of the used terms.

On top of that the research being done so far has been concentrated on the pragmatic, technical side of the spectrum, i.e. ways of detecting the increasing amount of malware (malicious software) being written. The classification and grouping of the malware has been given less attention and the area is therefore hard to take in.

To enable the development of efficient countermeasures there is a need for an unambiguous classification of the software used to cause harm to individuals and organisations via computer systems. Also the users of the computer systems need to have a general understanding of the threats posed by different types of

malware. This would lead to a higher degree of awareness of possible malware attacks and in that way higher security. One step towards this goal is to have commonly acknowledged names for the separate types of malware. Consequently these types must be well defined too.

Furthermore, the education of IT security personnel would benefit from a structured classification of the malware area. A common vocabulary would for example decrease the risk of misunderstandings.

Whether a specific tool would be classified as a weapon or not is often judged from the context of the situation where the tool is used. This is the juridical point of view, the tool a murderer used is to be regarded as a weapon, because he or she used it with the intent to kill. Consequently, anything can be a weapon.

The sentence 'He used a pillow as a weapon' gives that the pillow was a weapon *in that specific situation*. But by disregarding the context and just concentrate on the '*as a weapon*' part of the sentence, we see that a tool must have certain properties to be a weapon.[1] These properties are in some way measurable; they do harm (why else would they be used for fighting and attacking?). If the line of argument is transferred to the world of computers, the result is that a certain class of software has a specific set of properties, which are measurable, and those properties define the software as weapons.

The advantage of this approach is the much lesser degree of ambiguity. A weapon is a weapon because of its properties and as long as the purpose is to study it technically, that is enough. With a deeper knowledge of the technical details of software weapons (malware) as a group, they can be classified, properly named, etc. This in turn leads to a more structured knowledge of the area and thus a possibility to develop better defences, maybe even in advance. Or as Sun Tzu once wrote in his book The Art of War [5, chapter VI]:

> Whoever is first in the field and awaits the coming of the enemy, will be fresh for the fight; whoever is second in the field and has to hasten to battle will arrive exhausted.

1.1 Background

This is an updated version of a report [6] written in Swedish. The amendments were mostly related to preparing the report for international publishing. Some parts have also been rewritten because new background material has been found.

In an attempt to somewhat lessen the emotional charge in the used vocabulary, the term *software weapon* will be used throughout the text. Something malicious can be nothing but evil, but a weapon is simply a tool that has the ability to cause harm and that can be used in both offensive and defensive situations.

Regarding the area of malware, several definitions and classification schemes exist for different parts of the area (see for example [7,8,9]). Most of them deal

[1] One definition is 'an object such as a knife, gun, bomb, etc. that is used for fighting or attacking sb'. [4]

with viruses and worms, and only casually mention other types of malicious software. They all give their own way of measuring, or quantifying, maliciousness and at the same time conclude that this cannot be done objectively.

No definition or taxonomy[2] covering the complete set of software-based IT weapons has yet been found by the author.

1.2 Purpose

The purpose of the report is to present a taxonomy of software weapons, and also give a definition supporting the taxonomy. The taxonomy and definition are not meant to be complete in any way, but merely suggestions for future work.

The purpose of the taxonomy is to fill the needs stated in the introduction, or at least lay the foundation for a future fulfilment of them.

1.3 Scope

The taxonomy only handles software based (IT) weapons from a technical point of view. Chipping[3] is considered to be hardware based and is therefore not discussed.

1.4 Method

The study has been done with a broad technical base. Several different types of material have been studied. Most of the material has been taken from the Internet to give up to date information. It has mostly been descriptions of tools and methods used by hackers. Also technical reports, dissertations, and taxonomies focused on IT security and malware have been used.

2 A Taxonomy of Software Weapons

My own hypothesis of why no other taxonomy of software weapons has yet been found can be summarised in the following points:

- The set of all software weapons is (at least in theory) infinite, because new combinations and strains are constantly evolving. Compared to the biological world, new mutations can be generated at light speed.
- It is hard to draw a line between administrative tools and software weapons. Thus it is hard to strictly define what a software weapon is.
- Often software weapons are a combination of other, atomic, software weapons. It is therefore difficult to unambiguously classify such a combined weapon.

[2] The word originates from the two Greek words *taxis*, arrangement, order, and *nomos*, distribution.

[3] Malicious alterations of computer hardware.

- There is no unanimously accepted theoretical foundation to build a taxonomy on. For instance there are (at least) five different definitions of the term *worm* [10] and seven of *trojan* [11].
- By using the emotionally charged word *malicious* together with *intent*, the definitions have been crippled by the discussion whether to judge the programmer's or the user's intentions.

2.1 Theory

As a consequence of some of the problems mentioned above, the set of software weapons will grow continuously. Therefore it can be regarded as always new and unexplored. The fact that software weapons can be created from combinations of other software weapons, without limitations, gives that a traditional taxonomy based on relationships would not work very well. The rules for classification would grow indefinitely complex and soon get out of hand. A better solution would be to base the taxonomy on technical characteristics. With a proper selection of characteristics, such a taxonomy would have the potential to work for more than a few years.

It is not enough to find a working set of characteristics to get a good taxonomy, though. It must fulfil a few more requirements to be useful. Daniel Lough has created a list of 18 properties from 5 different taxonomies of IT security, which he presents in his dissertation [12]. I consider the following properties taken from two of those taxonomies [13,14] to be the most important. The categories of the taxonomy should:

- Be *mutually exclusive* and *exhaustive* so that the taxonomy completely covers the intended area, i.e. be a *partitioning* of the area
- Be *unambiguous* to prevent subjective interpretations
- *Usable* through the use of well known and consistent terminology.

To minimise the risk of subjective interpretations when classifying objects, the alternatives in each category should be based on measurable or observable characteristics [15]. In the case of software these characteristics are the instructions and algorithms constituting the software [16]. This will guarantee that the classification of a software weapon will be the same, regardless of who is classifying.

How the characteristics of the software weapon shall be found is a separate problem. It can be done by either having access to the source code of the weapon, or by re-engineering the source code from a binary version of the weapon. A third way is to have some sort of automatic analysis software; a virtual environment where the software weapon could be scientifically studied in a controlled manner. Such an environment already exists for worms and viruses [10].

2.2 Definition

In this section a definition of software weapons is presented, together with the reasons for developing it. To avoid influences from the definitions of malware mentioned earlier, the new definition has been constructed with information warfare as a base.

Background. There are several definitions of IT and cyber warfare. Of course they cover a much larger area than just software weapons, but they do give a hint of what the important things are. The US Department of Defense has the following definition of the military part of information warfare [17]:

> Information Warfare - Actions taken to achieve information superiority in support of national military strategy by affecting adversary information and information systems while leveraging and defending our information and systems.

Dr. John Alger, MITRE Corporation, Enterprise Security Solutions Department, gives the following definition of information warfare in a book by Winn Schwartau [18, p. 12]:

> Information Warfare consists of those actions intended to protect, exploit, corrupt, deny, or destroy information or information resources in order to achieve a significant advantage, objective, or victory over an adversary.

A similar definition is given by Ivan Goldberg [19], head of IASIW (US Institute for the Advanced Study of Information Warfare):

> Information warfare is the offensive and defensive use of information and information systems to deny, exploit, corrupt, or destroy, an adversary's information, information-based processes, information systems and computer-based networks while protecting one's own. Such actions are designed to achieve advantages over military or business adversaries.

All the above definitions mentions that an advantage over an adversary should be achieved and this should be done by influencing the adversary's information systems. An advantage in the software context would correspond to a breach in the security of the adversary's computer system. The influencing part would then be the instructions of the tool(s) used for the attack. Thus a software weapon should have such properties.

The definitions mentioned above are all very much alike, which might indicate that they all have the same source. If so, three renowned institutions has adopted it, which in that case strengthens its importance. I therefore think that the definitions above carry such weight that they can be used as a basis for the definition of software weapons used in this report.

Preliminary Definition. The preliminary definition of software weapons[4] used at FOI[5] has the following wording (translated from Swedish):

> [...] software for logically influencing information and/or processes in IT systems in order to cause damage.[6]

[4] The term *IT weapon* is used in the report FOI report.

[5] Swedish Defence Research Agency

[6] In Swedish: '[...] programvara för att logiskt påverka information och/eller processer i IT-system för att åstadkomma skada.'

This definition satisfies the conditions mentioned earlier in the text. One thing worth mentioning is that tools without any logical influence on information or processes are not classified as software weapons by this definition. This means that for instance a sniffer is not a software weapon. Even a denial of service weapon might not be regarded as a weapon depending on the interpretation of 'logically influencing ... processes'. A web browser on the other hand falls into the software weapon category, because it can be used in a dot-dot[7] attack on a web server and thus affect the attacked system logically.

Furthermore, the definition does not specify if it is the intention of the user or the programmer, that should constitute the (logical) influence causing damage. If it is the situation where the tool is used that decides whether the tool is a software weapon or not, theoretically all software ever produced can be classified as software weapons.

If instead it is the programmer's intentions that are decisive, the definition gives that the set of software weapons is a subset (if yet infinite) of the set of all possible software. But in this case we have to trust the programmer to give an honest answer (if we can figure out whom to ask) on what his or her intentions was.

A practical example of this dilemma is the software tool *SATAN*, which according to the creators was intended as a help for system administrators [20,21]. *SATAN* is also regarded as a useful tool for penetrating computer systems [22]. Whether *SATAN* should be classified as a software weapon or not when using the FOI definition is therefore left to the reader to subjectively decide.

New Definition. When a computer system is attacked, the attacker uses all options available to get the intended result. This implies that even tools made only for administration of the computer system can be used. In other words there is a grey area with powerful administrative tools, which are hard to decide whether they should be classified as software weapons or not. Hence a good definition of software weapons is hard to make, but it might be done by using a mathematical wording and building from a foundation of measurable characteristics.

With the help of the conclusions drawn from the definitions of information warfare the following suggestion for a definition of software weapons was formulated:

> A software weapon is software containing *instructions* that are necessary and sufficient for a *successful attack* on a *computer system*.

Even if the aim was to keep the definition as mathematical as possible, the natural language format might induce ambiguities. Therefore a few of the terms used will be further discussed in separate paragraphs.

[7] A dot-dot attack is performed by adding two dots directly after a URL in the address field of the web browser. If the attacked web server is not properly configured, this might give the attacker access to a higher level in the file structure on the server and in that way non-authorised rights in the system.

Since it is a definition of *software* weapons, manual input of instructions is excluded.

Instructions. It is the instructions and algorithms the software is made of that should be evaluated, not the programmer's or the user's intentions. The instructions constituting a software weapon must also be of such dignity that they together actually will allow a breakage of the security of an attacked system.

Successful. There must be at least one computer system that is vulnerable to the tool used for an attack, for the tool to be classified as a software weapon. It is rather obvious that a weapon must have the ability to do harm (to break the computer security) to be called a weapon. Even if the vulnerability used by the tool might not yet exist in any working computer system, the weapon can still be regarded as a weapon, as long as there is a theoretically proved vulnerability that can be exploited.

Attack. An attack implies that a computer program in some way affects the *confidentiality*[8], *integrity*[9] or *availability*[10] of the attacked computer system. These three terms form the core of the continually discussed formulation of computer security. Until any of the suggested alternatives is generally accepted, the definition of attack will adhere to the core.

The security breach can for example be achieved through taking advantage of flaws in the attacked computer system, or by neutralising or circumventing its security functions in any way.

The term *flaw* used above is not unambiguously defined in the field of IT security. Carl E Landwehr gives the following definition [24, p. 2]:

> [...] a security flaw is a part of a program that can cause the system to violate its security requirements.

Another rather general, but yet functional, definition of ways of attacking computer systems is the definition of vulnerability and exposure [25] made by the CVE[11] Editorial Board.

Computer System. The term computer system embraces all kinds of (electronic)[12] machines that are programmable and all software and data they contain. It can be everything from integrated circuits to civil and military systems (including the networks connecting them).

---

[8] '[P]revention of unauthorised disclosure of information.'[23, p. 5]

[9] '[P]revention of unauthorised modification of information.'[23, p. 5]

[10] '[P]revention of unauthorised withholding of information or resources.'[23, p. 5]

[11] '[CVE is a] list of standardized names for vulnerabilities and other information security exposures – CVE aims to standardize the names for all publicly known vulnerabilities and security exposures. [...] The goal of CVE is to make it easier to share data across separate vulnerability databases and security weapons.' [26]. The list is maintained by MITRE [27].

[12] This term might be to restrictive. Already advanced research is done in for example the areas of biological and quantum computers.

Evaluation. To test if the new definition has the intended extent, it is applied to a selection of common hacker tools. First five classes of tools chosen from a list made by David Icove [28, pp. 29–60] is used, then two tools not commonly regarded as software weapons, a web browser and a word processor.

To get as relevant a test as possible, tools that have a high ambiguity with respect to whether they should be regarded as software weapons or not are selected.

Denial of Service. Tools that in some way degrade the service of a computer system exist in several versions. The instructions of such a tool is both necessary and sufficient to successfully degrade the availability of the attacked system and it is thus a software weapon.

Data Diddling. A tool performing unauthorised manipulation of data on the attacked system can for instance be a log eraser. The definition states that this is a software weapon, because the tool affects the integrity of the attacked system.

Port Scanning. A port scan can be compared to going round a house (in full daylight) trying all the doors and windows to see if any of them is open [29]. Such knowledge can then be used for intrusion.

On the other hand, merely studying the visual characteristics of an object does not affect its confidentiality. Something clearly visible cannot be regarded as secret. Thus, such a simple port scanner as the one described above is not sufficient enough to affect the confidentiality of the scanned system and is therefore not a software weapon.

However, what today commonly is known as a security scanner is more powerful than the tool described above. A few examples are *SATAN*, *Nessus*, and *NSS*. They can for instance search for specific vulnerabilities and perform port mapping for different applications. Such a tool contains instructions both necessary and sufficient to affect the confidentiality of the attacked system.

Password Sniffing. By analysing the content of packets sent over a network passwords can be found, without interrupting the network traffic. If the sniffed passwords are unencrypted (or can be decrypted by the sniffer), the password sniffing is necessary and sufficient to violate the confidentiality of the attacked system and the sniffer is therefore a software weapon.

On the other hand, if the sniffer tool itself does merely send the encrypted passwords to another tool for decryption, its instructions is not sufficient for a successful attack. In other words, a sniffer is a good example of a tool that reside in the grey area.

Traffic Analysis. Tools performing traffic analysis work in a similar way to password sniffers, but instead they use the address field of the data packet. In that way they can be used for mapping the topology of a network. The information can be used to identify specific servers and security functions. These can then be circumvented or attacked.

The situation can be compared to a reconnaissance device collecting data on the positions of enemy troops on a battle field. Such data is most likely confidential and might be necessary and sufficient for a successful attack on the enemy, i.e. a traffic analysis tool is a software weapon [30,29]

However, many traffic analysis tools are manually operated, i.e. the user gives the parameters that control the operation. These parameters can then be viewed as the instructions that perform the actual attack. Thus in this case the traffic analysis tool itself cannot be regarded as being a software weapon. Instead it should be compared to a terminal program.

From the above we can see that a traffic analyser occupies the grey area mentioned before. Each traffic analyser therefore has to be inspected separately to determine whether it should be classified as a software weapon or not.

Web Browser. Using a web browser a hacker can make a dot-dot attack on a computer system. In this case the actual instructions representing the attack are given by the user, not the web browser. Thus the instructions constituting a web browser are not sufficient to successfully attack a computer system and consequently the browser is not a software weapon. Instead it can be regarded as a manually operated terminal program.

Word Processor. Through the built-in macro language, a word processor can be utilised to perform unauthorised actions on an attacked system. The instructions used for the attack are given by the macro, the word processor only interprets them. In other words the word processor does not in itself contain the instructions that perform the attack. Thus a word processor is not a software weapon (but the macro is).

Summary of Evaluation. The tools that challenged the definition the most were the traffic analyser and the port scanner. Both tools can very well be used by a system administrator for totally legitimate purposes. For example a traffic analyser can be used by an administrator to continuously monitor the traffic in a network and in that way detect anomalies signalling an intrusion. A port scanner can be used to test the security configuration of the system and especially the firewall set-up.

It is therefore important to remember that it is the code constituting the software that should contain instructions that are necessary and sufficient be used for a successful attack. If a port scanner does more than just scan for open ports in a firewall, it might very well perform actions successfully affecting the confidentiality of the scanned system and as a result be a software weapon, regardless of the context.

2.3 A Draft for a Taxonomy

The categories of the taxonomy are independent and the alternatives of each category together form a partition of the category. It is possible to use several alternatives (where applicable) in a category at the same time. In this way even

combined software weapons can be unambiguously classified. This model, called *characteristics structure*, is suggested by Daniel Lough [12, p. 152].

In Table 1 the 15 categories and their alternatives are presented. The alternatives are then explained in separate paragraphs.

Table 1. The taxonomic categories and their alternatives

Category	Alternative 1	Alternative 2	Alternative 3
Type	atomic	combined	
Affects	confidentiality	integrity	availability
Duration of effect	temporary	permanent	
Targeting	manual	autonomous	
Attack	immediate	conditional	
Functional area	local	remote	
Sphere of operation	host-based	network-based	
Used vulnerability	CVE/CAN	other method	none
Topology	single source	distributed source	
Target of attack	single	multiple	
Platform dependency	dependent	independent	
Signature of code	monomorphic	polymorphic	
Signature of attack	monomorphic	polymorphic	
Signature when passive	visible	stealth	
Signature when active	visible	stealth	

Type. This category is used to distinguish an *atomic* software weapon from a *combined* and the alternatives therefore cannot be used together.

A combined software weapon is built of more than one stand-alone (atomic or combined) weapon. Such a weapon can utilise more than one alternative of a category. Usage of only one alternative from each category does not necessarily implicate an atomic weapon. In those circumstances this category indicates what type of weapon it is.

Affects. At least one of the three elements *confidentiality*, *integrity* and *availability* has to be affected by a tool to make the tool a software weapon.

These three elements together form the core of most of the definitions of IT security that exist today. Many of the schemes propose extensions to the core, but few of them abandon it completely.

Duration of effect. This category states for how long the software weapon is affecting the attacked system. It is only the effect(s) the software weapon has on the system during the weapon's active phase that should be taken into account. If the effect of the software weapon ceases when the active phase is over, the duration of the effect is *temporary*, otherwise it is *permanent*.

Regarding an effect on the confidentiality of the attacked system, it can be temporary. If for example a software weapon e-mails confidential data to the attacker (or another unauthorised party), the duration of the effect is temporary. On the other hand, if the software weapon opens a back door into the system (and leaves it open), the effect is permanent.

Targeting. The target of an attack can either be selected *manual*[ly] by the user, or *autonomous*[ly] (usually randomly) by the software weapon. Typical examples of autonomously targeting software weapons are worms and viruses.

Attack. The attack can be done *immediate*[ly] or *conditional*[ly]. If the timing of the attack is not governed by any conditions in the software, the software weapon uses immediate attack.

Functional Area. If the weapon attacks its host computer, i.e. hardware directly connected to the processor running its instructions, it is a *local* weapon. If instead another physical entity is attacked, the weapon is *remote*.

The placement of the weapon on the host computer can be done either with the help of another, separate tool (including manual placement), or by the weapon itself. If the weapon establishes itself on the host computer (i.e. breaks the host computer's security) it certainly is local, but can still be remote at the same time. A weapon which is placed on the host computer manually (or by another tool) need not be local.

Sphere of Operation. A weapon affecting network traffic in some way, for instance a traffic analyser, has a *network-based* operational area. A weapon affecting stationary data, for instance a weapon used to read password files, is *host-based*, even if the files are read over a network connection.

The definition of stationary data is data stored on a hard disk, in memory or on another type of physical storage media.

Used Vulnerability. The alternatives of this category are *CVE/CAN*[13], *other method* and *none*. When a weapon uses a vulnerability or exposure [25] stated in the CVE, the CVE/CAN name of the vulnerability should be given[14] as the alternative (if several, give all of them).

[13] The term CAN (Candidate Number) indicates that the vulnerability or exposure is being investigated by the CVE Editorial Board for eventually receiving a CVE name [31].

[14] NIST (US National Institute of Standards and Technology) has initiated a meta-base called ICAT [32] based on the CVE list. This meta-base can be used to search for CVE/CAN names when classifying a software weapon.

The meta-base is described like this: 'ICAT is a fine-grained searchable index of standardized vulnerabilities that links users into publicly available vulnerability and patch information'. [33]

The alternative *other method* should be used with great discrimination and only if the flaw is not listed in the CVE, which then regularly must be checked to see if it has been updated with the new method. If so, the classification of the software weapon should be changed to the proper CVE/CAN name.

Topology. An attack can be done from one *single source* or several concurrent *distributed sources*. In other words, the category defines the number of concurrent processes used for the attack. The processes should be mutually coordinated and running on separate and independent computers. If the computers are clustered or in another way connected as to make them simulate a single entity, they should be regarded as one.

Target of Attack. This category is closely related to the category *topology* and has the alternatives *single* and *multiple*. As for the category topology, it is the number of involved entities that is important. A software weapon concurrently attacking several targets is consequently of the type multiple.

Platform Dependency. The category states whether the software weapon (the executable code) can run on one or several platforms and the alternatives are consequently *dependent* and *independent*.

Signature of Code. If a software weapon has functions for changing the signature of its code, it is *polymorphic*, otherwise it is *monomorphic*. The category should not be confused with *Signature when passive*.

Signature of Attack. A software weapon can sometimes vary the way an attack is carried out, for example perform an attack of a specific type, but in different ways, or use different attacks depending on the status of the attacked system. For instance a dot-dot attack can be done either by using two dots, or by using the sequence `%2e%2e`. If the weapon has the ability to vary the attack, the type of attack is *polymorphic*, otherwise it is *monomorphic*.

Signature When Passive. This category specifies whether the weapon is *visible* or uses any type of *stealth* when in a passive phase[15]. The stealth can for example be achieved by catching system interrupts, manipulating checksums or marking hard disk sectors as bad in the FAT (File Allocation Table).

Signature When Active. A software weapon can be using instructions to provide *stealth* during its active phase. The stealth can be achieved in different ways, but the purpose is to conceal the effect and execution of the weapon. For

[15] A passive phase is a part of the code constituting the software weapon where no functions performing an actual attack are executed.

example man-in-the-middle or spoofing weapons use stealth techniques in their active phases through simulating uninterrupted network connections.

If the weapon is not using any stealth techniques, the weapon is *visible*.

3 Examples

In this section, as a test, two software weapons are classified using the taxonomy. The weapons used are the distributed denial of service (DDoS) weapon Stacheldraht and the worm CodeRed. They were chosen for being well documented and well known.

The test is in no way exhaustive. It is only meant to function as a demonstration of what a classification can look like for a particular software weapon.

3.1 Stacheldraht.

The classification of the DDoS weapon Stacheldraht was made with the help of [34,35] and looks like this:

Type: *combined*
Affects: *availability*
Duration of effect: *temporary*. The agents used to get the distributed characteristic of the weapon are installed permanently in the computers they reside on. To get them in place other tools are used [34], so the placing of the agents is to be considered as a separate attack not done with Stacheldraht.
The actual denial of service attack affects the attacked system until the attacker decides to quit.
Targeting: *manual*
Attack: *conditional*
Functional area: *remote*. As stated above, the placement of the agents is not considered an attack performed by Stacheldraht.
Sphere of operation: *network based*
Used vulnerability: *none*
Topology: *distributed source*
Target of attack: *multiple*
Platform dependency: *dependent*
Signature of code: *monomorphic*
Signature of attack: *monomorphic* The weapon can use ICMP flood, SYN flood, UDP flood, and Smurf style attacks, which are separate types of attacks.
Signature when passive: *visible*
Signature when active: *visible, stealth* The stealth is used in the communication between the different parts of the weapon (client, handler, and agent). This is done through using `ICMP_ECHOREPLY` packets and encrypted TCP [34].

3.2 CodeRed.

The classification of the worm CodeRed was made with the help of [36,37,38] and looks like this:

Type: *combined*
Affects: *integrity, availability*
Duration of effect: *temporary.* The documentation states that nothing is written to disk [37]. However, the weapon is also said to look for a file named 'NOTWORM' in the root of C:. How that file ends up there is not mentioned. Regarding the defacing of the server it is done in memory by hooking and redirecting incoming request to the worm code during 10 hours [36], i.e. a temporary effect. The DoS attack is also limited in extent and therefore a temporary effect.
Targeting: *autonomous*
Attack: *conditional*
Functional area: *local, remote.* The weapon (in version 1) defaces the server it has infected and also performs a DoS attack on a specific IP address. Therefore it is both local and remote.
Sphere of operation: *host based, network based.* See the previous category.
Used vulnerability: *CVE-2001-0500 (idq.dll), CVE-2001-0506 (SSI)*
Topology: *single source*
Target of attack: *single, multiple.* The weapon executes a DoS attack on a single IP address. It is also divided into several (99 + 1) threads, which all concurrently tries to infect (attack) randomly chosen IP addresses. This makes it both a single and multiple target attacking weapon.
Platform dependency: *dependent*
Signature of code: *monomorphic*
Signature of attack: *monomorphic* There are both a DoS attack and an infection mechanism, but each type of those two attacks are always executed in the same way.
Signature when passive: *visible.* The weapon is put to sleep when certain conditions are met. This cannot be regarded as using any stealth technique.
Signature when active: *visible*

4 Summary

The report has outlined a suggestion for a taxonomy, i.e. a classification scheme and a definition of software weapons. The definition part has been given much weight, because a classification scheme must have a solid base to work properly. To enable an unambiguous definition the emphasis was moved from the use of the weapon, to the technical (measurable) characteristics of the weapon. This gave the following formulation:

> A software weapon is software containing *instructions* that are necessary and sufficient for a *successful attack* on a *computer system.*

The classification part is meant to be used at a rather abstract level and for that reason the categories (and their alternatives) are chosen to be general properties held by all software weapons. A classification of a weapon must contain at least one alternative from each category.

By incorporating CVE names the taxonomy offers a connection to a global standard for naming vulnerabilities and exposures in software. This means that a meta-base of software weapons can be built, which can offer a global standardisation of the area.

The work done so far has been mainly theoretical. The next thing to do is to test the taxonomy empirically. Each category and its alternatives must be thoroughly tested to see if any of them needs to be changed.

Also the quality of the classification scheme needs to be tested. Software weapons related by common sense shall also have fairly similar classifications and unrelated weapons more or less be orthogonally classified.

References

1. Ford, R.: (Malware) `http://www.malware.org/malware.htm`, accessed 17 July 2002.
2. Helenius, M.: Problems, advantages and disadvantages of malware testing. In: EICAR 1999 Best Paper Proceedings. (1999) `http://conference.eicar.org/past_conferences/1999/other/Helenius.pdf`, accessed 18 July 2002.
3. Kaminski, J., O'Dea, H.: (How to smell a RAT - remote administration tools vs backdoor Trojans) `http://www.virusbtn.com/conference/this_year/abstracts/remote_administration.xml`, accessed 22 July 2002.
   Only the abstract of the paper was available and therefore no references are made to the body of the document.
4. Hornby, A.S.: Oxford advanced learner's dictionary of current English. 6 edn. Oxford University Press, Oxford (2000)
5. Tzu, S.: The Art of War. (500 B.C.) `http://all.net/books/tzu/tzu.html`, accessed 12 June 2002.
   Translation by Lionel Giles, 1910.
6. Karresand, M.: Tebit – teknisk beskrivningsmodell för it-vapen. Technical report, Command and Control Warfare Technology, FOI - Swedish Defence Research Agency (2001)
7. Brunnstein, K.: From AntiVirus to AntiMalware Software and Beyond: Another Approach to the Protection of Customers from Dysfunctional System Behaviour, Faculty for Informatics, University of Hamburg, Germany. (1999) `http://csrc.nist.gov/nissc/1999/proceeding/papers/p12.pdf`, accessed 22 July 2002.
8. Helenius, M.: A System to Support the Analysis of Antivirus Products' Virus Detection Capabilities. PhD thesis, University of Tampere, Finland (2002) `http://acta.uta.fi/pdf/951-44-5394-8.pdf`, accessed 22 July 2002.
9. Swimmer, M.: (Malware) `http://www.swimmer.org/morton/malware.html`, accessed 18 July 2002.
10. Whalley, I., Arnold, B., Chess, D., Morar, J., Segal, A., Swimmer, M.: An Environment for Controlled Worm Replication and Analysis or: Internet-inna-Box. (2000) `http://www.research.ibm.com/antivirus/SciPapers/VB2000INW.htm`, accessed 18 July 2002.

11. Whalley, I.: Testing Times for Trojans. (1999) http://www.research.ibm.com/antivirus/SciPapers/Whalley/inwVB99.html, accessed 18 July 2002.
12. Lough, D.L.: A Taxonomy of Computer Attacks with Applications to Wireless Networks. PhD thesis, Virgina Polytechnic Institute and State University (2001) http://scholar.lib.vt.edu/theses/available/etd-04252001-234145/unrestricted/lough.dissertation.pdf, accessed 13 June 2002.
13. Howard, J.D.: An Analysis of Security Incidents on the Internet 1989-1995. PhD thesis, Carnegie Mellon University, Pittsburg (1997) http://www.cert.org/research/JHThesis/Word6/, accessed 12 June 2002.
14. Lindqvist, U., Jonsson, E.: How to systematically classify computer security intrusions. In: Proceedings of the 1997 IEEE Symposium on Security & Privacy, Oakland, CA, IEEE Computer Society Press (1997) 154–163 http://www.ce.chalmers.se/staff/ulfl/pubs/sp97ul.pdf, accessed 12 June 2002.
15. Krsul, I.V.: Software Vulnerability Analysis. PhD thesis, Purdue University (1998) http://www.acis.ufl.edu/~ivan/articles/main.pdf, accessed 13 June 2002.
16. Bagnall, R.J., French, G.: The Malware Rating System (MRS)^{TM}. (2001) http://www.dodccrp.org/6thICCRTS/Cd/Tracks/Papers/Track7/105_tr7.pdf, accessed 22 July 2002.
17. Haeni, R.: What is Information Warfare. (1996) http://tangle.seas.gwu.edu/~reto/infowar/what.htm, accessed 27 June 2001.
18. Schwartau, W.: Information Warfare – Cyberterrorism: Protecting Your Personal Security in the Electronic Age. 2 edn. Thunder's Mouth Press, New York, NY (1996)
19. Goldberg, I. (2001) http://www.psycom.net/iwar.1.html, accessed 26 June 2002.
20. CERT (Computer Emergency Response Team): CERT Advisory CA-1995-06 Security Administrator Tool for Analyzing Networks (SATAN). (1995) http://www.cert.org/advisories/CA-1995-06.html, accessed 12 June 2002.
21. Gordon, S.: Devil's Advocate. (1995) http://www.commandsoftware.com/virus/satan.html, accessed 23 July 2002.
22. CIAC (Computer Incidents Advisory Center): Information Bulletin F-20: Security Administrator Tool for Analyzing Networks (SATAN). (1995) http://www.ciac.org/ciac/bulletins/f-20.shtml, accessed 12 June 2002.
23. Gollmann, D.: Computer Security. John Wiley & Sons (1999)
24. Landwehr, C.E., Bull, A.R., McDermott, J.P., Choi, W.S.: A taxonomy of computer security flaws. ACM Computing Surveys **26** (1994) http://chacs.nrl.navy.mil/publications/CHACS/1994/1994landwehr-acmcs.pdf, accessed 12 June 2002.
A note taken from the text published on the web: 'As revised for publication in ACM Computing Surveys 26, 3 (Sept., 1994). This version, prepared for electronic distribution, reflects final revisions by the authors but does not incorporate Computing Surveys´ copy editing. It therefore resembles, but differs in minor details, from the published version. The figures, which have been redrawn for electronic distribution are slightly less precise, pagination differs, and Table 1 has been adjusted to reflect this'.
25. (CVE) http://cve.mitre.org/about/terminology.html, accessed 4 July 2002.
26. (CVE) http://cve.mitre.org/about/index.html, accessed 24 June 2002.
27. MITRE: (The Early Years) http://www.mitre.org/about/history.shtml, accessed 12 June 2002.
28. Icove, D., Seger, K., VonStorch, W.: Computer Crime: A Crimefighter's Handbook. O´Reilley & Associates Inc, Sebastopol, CA (1995)

29. Anonymous: Maximum Security – A Hacker's Guide to Protecting Your Internet Site and Network. 2 edn. Sams Publishing (1998)
30. Stallings, W.: Cryptography and Network Security, Principles and Practice. 2 edn. Prentice Hall (1999)
31. (CVE) http://cve.mitre.org/docs/docs2000/naming_process.html, accessed 12 June 2002.
32. (ICAT) http://icat.nist.gov/icat.cfm, accessed 12 June 2002.
33. (ICAT) http://icat.nist.gov/icat_documentation.htm, accessed 27 September 2002.
34. Dittrich, D.: The "stacheldraht" distributed denial of service attack tool. (1999) http://staff.washington.edu/dittrich/misc/stacheldraht.analysis, accessed 24 July 2002.
35. Dittrich, D.: The DoS Project's "trinoo" distributed denial of service attack tool. (1999) http://staff.washington.edu/dittrich/misc/trinoo.analysis, accessed 24 July 2002.
36. eEye Digital Security: .ida "Code Red" Worm. (2001) http://www.eeye.com/html/Research/Advisories/AL20010717.html, accessed 13 September 2002.
37. Chien, E.: CodeRed Worm, Symantec. (2002) http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html, accessed 18 July 2002.
38. Trend Micro: CODERED.A. (2001) http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=CODERED.A&VSect=T, accessed 24 July 2002.

A Step towards Quantification of IT Security
Extended abstract

Amund Hunstad <*amund@foi.se*> Jonas Hallberg Anna Stjerneby

Dept. of Systems Analysis and IT Security
Swedish Defence Research Agency

The concept of a networked society by default results in widely distributed information systems that are difficult to control or even comprehend. To be able to comprehend these widely distributed information systems, efficient modeling techniques are needed. The evaluation of the IT-security ability of systems requires quantitative efficiency measures for the security-enabling mechanisms. Efficient evaluation enables novel design methods and design support tools.

Thus, the identification of security-related system characteristics is required: 1) to support the formulation of the security requirements of systems and the transformation of them into a policy; 2) as a base and support for the design process, system modeling, implementation, and evaluation; and 3) to facilitate development of quantitative security measures of systems and subsystems.

To identify a set of relevant characteristics four tasks were carried out. These tasks are: 1) a literature study, to detect what the state of the art is; 2) a structured analysis, to obtain a suitable structure for further analysis and detect additional characteristics; 3) a brainstorm session, to obtain material for extension of the set of characteristics; and 4) a cross checking, to verify the suitability of the whole structure and its contents. The structured analysis used a tree structure as illustrated by Figure 1. The whole process resulted in a structure with 55 distinct characteristics[1].

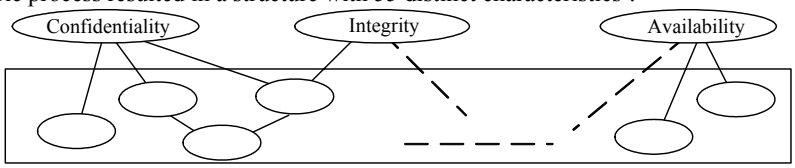

Figure 1: Security characteristics in a tree structure.

In a fully developed tree, all the leaves should be attributes that can be assigned a value. This has not been fully accomplished within the scope of this work. Thus, further refinement of characteristics is necessary. However, the development of the tree should be a continuous process where some branches grow while others are cut off to accommodate the dynamics of system development and IT security. The current graph can be used for all the purposes listed above, i.e. it can serve as a base for (1) the formulation of security requirements, (2) system modeling techniques and design methods and tools, and (3) the refinement of security characteristics to reach levels where quantifiable characteristics appear.

---

[1] Stjerneby, A. (2002). *Identification of security relevant characteristics in distributed information systems*. Master's Thesis. LiTH-ISY-EX-3278-2002. Linköpings universitet.

Using Computer Games in IT Security Education

Analyses and Continuation

Kjell Näckros

Department of Computer and Systems Sciences,
Stockholm University and Royal Institute of Technology, Sweden
email: kjellna@dsv.su.se
Url: http://www.dsv.su.se/~kjellna

Abstract. A general holistic understanding of information security and privacy issues is vital for the individual as well as for the society. Ambiguities concerning user's privacy, integrity and confidentiality are major obstacles towards a sound and functional electronic society. System designers ought to pay more attention to these threats in order to support the creation of reliable trust between consumers/users/producers. To empower the average computer user to gain control of their information and communication technologies (ICT) they need to have the necessary ICT security knowledge. Since new user groups, with different kind of learning capabilities are emerging, in conjunction with an overload of information there is a need for alternative approaches to offer the necessary knowledge. This article discusses an ongoing research on visualising security aspects using computer games as a complement to the conventional linear instruction. Based on the findings from a series of experiments evaluating a computer game's impact on learning ICT security[1,2], the continuation towards an applicable instructional framework of general ICT security understanding is discussed. The findings show that many computer users understand ICT security more thoroughly when a non-linear computer game is used than when reading a linear text. Additional experiments, investigating the applicability of the knowledge acquired from the non-linear instruction, will be conducted during 2002-2003. The ICT-Sec instruction will also be more generalisable, scaleable and adaptable towards mobile computing.

[1] This research and the result from the preliminary study were presented at Nordsec2000 [Näckros, K. (2000). *Using Computer Games in IT Security Education - preliminary results of a study.* pp.251-258] and discussed at Wise2 [Näckros, K. (2001). *Game-Based Learning within IT Security Education.* Wise2: IFIP TC11 WG11.8 pp.243-260].

[2] The research is part of a Licentiate thesis presented in December 2001 [Näckros, K. (2001). *Game-Based Instruction within IT Security Education.* Department of Computer and Systems Sciences(DSV), Report Series: No-01-018-DSV-SU. Kista, Sweden, Stockholm University (SU)/ Royal Institute of Technology(KTH)].