Gilles Barthe
Benjamin Livshits
Riccardo Scandariato (Eds.)

Engineering Secure Software and Systems

LNCS 7159

4th International Symposium, ESSoS 2012
Eindhoven, The Netherlands, February 2012
Proceedings

Springer

Lecture Notes in Computer Science 7159

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Gilles Barthe Benjamin Livshits
Riccardo Scandariato (Eds.)

Engineering Secure Software and Systems

4th International Symposium, ESSoS 2012
Eindhoven, The Netherlands, February, 16-17, 2012
Proceedings

Springer

Volume Editors

Gilles Barthe
Universidad Politecnica de Madrid, Fundación IMDEA Software
Facultad de Informática, Campus Montegancedo
28660 Boadilla del Monte, Madrid, Spain
E-mail: gjbarthe@gmail.com

Benjamin Livshits
Microsoft Research
One Microsoft Way, 98052-6399 Redmond, WA, USA
E-mail: livshits@microsoft.com

Riccardo Scandariato
Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, 3001 Heverlee, Belgium
E-mail: riccardo.scandariato@cs.kuleuven.be

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-28165-5 e-ISBN 978-3-642-28166-2
DOI 10.1007/978-3-642-28166-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012930018

CR Subject Classification (1998): C.2, E.3, D.4.6, K.6.5, J.2

LNCS Sublibrary: SL 4 – Security and Cryptology

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

It is our pleasure to welcome you to the fourth edition of the International Symposium on Engineering Secure Software and Systems.

This unique event aims at bringing together researchers from software engineering and security engineering, which might help to unite and further develop the two communities in this and future editions. The parallel technical sponsorship from the ACM SIGSAC (the ACM interest group in security) and ACM SIGSOFT (the ACM interest group in software engineering) is a clear sign of the importance of this interdisciplinary research area and its potential.

The difficulty of building secure software systems is no longer focused on mastering security technology such as cryptography or access control models. Other important factors include the complexity of modern networked software systems, the unpredictability of practical development life cycles, the intertwining of and trade-off between functionality, security and other qualities, the difficulty of dealing with human factors, and so forth. Over the last few years, an entire research domain has been building up around these problems.

The conference program include two major keynotes from Cristian Cadar (Imperial College London) on improving software reliability and security via symbolic execution and Thorsten Holz (Ruhr University Bochum) on an overview of modern security threats, and an interesting blend of research and idea papers.

In response to the call for papers, 53 papers were submitted. The Program Committee selected seven contributions as research papers (13%), presenting new research results in the realm of engineering secure software and systems. It further selected seven idea papers, which gave crisp expositions of interesting, novel ideas in the early stages of development.

Many individuals and organizations contributed to the success of this event. First of all, we would like to express our appreciation to the authors of the submitted papers and to the Program Committee members and external referees, who provided timely and relevant reviews. Many thanks go to the Steering Committee for supporting this and future editions of the symposium, and to all the members of the Organizing Committee for their tremendous work and for excelling in their respective tasks. The DistriNet research group of the K.U. Leuven did an excellent job with the website and the advertising for the conference. Finally, we are also grateful to Andrei Voronkov for his EasyChair system.

We owe gratitude to ACM SIGSAC/SIGSOFT, IEEE TCSP and LNCS for supporting us in this new scientific endeavor.

December 2011

Gilles Barthe
Benjamin Livshits
Riccardo Scandariato

Conference Organization

General Chair

Sandro Etalle	Eindhoven University of Technology, The Netherlands

Program Co-chairs

Gilles Barthe	IMDEA Software Institute, Spain
Ben Livshits	Microsoft Research, USA

Publication Chair

Riccardo Scandariato	Katholieke Universiteit Leuven, Belgium

Publicity Chair

Pieter Philippaerts	Katholieke Universiteit Leuven, Belgium

Local Arrangements Co-chairs

Jerry den Hartog	Eindhoven University of Technology, The Netherlands
Jolande Matthijsse	Eindhoven University of Technology, The Netherlands

Steering Committee

Jorge Cuellar	Siemens AG, Germany
Wouter Joosen	Katholieke Universiteit Leuven, Belgium
Fabio Massacci	Università di Trento, Italy
Gary McGraw	Cigital, USA
Bashar Nuseibeh	The Open University, UK
Daniel Wallach	Rice University University, USA

Program Committee

Davide Balzarotti	Eurecom, France
Gilles Barthe	IMDEA Software Institute, Spain
David Basin	ETH Zurich, Switzerland
Hao Chen	UC Davis, USA

Manuel Costa	Microsoft Research, USA
Julian Dolby	IBM Research, USA
Maritta Heisel	University of Duisburg-Essen, Germany
Thorsten Holz	Ruhr-Universität Bochum, Germany
Collin Jackson	Carnegie Mellon University
Martin Johns	SAP Research - CEC Karlsruhe, Germany
Jan Jürjens	TU Dortmund and Fraunhofer ISST, Germany
Engin Kirda	Eurecom, France
Ben Livshits	Microsoft Research, USA
Javier Lopez	University of Malaga, Spain
Sergio Maffeis	Imperial College London, UK
Heiko Mantel	TU Darmstadt, Germany
Fabio Martinelli	IIT-CNR, Italy
Haris Mouratidis	University of East London, UK
Anders Møller	Aarhus University, Denmark
Frank Piessens	K.U. Leuven, Belgium
Erik Poll	Radboud Universiteit Nijmegen, The Netherlands
Pierangela Samarati	Università di Milano, Italy
Ketil Stølen	SINTEF, Norway
Laurie Williams	North Carolina State University, USA
Jianying Zhou	Institute for Infocomm Research, Singapore

External Reviewers

Aderhold, Markus	Jawurek, Marek
Aizatulin, Misha	King, Jason
Anguraj, Baskar	Krautsevich, Leanid
Beckers, Kristian	Lazouski, Aliaksandr
Bhargavan, Karthikeyan	Lekies, Sebastian
Brucker, Achim D.	Li, Yan
Chu, Cheng-Kang	Lund, Mass Soldal
Costa, Gabriele	Lux, Alexander
De Capitani Di Vimercati, Sabrina	Morrison, Pat
Dipietro, Roberto	Moyano, Francisco
Dupressoir, Francois	Nieto, Ana
Erdogan, Gencer	Nikiforakis, Nick
Ereth, Sarah	Nuñez, David
Francis, Patrick	Pape, Sebastian
Frank, Mario	Perner, Matthias
Gay, Richard	Petrocchi, Marinella
Havaldsrud, Tormod	Philippaerts, Pieter
Helms, Eric	Pironti, Alfredo
Huang, Xinyi	Ruhroth, Thomas
Humberg, Thorsten	Schlaepfer, Michael

Schmidt, Benedikt
Schmidt, Holger
Seehusen, Fredrik
Sgandurra, Daniele
Slankas, John
Smart, Nigel
Smith, Ben
Smyth, Ben
Snipes, Will
Solhaug, Bjørnar
Sprenger, Christoph
Starostin, Artem
Torabi Dashti, Mohammad
Van Acker, Steven
Vullers, Pim
Weinberg, Zack
Yskout, Koen

Table of Contents

Application-Replay Attack on Java Cards: When the Garbage Collector Gets Confused

Guillaume Barbu[1,2], Philippe Hoogvorst[1], and Guillaume Duc[1]

[1] Institut Télécom / Télécom ParisTech, CNRS LTCI,
Département COMELEC,
46 rue Barrault, 75634 Paris Cedex 13, France
[2] Oberthur Technologies, Innovation Group,
Parc Scientifique Unitec 1 - Porte 2,
4 allée du Doyen George Brus, 33600 Pessac, France

Abstract. Java Card 3.0 specifications have brought many new features in the Java Card world, amongst which a true garbage collection mechanism. In this paper, we show how one could use this specific feature to predict the references that will be assigned to object instances to be created. We also exploit this reference prediction process in a combined attack. This attack stands as a kind of "application replay" attack, taking advantage of an unspecified behavior of the Java Card Runtime Environment (JCRE) on application instance deletion. It reveals quite powerful, since it potentially permits the attacker to circumvent the application firewall: a fundamental and historical Java Card security mechanism. Finally, we point out that this breach comes from the latest specification update and more precisely from the introduction of the automatic garbage collection mechanism, which leads to a straightforward countermeasure to the exposed attack.

Keywords: Java Card, Combined Attack, Garbage Collection, Application Firewall.

1 Introduction

To follow the emergence of new communication technologies, new Java Card specifications have recently been released: Java Card 3.0. This new standard comes in two editions: *Classic* and *Connected*. The *Classic* edition stands as an evolution of previous versions of Java Card. The *Connected* edition represents the real novelty of this version, adding numerous features in the Java Card world such as an embedded web server, the multithreading support, enhanced security policy facilities, an extended API (Application Programming Interface). All along this paper, we will focus on one of the new features introduced by the Java Card 3.0 specifications: a true automatic garbage collection mechanism. This particular feature is one of the rare novelty to be present in both editions of the specifications.

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 1–13, 2012.

The contribution of this paper is twofold. First, we introduce the concept of reference prediction, taking advantage of the garbage collection. In addition, we show how an attacker with fault injection capacity could, under certain assumptions, use this concept to circumvent the application firewall through a so-called replay attack.

The remainder of this paper is organized as follows: In Section 2, we introduce the principle of reference prediction on Java Card platforms. In Section 3, we describe the application firewall mechanism and expose the application-replay attack under a given implementation assumption. Finally, we discuss the issues raised by the predictability of object references and the different countermeasures that could be implemented in Section 4.

2 Java Card Reference Prediction

This section introduce the notions of Java reference and garbage collection and state the assumptions under which this work is based. Finally, we describe the process we put into practice to achieve this prediction on the tested platforms.

2.1 Reference Assignment

The Java Card Virtual Machine (JCVM) aims at providing an abstraction layer between the hardware device and Java Card applications. This abstraction is the basement of the *write once - run everywhere* philosophy of the Java language. On object instantiation, the JCVM is then responsible for allocating the memory to store this object and assigning it a Java reference, possibly the allocated memory address or a value abstracting this address. Regardless of its exact implementation, we assume in the remainder of this article that these Java references are assigned following a straightforward linear process. That is to say, the next reference to be assigned is the smallest reference that is not already assigned.

Formally, with $(r_i)_{1 \leq i \leq n}$ the previously allocated references, a new reference r_{n+1} is allocated such that:

$$r_{n+1} = \min\{r_i \text{ s.t. } \forall\ r_j < r_i,\ r_j \text{ is used}\} \tag{1}$$

We believe this assumption is not very restrictive since we have successfully tested it on different cards from different manufacturers and different versions of Java Card with the method given in Section 2.3.

For the sake of simplicity we will consider in the remainder of this article that a reference is a value abstracting the physical address of an object. Otherwise, the method would need to be adapted, taking into account the size of each object instance.

2.2 Garbage Collection

The principle of garbage collection is not a novelty, even in the Java Card context. Indeed, Java Card 2.2 proposes (optionally) a memory reclaiming process through the method `JCSystem.requestObjectDeletion()` [1]. However, this method only schedules the object deletion service prior to the next invocation of the `Applet.process()` method. That is to say, unreferenced objects are not actually deleted on the method's call.

The real novelty in the latest version of the Java Card standard is that garbage collection is automatically triggered when memory space becomes insufficient, or on specific event such as card reset for instance. Furthermore, the `System.gc()` [2] method can be called at any time within an application and runs the garbage collector. Unlike the `JCSystem.requestObjectDeletion()` method, when control returns from the `System.gc()` method, the garbage collector should have actually been executed and reclaimed unused memory.

We will not go into further details in the garbage collection mechanism and will only consider that the garbage collector implementation ensures that it will reclaim the memory used by objects that are not accessible anymore (unreferenced). The important point to bare in mind is rather the evolution of the Java Card specifications regarding this functionality.

2.3 Reference Prediction

We can now introduce one of the contribution of our work, the reference prediction process. As this process relies on a particular type confusion, we recall the previous works achieved on this particular topic before giving a complete description of the process.

Previous Works on Type Confusion. Type safety is one of the cornerstone of the Java language security. Most of the literature presenting potential attacks on Java Card (or even on Java SE [3]) use type confusion at some point in the attack path.

Until 2009, attackers have to count on bugs on specific mechanisms or to load an ill-formed application thanks to `.CAP` file manipulation to provoke a type confusion [4, 5, 6]. The release of the Java Card 3.0 Connected Edition has rendered this path theoretically impracticable making the On-Card Bytecode Verification (OCBV) of application mandatory. The use of fault attacks has then emerged as a quite efficient technique to reach this point, as exposed in a couple of recent publications [7, 8, 9, 10, 11, 12].

In [7], Barbu et al. describe a way to forge object's reference thanks to a type confusion (*e.g.* `Object o = 0x12345678;`). For that matter, they achieve a physical fault injection during a `checkcast` execution in order to render it successful and provoke a type confusion between two instances of different classes. The first class holding an `Object` class field, and the other an integral class

field (`short` or `int` depending on the size of an object's reference). Getting the reference of an object is then as easy as reading an integral field. Similarly, forging the reference of the `Object` field is then as easy as assigning a value to the integral field.

In the remainder of this paper we will consider that an adversary has the ability to read and forge references.

How to Get the Reference of an Object ?. The aim of this section is to expose a process to predict the values by which object instances to be created will be referred to. As the previous section lets guess, this process involves the memory allocation and reclaiming mechanisms. But the first requirement for this process is to be able to learn the value of an object's reference.

In the context of Java Card 3.0, this question is answered within the API specification [2] (at least, one answer is suggested).

QUOTE 1. *"As much as is reasonably practical, the `hashCode` method defined by class `Object` does return distinct integers for distinct objects. (This is typically implemented by converting the internal address of the object into an integer, but this implementation technique is not required by the Java™ programming language.)"*

This suggestion can be argued by the fact that two instances of the `Object` class will only differ by their internal addresses (or Java references if these two concepts are not merged within the considered platform). It is then consistent to use it to distinguish such objects.

On Java Card 2.x.y platforms (as well as on Java Card 3.0 platforms that do not implement the `hashCode` method as suggested) the following approach including type confusion has to be considered as described in Listing 1.1.

Assuming that it is possible to learn the reference of an object instance appears then reasonable.

How to Predict the Reference of an Object ?. Under the linear reference assignment assumption, the prediction process is described by Algorithm 1.

With regards to (1), on step 1 the allocated reference r_k is s.t.

$$r_k = \min\{r_i \text{ s.t. } \forall\ r_j < r_i,\ r_j \text{ is used}\} \tag{2}$$

After, step 2 and 3, we know that r_k is not used anymore.
The following allocated reference r_l will be s.t.

$$r_l = \min\{r_i \text{ s.t. } \forall\ r_j < r_i,\ r_j \text{ is used}\} \tag{3}$$

Consequently, putting (2) and (3) together, we have,

$$r_l = r_k \tag{4}$$

The successive object instantiation and deletion allow us to discover the next available reference, since it is the one that has just been released. Actually,

Listing 1.1. Getting the reference of an object

```
/**
 * Class A holds an Object field: o
 * Class B holds an integral field: ref
 * A a and B b are public fields of the class.
 */
public void initConfusion() {
    A a = new A();
    // Need a fault injection at runtime to avoid
    // a ClassCastException throwing.
    B b = (B) (Object) a;
}
public short getReference(Object o) {
    // Type confusion has "merged" a.o and b.ref
    a.o = o;
    return b.ref;
}
```

Algorithm 1: ReferencePrediction()

0 Delete current unreachable object instances: `System.gc();`

1 Create a new object instance: `Object o = new Object();`

2 Get the reference of this instance: `ref = getReference(o);`

3 Make this object unreachable: `o = null;`

4 Delete this unreferenced object: `System.gc();`

5 The next assigned reference will be `ref`

this behavior has been previously observed by Hogenboom et al. [13] in the context of another mechanism leading to memory reclaiming on a Java Card 2.1.1: transaction aborting.

This can be easily tested on any platform supporting the Java Card 3 specifications by running the code in Listing 1.2.

Under the assumptions previously stated, we can now consider that we are able of reading/writing the reference of an object, but also that we can predict the reference of future object instances.

Listing 1.2. Testing the prediction process

```
System.gc();            // First call to the garbage collector to
                        // delete current unreachable references.

o1 = new Object();      // Assign a new reference to o1 and store
h1 = o1.hashCode();     // the value of this reference in h1.

o1 = null;              // Set o1 to null and call the garbage
System.gc();            // collector to actually delete it.

o2 = new Object();      // Assign a new reference to o2 and store
h2 = o2.hashCode();     // the value of this reference in h2.

if (h1 == h2)           // Compare stored references...
    // The assumption is verified.
else
    // The assumption is not verified.
```

3 Application Replay to Circumvent the Application Firewall

This section exposes how the ability of predicting and forging references can be used to circumvent the application context isolation through a so-called replay attack.

3.1 Java Card Context Isolation : The Application Firewall

Java platforms (Java Standard Edition and Java Micro Edition for instance) usually execute one application per virtual machine instance. One difference of the Java Card platform is that it executes a single instance of the virtual machine. Therefore, it must ensure each hosted application that other applications will not access its own data or code within this single virtual machine instance. For that purpose, the specifications mandate the implementation of an application firewall, isolating each application as well as the Java Card Runtime Environment. Figure 1 depicts the context isolation mechanism and the firewall crossing permission in the platform.

Application Firewall Implementation. A possible implementation of the context isolation would be to assign each application group a context identifier. When an application creates an object, this object would then inherit the context identifier of its "maker". Then access across context can be easily checked by comparing the accessing context identifier and the accessed context identifier. If the context identifiers are matching, access is granted, else the firewall deny the access and a `SecurityEcxeption` is thrown.

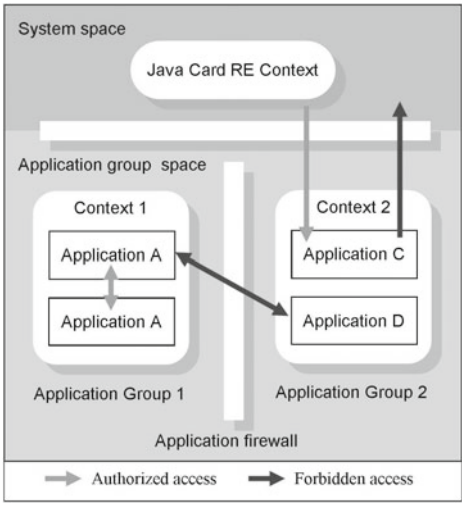


Fig. 1. Contexts within the Java Card platform's object system (as per [14])

Such an implementation appears quite suitable in a constrained system. It does not consume too much memory (one 8/16/32-bit word per object to protect it), the access decision is simple (a word comparison) and it does not constrain the number of objects an application can hold. Actually, some experiments based on ill-formed applet loading on Java Cards 2.X.Y from different card manufacturers and on the C reference implementation provided within the Java Card 2.2.2 Development Kit[1] has proven this implementation is (at least has been) used. We will consider such an implementation in the remainder of this article.

This implementation choice leads to a first question:

Question 1. Where does the context identifier comes from ?

Many answers could be given to that question (a random value, an internal counter, the hash of that application's name, ...). However, the important thing is to ensure that two application instances living at the same time in the card does not have the same context identifier. Hence the answer to this question has only a limited interest from an attacker's point of view. We will come back to that point in Section 3.3.

3.2 Application Instance Deletion

Java Card platforms allow the post-issuance loading of application. With this capacity comes also that of unloading application. To sum up, an application will then go through the following cycle during its life:

[1] JCDK available at `http://download.oracle.com/otn-pub/java/java_card_kit/2.2.2/java_card_kit-2_2_2-linux.zip`

1. Application module loading.
2. Application instance creation.
3. Application execution.
4. Application instance deletion.
5. Application module unloading.

The step we are particularly interested in is the application instance deletion. In order not to depend on any implementation specific mechanism, we will only consider this process according to the specifications.

Java Card 2.2.2. On Java Card 2.2.2, applet instance deletion is processed by an entity referred to as the Applet Deletion Manager (ADM). The behavior of the ADM is specified in the JCRE specification. In particular, it is stated that:

QUOTE 2. *"Applet instance deletion involves the removal of the applet object instance and the objects owned by the applet instance and associated Java Card RE structures."*

Consequently, all objects owned by the applet instance should be actually deleted within the deletion process.

Java Card 3.0. On Java Card 3.0, applet instance deletion is processed by the Card Management Facility (CMF). The behavior of the CMF is specified in the JCRE specification. In particular, it is stated that:

QUOTE 3. *"An application instance is successfully deleted when all objects owned by the application instance are inaccessible on the card."*

QUOTE 4. *"Upon successful deletion, [the card management facility] fires an application instance deletion event -* `event:///standard/app/deleted` *- with the application instance URI as the event source to all registered listeners."*

The important point to notice here is that what happens between the application instance deletion, the event firing and the actual notification of potential event listeners is not addressed in the specifications. This is indeed the starting point of the attack described in the following section.

3.3 "Application-Replay" Attack on a Java Card 3.0

The previous section has highlighted a difference between the application instance deletion processes on Java Card 2.2.2 and 3.0. Indeed the later does not mandate that object instances belonging to an application instance are deleted together with the application instance. This lead us to think that the mandatory deletion of objects on Java Card 2.2.X has not been thought as a security mechanism, but rather as a functional one. Actually, since the garbage collection is not mandatory on Java Card 2.2.X platforms, one has to explicitly delete objects that are not used anymore. Else they will still be consuming memory for the whole card lifetime. This explains the disappearing of this statement in the

Java Card 3.0 specifications since the garbage collector ensures that those objects will be deleted eventually. The application-replay attack detailed hereafter is then limited to Java Card 3.0 platforms.

The remainder of this section describe a possible attack scenario divided into two steps:

- First, the attacker needs to prevent the deletion of the targeted application's objects;
- Then, the attacker must find a way to access these objects despite the application firewall.

Illegal Memory Consumption. The aim of this first step of the attack is, for the adversary's application, to gain references to objects belonging to the targeted application, even though this application gets deleted.

Let us first put together the different pieces of information gleaned so far. We know from *Quote 3* that the CMF should consider an application instance deletion successful when all objects it owned are inaccessible. This means these objects are garbage-collectable, but not necessarily that they have been garbage-collected. In addition, we know from *Quote 4* that the CMF will fire an event on successful deletion. Finally, we know how to predict and forge object references from Section 2.

Bounding Object Instances Owned by Another Application Instance. Consider now two applications called `Forgery` and `Target`, respectively the adversary's and the targeted application. We assume that `Forgery` is loaded and instantiated. That is to say, its binary representation is on-card and it has been initialized. On the other hand, we assume that that `Target` is only loaded. That is to say its binary representation is on-card but it has not been initialized. Furthermore, we let `Forgery` register an event listener to be notified of application instance deletions (say on the URI `event:///standard/app/deleted/*`).

The `Forgery` application instance can then guess the starting and ending bounds of the `Target` application instance to be, following these steps:

1. Call the garbage collector.
2. Predict the next reference (let us call it `start`).
3. Let `Target` be instantiated.
4. Instantiate an object to get the "current" reference and deduce the last reference instantiated by `Target` (let us call it `end`).

The `Forgery` application then knows that the references of `Target`'s objects are s.t. $\forall r_i \in \texttt{Target}, \texttt{start} \leq r_i \leq \texttt{end}$.

Preventing the Deletion of Objects on Application Instance Deletion. The attacker can then request the `Target` application instance deletion. During the deletion process, the card management facility will ensure that all objects belonging to this application instance are not referenced anymore. We emphasize the fact that these objects are not necessarily deleted as long as the garbage collector is not executed.

On notification of the application instance successful deletion, the `Forgery` application can then forge object's references in an array of `end − start` objects to values between `start` and `end − 1`. This is achieved through a type confusion similar to that exposed in Listing 1.1, considering the confused object is an instance field[2]. By doing so, the attacker prevents these objects from being actually garbage collected, so-to-speak consuming their references.

At this point, the attacker's application instance hold references to objects that do not belong to it. Trying to access these objects in that application would then irremediably lead to a `SecurityException` throwing. The following section adapts the principle of the replay attack to overcome this.

Application Firewall Circumvention. The adversary's application holds references belonging to a deleted application instance. The only way to access these references would then be to collaborate with a new application instance impersonating the deleted one.

It becomes obvious now that the answer to *Question* 1 would then only be useful to help answering the real critical question:

Question 2. Can a new application instance be given the same context identifier as a former (deleted) application instance ?

If we cannot give an accurate answer to *Question* 1 without knowing the exact implementation of the platform, this last question could be answered by experimentation. Given that no Java Card 3.0 platforms have been publicly released so far we have not been able to test this particular behavior on various Java Card 3.0 platforms. Nevertheless, we have run our experimentation on different cards implementing different versions of the Java Card 2 specifications with mostly positive results. Let us assume now that the answer to the last question was "Yes".

The attacker would then only have to instantiate a new application, "send" the forged objects from `Forgery` to that new application and try to access them. This operation can be repeated until no `SecurityException` is thrown, which means that the new application has been assigned the same context identifier as the original `Target` application instance. That is to say, the new application impersonates the previous `Target`'s application instance.

The last difficulty resides in the "sending" of the forged objects from `Forgery` to the new application, since `Forgery` is not authorized to use these objects by the application firewall. This is why we considered in 3.3 an array of forged objects (the array itself is then still legally usable by `Forgery`). A mere library permits then to store this array from `Forgery` and access its content from the new application without having to pass through the application firewall.

Eventually, the new application instance has then full access to the objects created by the `Target`'s application instance. The application firewall has been circumvented.

[2] Consequently a single fault injection is necessary for all reference forgeries.

So far, this article has proven the possibility to circumvent the Java Card application firewall, under certain assumptions. Nevertheless, the following section shows that this attack can be thwarted with an adequate implementation.

4 Analysis and Countermeasures

The attack describe in the previous section relies on two key elements:

- a "lazy" application instance deletion process.
- the attacker's ability to provoke a type flaw and to forge an object's reference.

Object Deletion. This basement of our attack lies in *Quote* 3, *i.e.* the card manager only ensures that objects owned by the application instance to be deleted are not accessible anymore. In a way, the specifications encourage implementors to give in to the temptation to rely on automatic garbage collection for the effective deletion of these objects.

Thus, it is assumed that the garbage collection will be executed later and that it will delete all inaccessible objects. But between the successful deletion event is fired and the next garbage collection is requested, many things can happen, as exposed within the previous section.
It appears then necessary that the application instance deletion process ensures not only that the objects previously owned by the application to be deleted are inaccessible but also that they are actually deleted when the application instance is deleted. This is indeed what prevents the attack from succeeding on Java Card 2.X platforms, although it seems to us that this has not been specified to enforce security.

This possible breach may be easily taken care of within the implementation of the CMF. Nevertheless, JC3 platforms intending to be considered as secure systems, we go as far as to recommend that mandatory deletion of objects belonging to a deleted application instance should be added in the next update of the Java Card 3.0 specifications.

Ensuring Type Safety. Yet, the type confusion is the technical root of our attack. Without the type confusion, we would have not been able to recover the undeleted objects. Actually, even on platforms supporting OCBV, such a type flaw can be caused by various fault injections. Barbu et al. take advantages of a faulty `checkast` execution, while Vetillard et al. manages to turn an instruction into a `nop`, thus making an instruction from a parameter and provoking an early method return. At CARDIS'11, Barbu et al. presented another way to provoke a type confusion through a faulty operand stack as well as a countermeasure to ensure the operand stack integrity. However this does not prevent the success of the previously cited attacks. Furthermore, other paths to type confusion might be discovered.

The study of the different ways to provoke a type confusion through physical perturbations, as well as the design of countermeasures ensuring type safety in the presence of faults is an ongoing work.

5 Conclusion

In this paper, we have introduced the principle of reference prediction on Java Card platforms and exposed an attack based on the Java Card 3.0 specifications leading to the circumvention of the application firewall.

This work has been based on several assumptions concerning the implementation of the attacked platform which implicitly sketches possible countermeasures (type safety enforcement, actual deletion of objects, unpredictable reference assignment, uniqueness of firewall identifiers). Although we did not put into practice the complete attack path on a Java Card 3.0 platform, we have been able to test successfully the different assumptions required to achieved it on Java Card 2.X platforms from different origins.

Finally, this work outlines a possible weakness in the Java Card 3.0 specifications. Although the exposed attack scenario may appear unlikely on the field, we believe it should be taken into consideration in a future update of the specifications.

References

1. Sun Microsystems Inc.: Application Programming Interface, Java Card Platform Version 2.2.2 (2006)
2. Sun Microsystems Inc.: Application Programming Interface, Java Card Platform Version 3.0.1 Connected Edition (2009)
3. Govindavajhala, S., Appel, A.W.: Using Memory Errors to Attack a Virtual Machine. In: SP 2003: Proceedings of the 2003 IEEE Symposium on Security and Privacy, Washington, DC, p. 154 (2003)
4. Witteman, M.: Java Card Security. Information Security Bulletin 8, 291–298 (2003)
5. Mostowski, W., Poll, E.: Malicious Code on Java Card Smartcards: Attacks and Countermeasures. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 1–16. Springer, Heidelberg (2008)
6. Iguchi-Cartigny, J., Lanet, J.L.: Developping a Trojan Applet in a Smart Card. Journal in Computer Virology (2010)
7. Barbu, G., Thiebeauld, H., Guerin, V.: Attacks on Java Card Combining Fault and Logical Attacks. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 148–163. Springer, Heidelberg (2010)
8. Vetillard, E., Ferrari, A.: Combined Attacks and Countermeasures. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 133–147. Springer, Heidelberg (2010)
9. Sere, A., Lanet, J.L., Iguchi-Cartigny, J.: Checking the Paths to Identify Mutant Application on Embedded Systems. In: Kim, T.-h., Lee, Y.-h., Kang, B.-H., Ślęzak, D. (eds.) FGIT 2010. LNCS, vol. 6485, pp. 459–468. Springer, Heidelberg (2010)
10. Barbu, G., Duc, G., Hoogvorst, P.: Java Card Operand Stack: Fault Attacks, Combined Attacks and Countermeasures. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 297–313. Springer, Heidelberg (2011)

11. Bouffard, G., Iguchi-Cartigny, J., Lanet, J.L.: Combined Software and Hardware Attacks on the Java Card Control Flow. In: Prouff, E. (ed.) CARDIS 2011. LNCS, vol. 7079, pp. 283–296. Springer, Heidelberg (2011)
12. Sere, A., Lanet, J.L., Iguchi-Cartigny, J.: Evaluation of Countermeasures Against Fault Attacks on Smart Cards. International Journal of Security and Its Applications (5), 49–61
13. Hogenboom, J., Mostowski, W.: Full memory read attack on a java card. In: 4th Benelux Workshop on Information and System Security Proceedings, WISSEC 2009 (2009)
14. Sun Microsystems Inc.: Runtime Environment Specification, Java Card Platform Version 3.0.1 Connected Edition (2009)

Supporting the Development and Documentation of ISO 27001 Information Security Management Systems through Security Requirements Engineering Approaches*

Kristian Beckers[1], Stephan Faßbender[1], Maritta Heisel[1], Jan-Christoph Küster[2], and Holger Schmidt[1]

[1] paluno - The Ruhr Institute for Software Technology University of Duisburg-Essen
firstname.lastname@paluno.uni-due.de
[2] Fraunhofer Institut for Software and Systems Engineering ISST
Jan-Christoph.Kuester@isst.fraunhofer.de

Abstract. Assembling an information security management system according to the ISO 27001 standard is difficult, because the standard provides only sparse support for system development and documentation.

We analyse the ISO 27001 standard to determine what techniques and documentation are necessary and instrumental to develop and document systems according to this standard. Based on these insights, we inspect a number of current security requirements engineering approaches to evaluate whether and to what extent these approaches support ISO 27001 system development and documentation. We re-use a conceptual framework originally developed for comparing security requirements engineering methods to relate important terms, techniques, and documentation artifacts of the security requirements engineering methods to the ISO 27001.

Keywords: Security standards, requirements engineering, ISO27000, ISO27001, compliance, security.

1 Introduction

Aligning organizations to meet security demands is a challenging task. Security standards, e.g. the ISO 27000 series of standards, offer a way to attain this goal. The normative standard of the aforementioned series, the ISO 27001, contains the requirements for an *Information Security Management System (ISMS)* [1]. The standard prescribes a process, which tailors security to the needs of any kind of organization. The remaining standards of the ISO 27000 series describe parts,

* This research was partially supported by the EU project Network of Excellence on Engineering Secure Future Internet Software Services and Systems (NESSoS, ICT-2009.1.4 Trustworthy ICT, Grant No. 256980).

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 14–21, 2012.

or usage scenarios, of the ISMS in detail [1]. For example, the ISO 27005 [2] describes information security risk management. The ISO 27005 has a certain significance as the ISO 27001 is risk-centered in many sections, and the ISO 27005 describes the risk assessment process and the risk documentation and management in detail. However, the ISO 27005 is not normative.

The ISMS consists of processes, procedures, and resources that can be software. Sparse descriptions in the standard are a problem during the establishment of an ISMS. For example, the required input for the *scope and boundaries* description is to consider "characteristics of the business, the organization, its location, assets and technology"[3, p. 4].

Moreover, the standard does not provide a method for assembling the necessary information or a pattern on how to structure that information.

Security requirements engineering (SRE) methods, on the other hand, provide structured elicitation and analysis of security requirements. SRE methods can be part of the early phases of a given software development process. However, we propose not to limit SRE methods to software development. The structured elicitation and analysis of security requirements of SRE methods is also useful for different security engineering contexts. Therefore, we propose to use SRE methods to support security engineers in the development and documentation of an ISMS, compliant to ISO 27001. In addition, the ISMS is a process for security that may also rely on secure software. Thus, SRE methods can also support software engineers in building secure software for an ISMS.

Our work addresses the research question, if and to what extent SRE approaches can support the development of an ISO 27001 compliant ISMS. Moreover, it addresses the question in what way SRE methods provide the required documentation for an ISMS and how existing SRE documentation can be re-used for an ISMS.

The rest of the paper is organized as follows. Section 2 presents background on the ISO 27001 standard, and Sect. 3 presents the CF of Fabian et al. [4]. We set up a relation between the ISO 27001 standard and the conceptual framework (CF) in Sect. 4. In addition, we obtain a relation of security requirements methods to the ISO 27001. Section 5 provides insights into the results of the relations and Sect. 6 presents related work. Section 7 concludes and gives directions for future research.

2 The ISO 27001 Standard

The ISO 27001 standard is structured according to the "Plan-Do-Check-Act" (PDCA) model, the so-called *ISO 27001 process* [3]. In the *Plan* phase an ISMS is established, in the *Do* phase the ISMS is implemented and operated, in the *Check* phase the ISMS is monitored and reviewed, and in the *Act* phase the ISMS is maintained and improved. In the *Plan* phase, the *scope and boundaries* of the ISMS, its *interested parties*, *environment*, *assets*, and all the *technology* involved are defined. In this phase, also the ISMS *policies*, *risk assessments*, *evaluations*, and *controls* are defined. Controls in the ISO 27001 are measures to *modify risk*.

The ISO 27005 [2] refines this process for risk management and extends it with a pre-phase for information gathering.

3 A Conceptual Framework for Security Requirements Engineering

Notions and terminology differ in different SRE methods In order to be able to compare different SRE methods, Fabian et al. [4] developed a CF that explains and categorizes building blocks of SRE methods. In their survey the authors also use the CF to compare different SRE methods. Karpati et al. [5, p. 714] conclude in their survey that the only existing "uniform conceptual framework for translations"of security terms and notions for SRE methods is the work of Fabian et al. [4]. Therefore, we use this CF and base our relations between the ISO 27001 and SRE methods on it. For simplicity's sake, we work on a subset of the CF. The CF considers security as a system property using the terminology of Michael Jackson [6], which defines that a *system* consists of a *machine* and its *environment*. The machine is the thing to be built, e.g., software. The part of the real world into which the machine will be integrated is the environment. The description of the desired behavior of the environment after the machine's integration is the so-called *requirement*. The CF considers four main building blocks of SRE methods: *Stakeholder Views*, *System Requirements*, *Specification and Domain Knowledge*, and *Threat Analysis*. **Stakeholder Views** identify and describe the stakeholders and their functional and non-functional goals and resulting functional and non-functional requirements. Stakeholders express security concerns via security goals. These goals are described towards an asset of the stakeholder, and they are refined into security requirements.

System Requirements result from a reconciliation of all functional, security and other non-funtional requirements, while the stakeholder view perspective focuses on the requirements of one stakeholder in isolation. Hence, the system requirements analysis includes the elimination of conflicts between requirements and their prioritization. The result is a coherent set of system requirements. Requirements are properties the system has after the machine is built. The **Specification and Domain Knowledge** building block consists of *specifications*, *assumptions* and *facts*. The specification is the description of the interaction behavior of the machine with its environment. It is the basis for the construction of the machine. Assumptions and facts make up the *domain knowledge*. The domain knowledge describes the environment in which the machine will be integrated. In practical terms this means the security requirements have to be reviewed in context of the environment. The **Threat Analysis** focuses on *security properties* required by stakeholders. A violation of a security property is a potential loss for a stakeholder. This loss constitutes a risk for the stakeholder, which is reduced by countermeasures. A vulnerability may lead to a violation of a security property, and it is mitigated by a countermeasure. Attacks actually exploit vulnerabilities, while threats only potentially exploit vulnerabilities. Attacks realize threats.

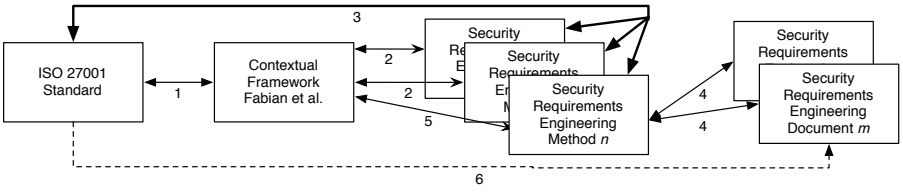


Fig. 1. Relating security requirements engineering methods with the ISO 27001

4 Relating the ISO 27001 Standard and Security Requirements Engineering Methods

Our work starts with a *top-down* approach. We systematically analyze the ISO 27001 standard in order to determine *where* and *how* SRE methods can support the development and documentation of an ISMS according to ISO 27001. We depict the analysis in Fig. 1. First, we create a relation between the ISO 27001 standard and the CF of Fabian et al. [4]. Second, we use the relation of terminologies and notions from the CF to numerous SRE methods already provided by Fabian et al. Third, combining the relations of steps 1 and 2 we can relate the ISO 27001 with different SRE methods.

The second part of our work is a *bottom-up* approach. We support re-using documents created by an SRE method, so-called *SRE documents* (step 4). We can re-use the relation between the used SRE method and the CF to figure out *what* ISO 27001 section the SRE documents support. If this relation does not yet exist, we have to create it (step 5). It is sufficient to create a relation between the CF and the SRE method, because of the existing relation between the CF and ISO 27001. Thus, transitive relations from the ISO 27001 to existing SRE documents are possible (step 6).

Note that the ISO 27001 sparsely describes the structure and content of an ISO 27001 compliant documentation. Thus, a relation between a specific artifact generated by a SRE method and a certain part of the documentation required by the ISO 27001 cannot firmly be established. It is up to the auditors to decide if an artifact fully fulfills an ISO 27001 documentation need.

Table 1 relates relevant terms for security from the CF by Fabian et al. [4] to the ISO 27001 standard. The matching benefits from the fact that both documents rely on ISO 13335 [7] definitions for several terms.

ISO 27001 Section 4 describes the ISMS. Hence, we focus on this section in particular. Table 2 lists relations between subsections of ISO 27001 Section 4 and the CF's building blocks. We present all subsections of ISO 27001 Section 4.2, because these describe the establishment of the ISMS. In addition, we show risk management as a separate column, even though it is part of the CF's building

Table 1. Correspondence between ISO 27001 terms and terms of the CF [4]

CF Fabian et al.	**ISO 27001**
System	The *organisation* is the "scope"of the standard [3, p. 1].
Machine	The *Information Security Management System (ISMS)* is the machine to be built [3, p. v].
Environment	The *scope and boundaries* of the "organization"[3, p. 4, Sec 4.2.1 a] relevant for the ISMS.
Security Goal	The standard uses *security objectives* [3, p. 4, Sec 4.2.1 b] instead of security goals.
Security Requirement	*Security requirement* is also used in ISO 27001 as a description of the "organization"after the "ISMS"is introduced [3, p. v,vi].
Specification	The ISMS's policy, controls, processes and procedures [3, p. vi] are the specification of the machine.
Stakeholder	The *Interested Parties* [3, p. vi] have security "expectations"that are input for the ISMS implementation as well as "security requirements".
Domain Knowledge	The *characteristics of the business, the organization, its location, assets and technology* [3, p. 4].
Availability	The definition in ISO/IEC 13335 [7] is also used [3, p. 2].
Confidentiality	The definition in ISO/IEC 13335 [7] is also used [3, p. 2].
Integrity	The definition in ISO/IEC 13335 [7] is also used [3, p. 2].
Asset	The definition in ISO/IEC 13335 [7] is also used [3, p. 2].
Threat	The definitions match. Threats are defined towards assets and threats exploit vulnerabilities [3, p. 4].
Vulnerability	The definitions match [3, p. 4].
Risk	The CF defines risk as "the potential loss of a stakeholder"[4, p. 13], while in ISO 27001 risk is not defined explicitly. However, the risk identification evolves around identifying asset, threat, vulnerability and the impact a loss of availability, confidentiality and availability has on an asset [3, p. 4]. Hence, we can conclude that the meaning is similar.

block *threat analysis*. The reason is that some subsections of ISO 27001 Section 4 and SRE methods specifically focus on risk management. Moreover, the importance of risk in the ISO 27000 series of standards resulted in the standard ISO 27005 for information security risk management that specifies the risk management of the ISO 27001 [2]. A "+"in Tab. 2 marks a part of the section that can be supported by a building block of the CF. However, the free cells of the table do not imply that a method could not support that section of the ISO 27001. A *grey* row indicates that there are no explicit matches between the ISO 27001 section and the CF.

5 Insights

We presented a relation between SRE methods and the ISO 27001 standard. The relations were obtained via the CF of Fabian et al. [4]. This CF presents four distinct building blocks of SRE methods. Table 2 relates the ISO 27001 standard to these building blocks. The *Stakeholder Views* building block has multiple relations to ISO 27001 sections. The reason is that the counterparts in the standard focus on the view of the organization including its stakeholders. The *Stakeholder Views* are part of numerous goal-oriented approaches, e.g., Secure Tropos [8] and KAOS [9]. This is no surprise, because these methods often derive their goals from the views of stakeholders.

Also the *Threat Analysis* building block has multiple counterparts in the ISO 27001. The reason for these is the strong emphasis of the standard on risk, which is part of that building block. Thus, risk management-oriented approaches, such as CORAS [10], play a crucial role in an ISO 27001 assembly. The problem-oriented approaches, e.g. SEPP [11], are useful for the structured collection of knowledge about the environment that must be considered.

Table 3 presents the mandatory documents for an ISO 27001 documentation according to [3, p.13]. In addition, Tab. 3 shows the kinds of SRE methods that support the assembly of these documents. The table is based upon our analysis in Sect. 4.

Table 2. Relating ISO 27001 Section 4 to CF building block

Section	Description	SV	SR	SDK	TA	RM
Sect. 4.1	General requirements	+	+	+	+	+
Sect. 4.2	Establish and manage the ISMS	+	+	+	+	+
Sect. 4.2.1	Establish the ISMS	+	+	+	+	+
Sect. 4.2.1 a	Define scope and boundaries	+		+		
Sect. 4.2.1 b	Define ISMS policy	+	+		+	+
Sect. 4.2.1 c	Define risk assessment					+
Sect. 4.2.1 d	Identify the risk	+			+	+
Sect. 4.2.1 e	Analyse and evaluate risk			+	+	+
Sect. 4.2.1 f	Identify risk treatment				+	+
Sect. 4.2.1 g	Select controls				+	+
Sect. 4.2.1 h,i	Obtain management approval					
Sect. 4.2.1 j	Prepare a statement of applicability				+	+
Sect. 4.2.2	Implement and operate the ISMS				+	+
Sect. 4.2.3	Monitor and review the ISMS	+	+	+	+	+
Sect. 4.2.4	Maintain and improve the ISMS					
Sect. 4.3	Documentation requirements	+	+	+	+	+

SV(Stakeholder Views),**SR**(System Requirements), **SDK**(Specification and Domain Knowledge),**TA**(Threat Analysis), **RM**(Risk Management)

Table 3. Support of SRE Methods for ISO 27001 documentation

Documentation Requirements ISO 27001	Support from SRE Methods
ISMS policies and objectives	Goal-/Problem-/Risk-oriented methods
Scope and boundaries of the ISMS	Goal-/Problem-/Risk-oriented methods
Procedures and controls	Risk-oriented methods
The risk assessment methodology	Risk-oriented methods
Risk assessment report	Risk-oriented methods
Risk treatment plan	Risk-oriented methods
Information security procedures	Goal-/Problem-oriented methods
Control and protection of records	No support from SRE methods
Statement of Applicability	Goal-/Problem-/Risk-oriented methods

6 Related Work

Mondetino et al. investigate possible automation of controls that are listed in the ISO 27001 and ISO 27002 [12]. Beckers et al. [13] propose a common pattern for the cloud computing domain to support context establishment and asset identification of the ISO 27000 series. Both works can complement our own.

7 Conclusion

We have established a relation between the ISO 27001 standard and SRE methods. Thereby we build on the CF of Fabian et al. [4], which already established relations between the CF's terms and notions of several SRE methods. We contribute further relations from the ISO 27001 standard to the CF. The two sets of relations can be combined to identify suitable SRE methods for establishing an ISMS compliant with ISO 27001.

Our approach offers the following main benefits:

- Re-using SRE methods to support the development and documentation of security standards (here: ISO 27001) compliant systems
- Systematic identification of relevant SRE methods for an ISO 27001 section
- Improving the outcome of ISO 27001 implementation by supporting establishment and documentation of an ISMS
- Re-using the structured techniques of SRE methods for analyzing and elicitating security requirements to support the refinement of sparsely described sections of the ISO 27001 standard

In the future we will look into extensions of SRE methods in order to be able to support the management and auditing demands of the ISO 27001 standard.

Acknowledgements. We thank Denis Hatebur for his extensive and valuable feedback on our work.

References

1. ISO/IEC: Information technology - Security techniques - Information security management systems - Overview and Vocabulary. ISO/IEC 27000, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) (2009)
2. ISO/IEC: Information technology - security techniques - information security risk management. ISO/IEC 27005, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) (2008)
3. ISO/IEC: Information technology - Security techniques - Information security management systems - Requirements. ISO/IEC 27001, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) (2005)
4. Fabian, B., Gürses, S., Heisel, M., Santen, T., Schmidt, H.: A comparison of security requirements engineering methods. Requirements Engineering – Special Issue on Security Requirements Engineering 15(1), 7–40 (2010)
5. Karpati, P., Sindre, G., Opdahl, A.L.: Characterising and analysing security requirements modelling initiatives. In: Proceedings of the International Conference on Availability, Reliability and Security (ARES), pp. 710–715. IEEE Computer Society (2011)
6. Jackson, M.: Problem Frames. Analyzing and structuring software development problems. Addison-Wesley (2001)
7. ISO/IEC: Information technology - Security techniques - Management of information and communications technology security - Part 1: Concepts and models for information and communications technology security. ISO/IEC 13335-1, International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) (2004)
8. Mouratidis, H., Giorgini, P.: Secure tropos: a security-oriented extension of the tropos methodology. International Journal of Software Engineering and Knowledge Engineering 17(2), 285–309 (2007)
9. van Lamsweerde, A.: Engineering requirements for system reliability and security. In: Broy, M., Grunbauer, J., Hoare, C.A.R. (eds.) Software System Reliability and Security. NATO Security through Science Series - D: Information and Communicarion Security, vol. 9, pp. 196–238 (2007)
10. Lund, M.S., Solhaug, B., Stølen, K.: Model-Driven Risk Analysis: The CORAS Approach, 1st edn. Springer, Heidelberg (2010)
11. Schmidt, H., Hatebur, D., Heisel, M.: A pattern- and component-based method to develop secure software. In: Mouratidis, H. (ed.) Software Engineering for Secure Systems: Academic and Industrial Perspectives, pp. 32–74. IGI Global (2011)
12. Montesino, R., Fenz, S.: Information security automation: how far can we go? In: Proceedings of the International Conference on Availability, Reliability and Security (ARES), pp. 280–285. IEEE Computer Society (2011)
13. Beckers, K., Küster, J.C., Faßbender, S., Schmidt, H.: Pattern-based support for context establishment and asset identification of the ISO 27000 in the field of cloud computing. In: Proceedings of the International Conference on Availability, Reliability and Security (ARES), pp. 327–333. IEEE Computer Society (2011)

Typed Assembler for a RISC Crypto-Processor

Peter T. Breuer[1,⋆] and Jonathan P. Bowen[2,⋆⋆]

[1] Department of Computer Science, University of Birmingham, UK
ptb@cs.bham.ac.uk
[2] Faculty of Business, London South Bank University, UK / Museophile Limited, UK
jonathan.bowen@lsbu.ac.uk
http://www.jpbowen.com

Abstract. Our general purpose crypto-processor runs RISC machine code in an encrypted environment, reading encrypted inputs and generating encrypted outputs while maintaining data encrypted in memory. Its intended use is secure remote processing. However, program addresses are processed unencrypted, resulting in a mix of encrypted and unencrypted data in memory and registers at any time. An aspect of compiling for it is typing the assembler code to make sure that those instructions that expect encrypted data always get encrypted data at execution time, and those that expect unencrypted data get unencrypted data. A type inference system is specified here and transformed into an executable typing algorithm, such that a type-checked asembler program is guaranteed type-safe.

1 Introduction

The term 'crypto-processor' has been used to label several hardware-based solutions aimed at helping system security [2,5,7]. Our crypto-processor [1] is a general purpose unit that performs computations on mixed unencrypted and encrypted data held at encrypted addresses in memory. Its instruction set is standard RISC [6] but interpreted on encrypted data. In other words, when the processor computes 43+43=1234789 via an '**addiu**' machine instruction, it may well be computing an encrypted version of 1+1=2 but the latter 'translation' should be unknown to all but the remote owner. The processor characteristics are summarised in Box 1.

The intention is to be able to hide data and process from prying eyes in a remote computing environment – a cloud, for example. The hardware design is intended to make that feasible and secure even in simulation. The detail of the design is such that: (i) arithmetic and logical machine code instructions act on encrypted data and encrypted data addresses and produce

Box 1. A crypto-processor ...

... for the purposes of this article is a RISC CPU that manipulates mixed encrypted and unencrypted data in general purpose registers and memory. It runs on:

- encrypted data values and addresses, giving encrypted results where appropriate;
- unencrypted program addresses; while
- data and function codes embedded in the machine code are encrypted, register indices unencrypted.

⋆ This paper was initiated through an academic visit by the first author to London South Bank University as a Visiting Research Fellow during 2011–12.
⋆⋆ Jonathan Bowen is grateful for financial support from Museophile Limited.

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 22–29, 2012.

encrypted data; (ii) control instructions (branches, jumps, etc) act on unencrypted program addresses; (iii) data embedded in machine code instructions and instruction function codes are encrypted; (iv) data or data addresses in memory and registers are encrypted; (v) program addresses in memory and registers are unencrypted; (vi) memory is divided into heap, which contains encrypted data, and stack, which contains mixed encrypted and unencrypted data.

As a consequence of the program addresses remaining unencrypted, programs are generally arranged in contiguous areas of memory. While that is certainly an advantage for cacheing, there is a different physical reason behind it: the circuit that updates the program counter is physically distinct from the circuitry that does the general arithmetic in the CPU. There is also a cryptographic reason: since the usual change in the program counter from cycle to cycle is a straightforward increment, plenty of information on the encryption could be gathered were the program counter to be observed (nevertheless, there is no fundamental design impediment to encrypting program addresses too).

That is a brief overview of how our crypto-processor works, but where is the necessity for type-checking assembler code? The answer is that at least one standard RISC machine instruction, the 'jump register' (**jr**) instruction, expects to read an unencrypted program address value from a register. And it is not the only instruction to expect unencrypted data – or to generate it. Thus, at any given moment, memory and registers in our crypto-processor contain a *mix* of encrypted and unencrypted data. That raises the question of whether a program for the crypto-processor is properly *type-safe*:

Definition 1. *A program is* type-safe *for a crypto-processor if those machine instructions in the program that work on encrypted data always get encrypted data on which to work during execution of the program, while those instructions that work on unencrypted data always get unencrypted data on which they can work.*

This paper sets out a type-checking algorithm for assembler programs written for the crypto-processor, such that when a program type-checks, then it is type-safe.

The organisation of this paper is as follows: Section 2 introduces type-checking. Section 3 details the RISC+CRYPT assembler language (in an unencrypted representation) for our crypto-processor, and gives the inference rules of a type-system. Section 4 turns the inference system into an algorithm which deduces types. Section 5 elaborates the type-system (and associated type-checking algorithm) so that a RISC+CRYPT program which type-checks successfully is guaranteed type-safe.

2 Type-Checking

Successfully type-checking a program guarantees that:

(a) the distribution of encrypted and unencrypted data in the registers at every pass through the encrypted machine code always satisfies the same pattern at the same point in the code;
(b) the distribution of encrypted and unencrypted data in the registers is compatible with the instruction operating on them at every point.

The first claim says that the pattern of encrypted and unencrypted data in registers is stable as loops are repeated, subroutines are called, etc. The simple system of encrypted '**c**' and unencrypted '**u**' data types used by the analysis is shown in Box 2.

Box 2. A simple system of types ...

... describes register contents each processor cycle:

```
c   // encrypted data
u   // unencrypted data
x   // type variables match all data
```

For example, a pattern of types in registers 0 to 31 before and after the **move t2 t3** (i.e., '$\mathbf{t2} \leftarrow \mathbf{t3}$' for registers **t2**, **t3**) instruction, is shown in Box 3. After the instruction, the type of the data in register **t2** may be either encrypted or unencrypted, but it is constrained to be the same as the type of the data in register **t3**. The type signature is expressed as follows:

$$\mathbf{move\ t2\ t3} :: [\mathbf{t3} : \boldsymbol{x}] \rightarrow [\mathbf{t2} : \boldsymbol{x},\ \mathbf{t3} : \boldsymbol{x}]$$

Variable $\boldsymbol{x}$ matches any data type and every register not explicitly mentioned remains unchanged in type. The notation will be used throughout this article.

What of the second guarantee afforded by the typing algorithm? The claim is that the before-after patterns of register occupation around each instruction conform to the semantics of the instruction. In the case of the **move t2 t3** instruction, that means that whatever kind of data was in register 11 (**t3**) at the beginning, is also the kind of data found in register 10 (**t2**) afterwards, and nothing else has changed, just as in Box 3.

Box 3. A register type pattern ...

... around the **move t2 t3** instruction (i.e., $\mathbf{t2} \leftarrow \mathbf{t3}$):

reg.	0	1	2	3	...	10	11	...
before	$\boldsymbol{x}_0$	**c**	$\boldsymbol{x}_2$	**u**	...	**c**	$\boldsymbol{x}_{11}$	...
after	$\boldsymbol{x}_0$	**c**	$\boldsymbol{x}_2$	**u**	...	$\boldsymbol{x}_{11}$	$\boldsymbol{x}_{11}$	...

Registers **t2**, **t3** are registers 10, 11 respectively.

The crypto-processor assembler contains 'CRYPT' pseudo-instructions that the compiler translates to plain RISC machine code, but which are there to allow assembler typing to proceed with accuracy. Box 4 lists these succinctly. They deal with data transfers to and from the stack area, which are implemented using RISC add, load and store machine instructions. However, the machine code may access anywhere in memory, both stack and heap, and those two areas are treated very differently by our analysis: heap may only contain encrypted data in our design, while stack may contain both encrypted and unencrypted data (a polyvalent stack is necessary to the design because some RISC machine instructions – jumps and branches – require unencrypted program addresses in registers, which data needs to be saved on the stack during subroutine calls). The extra CRYPT pseudo-instructions allow the type analysis of the assembler to adequately distinguish the two areas of memory.

Box 4. The assembly language ...

... for the crypto-processor includes 'CRYPT' pseudo-instructions in addition to RISC assembler:

- **push** n, **pop** n – in-/decrease stack by n words;
- **pushu** r, **pushc** r – append to stack one new word copied from plaintext/encrypted register r;
- **popu** r, **popc** r – displace last word of plaintext/encrypted stack content to register r;
- **putu** $r\ n$, **putc** $r\ n$ – copy plaintext/encrypted contents of register r to n'th stack word;
- **getu** $r\ n$, **getc** $r\ n$ – copy the plaintext/encrypted n'th stack word to register r.

The RISC part of the assembler instruction set is listed in Box 5. It is entirely standard and translates directly to machine code instructions.

Box 5. RISC assembly language.

lui $r\ n$	// Set reg. content.
sb $r_1\ n(r_2)$	// Store byte to mem.
lb $r_1\ n(r_2)$	// Load byte from mem.
sw $r_1\ n(r_2)$	// Store word to mem.
lw $r_1\ n(r_2)$	// Load word from mem.
jr r	// Jump to addr. in reg.
j a	// Jump to addr.
jal a	// Jump and link.
bnez $r\ a$	// Branch if reg. $\neq 0$.
nop	// No-op, do nothing.
move $r_1\ r_2$	// Copy from reg. to reg.
ori $r_1\ r_2\ n$	// Arithmetic bitwise or.
addiu $r_1\ r_2\ n$	// Arithmetic add op.
...	// ...

Embedded data n is encrypted, embedded program addresses a and register indices r are unencrypted.

3 Assembler Typing

This section will set out a type system based on the **c**, **u** types for the crypto-processor assembler code.

Most machine instructions do not perturb the ordinary linear flow of control through a program. These *linear* instructions comprise all instructions apart from jumps and branches. When a linear instruction i_a at address a executes, control inevitably passes afterward to the *next* program instruction after it in positional sequence in memory. In a RISC 32-bit MIPS [4] machine, the next instruction is at address $a+4$ (4 bytes further on). To avoid prejudice we set:

Definition 2. *a' is the address of the* successor *instruction sited immediately beyond the instruction i_a at address a in the program code.*

We will use a' throughout this paper in place of any particular increment $a+\text{length}(i_a)$.

Type signatures for linear assembler instructions can be expressed in the notation of Sect. 2, as shown in Box 6. For example, the **lui** $r\ n$ instruction sets the the content of a register r to the encrypted value $2^{16}n$ (n is supplied as an encrypted value embedded in the instruction itself), and thus its type signature is given as $[\,] \to [r : \mathbf{c}]$ in Box 6.

Placing two instructions of signatures $t_1 \to t_2$ and $t_3 \to t_4$ in sequence is only possible if the types t_2 and t_3 can be reconciled. If type t_2 says the type of register 1 is **c** and type t_3 says it is **u**, then it is not possible. But reconciliation, if possible, yields:

Box 6. Linear RISC+CRYPT signatures.

lui $r\ n$	$:: [\,] \to [r : \mathbf{c}]$
sb $r_1\ n(r_2)$	$:: [r_1, r_2 : \mathbf{c}] \to [r_1, r_2 : \mathbf{c}]$
lb $r_1\ n(r_2)$	$:: [r_2 : \mathbf{c}] \to [r_1, r_2 : \mathbf{c}]$
sw $r_1\ n(r_2)$	$:: [r_1, r_2 : \mathbf{c}] \to [r_1, r_2 : \mathbf{c}]$
lw $r_1\ n(r_2)$	$:: [r_2 : \mathbf{c}] \to [r_1, r_2 : \mathbf{c}]$
nop	$:: [\,] \to [\,]$
move $r_1\ r_2$	$:: [r_2 : \boldsymbol{x}] \to [r_1, r_2 : \boldsymbol{x}]$
ori $r_1\ r_2\ n$	$:: [r_2 : \mathbf{c}] \to [r_1, r_2 : \mathbf{c}]$
addi $r_1\ r_2\ n$	$:: [r_2 : \mathbf{c}] \to [r_1, r_2 : \mathbf{c}]$
...	
putc $r\ n$	$:: [r, \mathbf{sp} : \mathbf{c}] \to [r, \mathbf{sp} : \mathbf{c}]$
putu $r\ n$	$:: [r : \mathbf{u}, \mathbf{sp} : \mathbf{c}] \to [r : \mathbf{u}, \mathbf{sp} : \mathbf{c}]$
getc $r\ n$	$:: [\mathbf{sp} : \mathbf{c}] \to [r, \mathbf{sp} : \mathbf{c}]$
getu $r\ n$	$:: [\mathbf{sp} : \mathbf{c}] \to [r : \mathbf{u}, \mathbf{sp} : \mathbf{c}]$
push n	$:: [\mathbf{sp} : \mathbf{c}] \to [\mathbf{sp} : \mathbf{c}]$
pop n	$:: [\mathbf{sp} : \mathbf{c}] \to [\mathbf{sp} : \mathbf{c}]$

Definition 3. Sequential composition:

$$t_1 \to t_2\ ;\ t_3 \to t_4 \triangleq \text{unify}(t_2, t_3)(t_1 \to t_4)$$

where 'unify' delivers the variable bindings required to reconcile pattern t_2 with t_3, and applies them to the type $t_1 \to t_4$, giving the type of the sequential composition.

Other program type calculations are more complicated. In general two type patterns t_1, t_2 (written '$t_1 \to t_2$') have to be developed for every instruction address a in a program: t_1 is the most general register type pattern that the instruction may validly encounter when it starts; t_2 is that which subsequently obtains at program termination (after the instruction at some address b). For a subroutine, exit is just after the **jr** that returns control to caller.

Box 7. Simple RISC+CRYPT typing rules ...

... in terms of the instruction $[i_a]$ at address a and the type at the next instruction address a' after a by position:

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [\,] \to [r{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{lui}\; r\; n]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r_1, r_2{:}\mathbf{c}] \to [r_1, r_2{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{sb}\; r_1\; n(r_2)]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r_2{:}\mathbf{c}] \to [r_1, r_2{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{lb}\; r_1\; n(r_2)]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r_1, r_2{:}\mathbf{c}] \to [r_1, r_2{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{sw}\; r_1\; n(r_2)]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r_2{:}\mathbf{c}] \to [r_1, r_2{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{lw}\; r_1\; n(r_2)]$$

$$\frac{}{T \vdash a :: [r{:}\mathbf{u}] \to [r{:}\mathbf{u}]}[\mathbf{jr}\; r]$$

$$\frac{T \vdash b :: t_1 \to t_2}{T \vdash a :: t_1 \to t_2}[\mathbf{j}\; b]$$

$$\frac{T \vdash b :: t_1 \to t_2 \qquad T \vdash a' :: t_3 \to t_4}{T \vdash a :: [\,] \to [\mathbf{ra}{:}\mathbf{u}] \;;\; t_1 \to t_2 \;;\; t_3 \to t_4}[\mathbf{jal}\; b]$$

$$\frac{T \vdash b[r : \mathbf{c}] :: t_1 \to t_2 \quad T \vdash a'[r : \mathbf{c}] :: t_1 \to t_2}{T \vdash a :: [r : \mathbf{c}] \to [r : \mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{bnez}\; r\; b]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: t_1 \to t_2}[\mathbf{nop}]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r_2{:}\boldsymbol{x}] \to [r_1, r_2{:}\boldsymbol{x}] \;;\; t_1 \to t_2}[\mathbf{move}\; r_1\; r_2]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r_2{:}\mathbf{c}] \to [r_1, r_2{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{ori}\; r_1\; r_2\; n]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r_2{:}\mathbf{c}] \to [r_1, r_2{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{addiu}\; r_1\; r_2\; n]$$

$$\ldots$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r, \mathbf{sp}{:}\mathbf{c}] \to [r, \mathbf{sp}{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{putc}\; r\; n]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [r{:}\mathbf{u}, \mathbf{sp}{:}\mathbf{c}] \to [r{:}\mathbf{u}, \mathbf{sp}{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{putu}\; r\; n]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [\mathbf{sp}{:}\mathbf{c}] \to [r, \mathbf{sp}{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{getc}\; r\; n]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [\mathbf{sp}{:}\mathbf{c}] \to [r{:}\mathbf{u}, \mathbf{sp}{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{getu}\; r\; n]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [\mathbf{sp}{:}\mathbf{c}] \to [\mathbf{sp}{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{push}\; n]$$

$$\frac{T \vdash a' :: t_1 \to t_2}{T \vdash a :: [\mathbf{sp}{:}\mathbf{c}] \to [\mathbf{sp}{:}\mathbf{c}] \;;\; t_1 \to t_2}[\mathbf{pop}\; n]$$

Definition 4. *A collection of types $t_1 \to t_2$ indexed by entry addresses a is called a* theory. *We write*

$$T \vdash a :: t_1 \to t_2$$

for 'theory T lists the type $t_1 \to t_2$ against address a'.

In an actual program run, the register type pattern encountered by any particular instruction i_a at address a may be strictly less general than the type t_1 recorded in theory T. It will be the result $\sigma(t_1)$ of a substitution σ for type variables in t_1. The register type pattern at the end of the run will then match $\sigma(t_2)$.

The deduction rules of a type theory T for our cryptoprocessor are given in Box 7. Each rule is associated with a single program address a, and the instruction i_a located at that address. For the branch rule, both possible continuations after the branch test must give rise to the same register type pattern at program exit. The branch test requires an encrypted datum in register r and the following notation helps express the rule succinctly:

Definition 5. $T \vdash b[r : t] :: t_1 \to t_2 \;\triangleq\; T \vdash b :: t_3 \to t_4$
where $t_3 \to t_4$ is such that $[r : t] \to [r : t] \;;\; t_1 \to t_2 = [r : t] \to [r : t] \;;\; t_3 \to t_4$.

(two branch types become equal after substituting t for the type of r on entry to both).

4 The Basic Algorithm

Box 8. Altered rules ...

... for calculating the fixpoint type theory $T_n = T_{n-1}$ of a program, in terms of the instruction $[i_a]$ at address a and the following instruction address a'. All other rules from Box 7 have T_n substituted for T throughout.

$$\frac{}{T_0 \vdash a :: [0{:}\boldsymbol{x}_{a0}, 1{:}\boldsymbol{x}_{a1}, \dots] \to [0{:}\boldsymbol{y}_{a0}, 1{:}\boldsymbol{y}_{a1}, \dots]}\,[*]$$

$$\frac{T_{n-1} \vdash b :: t_1 \to t_2}{T_n \vdash a :: t_1 \to t_2}\,[\mathbf{j}\ b,\ b \le a]$$

$$\frac{T_{n-1} \vdash b[r : \mathbf{c}] :: t_1 \to t_2 \quad T_n \vdash a'[r{:}\mathbf{c}] :: t_1 \to t_2}{T_n \vdash a :: [r{:}\mathbf{c}] \to [r{:}\mathbf{c}]\ ;\ t_1 \to t_2}\,[\mathbf{bnez}\ r\ b,\ b \le a]$$

$$\frac{T_{n-1} \vdash b :: t_1 \to t_2 \qquad T_n \vdash a' :: t_3 \to t_4}{T_n \vdash a :: [\,] \to [\mathbf{ra} : \mathbf{u}]\ ;\ t_1 \to t_2\ ;\ t_3 \to t_4}\,[\mathbf{jal}\ b]$$

To that end, define T_n as the n'th iteration of a series leading to the final theory T that is a fixpoint of the iteration. Initially, the theory assigns to address a the 'any' pattern, in which all registers are bound to different type variables and inputs are not related to outputs (rule [*] in Box 8). A theory T_{n-1} in the sequence is used to help construct the next theory T_n, $n > 0$. by substitution for free variables in the types at each address a, as shown in Box 8. In all other cases, the rules are just as given in Box 7 but with T replaced with T_n throughout.

When no improvement is obtained from T_{n-1} to T_n at any address a, then the fixpoint theory T has been reached. Substituting T for T_n and T_{n-1} in Box 8 shows that the fixpoint T satisfies the rules of Box 7. There are only a finite number of proper substitutions possible as steps of the algorithm, so the iteration does terminate. In practice the number of iterations required is approximately the number of backward jumps and branches plus subroutine calls in the code.

5 Taking Account of the Stack

Although we have supplied a type system, it is not the case that, as is, the system constrains a type-checked program to be type-safe. The problem is evident in the fragment:

```
putu t1 0; getc t1 0
```

which writes an unencrypted value to the stack and then recovers the same datum as an encrypted value. The type of register **t1** changes from **u** to **c** yet the content of the register does not change. The **getc** instruction encounters an unencrypted value on the stack where it expects an encrypted value, yet the fragment type-checks. 'Typeable' via the system given so far means that code is type-safe for the crypto-processor only under the hypothesis that the stack operations in the code are independently type-safe.

To remove that additional assumption, the types of the values in different *stack slots* have to be tracked. The size of the stack will from now on be denoted by an annotation on the right side of a list of register types $[r_1 : t_1, r_2 : t_2, \dots]^d_k$. The subscript k indicates the list is $k \ge 32$ long, and the last $k - 32$ list entries represent the stack slots, while the first 32 represent the registers proper. The size $d \le k - 32$ of the current stack frame

is indicated by the superscript on the list. Writing to the n'th from the bottom word on the stack in the current frame with **putu** $r\ n$ accesses the n'th of the last d list entries, which is entry number $k-d+n$ in the list.

The type rules of Box 7 are altered as shown in Box 9. In particular, the evidently too-loose typing given for **putu**, **getu** in Box 7 is mended here so that **getu** requires to act on a stack slot of type **u**, and **putu** creates a stack slot of type **u**.

Box 9. Extended type rules ...

... which track types through the stack. The other rules of Box 7 uniformly have $\restriction_k^d$ added to the type assignment lists, indicating that the list is of length k and the last d entries represent the current stack frame (the first 32 of k are the registers), and is unchanged through the rule. The rules are for instruction $[i_a]$ at address a in terms of the type at the following program address a'.

$$\frac{T\vdash a'::t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}{T\vdash a::[r{:}\mathbf{u}]_k^d\to[r{:}\mathbf{u},k{-}d{+}n{:}\mathbf{u}]_k^d\ ;\ t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}[\mathbf{putu}\,r\,n]$$

$$\frac{T\vdash a'::t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}{T\vdash a::[r{:}\mathbf{c}]_k^d\to[r{:}\mathbf{c},k{-}d{+}n{:}\mathbf{c}]_k^d\ ;\ t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}[\mathbf{putc}\,r\,n]$$

$$\frac{T\vdash a'::t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}{T\vdash a::[k{-}d{+}n{:}\mathbf{u}]_k^d\to[r{:}\mathbf{u},k{-}d{+}n{:}\mathbf{u}]_k^d\ ;\ t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}[\mathbf{getu}\,r\,n]$$

$$\frac{T\vdash a'::t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}{T\vdash a::[k{-}d{+}n{:}\mathbf{c}]_k^d\to[r{:}\mathbf{c},k{-}d{+}n{:}\mathbf{c}]_k^d\ ;\ t_1\restriction_k^d\to t_2\restriction_{k'}^{d'}}[\mathbf{getc}\,r\,n]$$

$$\frac{T\vdash a'::t_1\restriction_{k+n}^{d+n}\to t_2\restriction_{k'}^{d'}}{T\vdash a::[\,]_k^d\to[k{:}x_k\ldots k{+}n{-}1{:}x_{k+n-1}]_{k+n}^{d+n}\ ;\ t_1\restriction_{k+n}^{d+n}\to t_2\restriction_{k'}^{d'}}[\mathbf{push}\,n]$$

$$\frac{T\vdash a'::t_1\restriction_{k-n}^{d-n}\to t_2\restriction_{k'}^{d'}}{T\vdash a::[k{-}n{:}x_{k-n}\ldots k{-}1{:}x_{k-1}]_k^d\to[\,]_{k-n}^{d-n}\ ;\ t_1\restriction_{k-n}^{d-n}\to t_2\restriction_{k'}^{d'}}[\mathbf{pop}\,n]$$

$$\frac{T\vdash b::t_1\restriction_{32,d}^{d}\to t_2\restriction_{32,d}^{d}\qquad T\vdash a'::t_3\restriction_k^d\to t_4\restriction_{k'}^{d'}}{T\vdash a::[\,]_k^d\to[\mathbf{ra}{:}\mathbf{u}]_k^d\ ;\ t_1\restriction_k^d\to t_2\restriction_k^d\ ;\ t_3\restriction_k^d\to t_4\restriction_{k'}^{d'}}[\mathbf{jal}\,b]$$

All other rules of Box 7 uniformly have $\restriction_k^d$ added to the type lists, indicating that the type list is of length k and the last d entries represent the current stack frame (the first 32 of k are the registers) and the list length/stack size is unchanged by the rule. The 'subroutine return' instruction **jr** in particular does *not* modify the stack – it is the **jal** 'subroutine call' rule that does the job. It drops consideration of all callee frames for the parent.

Proposition 1. *In the type system of Box 9 for the crypto-processor of Sect. 1,* type-checked implies type-safe *for RISC+CRYPT assembler programs.*

Proof. (Sketch) 'Notice' that the type rules and the algorithm that computes types from them together define an abstract interpretation [3] of the program: the values obtained in a program run match the type patterns computed, if the typing algorithm succeeds. Then the values obtained match the typing rule corresponding to each instruction in the program, since the algorithm works by refining each rule at each site where it is applicable and only one rule is applicable at each program address, that corresponding to the instruction located there. If a set of values obtained during execution does not match the instruction's input expectations, then – since it does match the input of the corresponding typing rule – the typing rule concerned permits inputs that do not match the instruction's expectations. But each typing rule can be seen by inspection not to allow inputs that are outside the corresponding instruction's expected range. That proves the result by contradiction. □

Where this argument falls down for the case of the basic type system of Sect. 3 and Box 7 is that its rules for **getc**, **getu** do permit inputs on the stack that do not match the expected range – an encrypted value is expected in the referenced stack slot for **getc**, but an unencrypted value is allowed by that type system, for example. The system of Box 9 mends that defect by tracking types through the stack.

6 Conclusion

We have introduced a RISC 'crypto-processor' that processes data kept in encrypted form in memory and registers, along with a type-checking algorithm for its assembly language. A type-checked program is type-safe: at run-time, encrypted data is always encountered by every instruction that expects encrypted data, and unencrypted data is always encountered by every instruction that expects unencrypted data.

7 Future Work

It turns out that it is possible to type-check RISC machine code directly by adapting the type system here from assembler to machine code. That enables a pre-existing machine code program to be type-checked, encrypted instruction by instruction, and run safely on our crypto-processor in a potentially hostile environment, the encrypted results of the computation being returned securely to the remote owner. The patent [1] contends that the processor design is secure even in simulation, making its (virtual) export feasible.

References

1. Breuer, P.T.: Encrypted data processing, patent pending, UK Patent Office GB1120531.7 (November 2011)
2. Buchty, R., Heintze, N., Oliva, D.: Cryptonite – A Programmable Crypto Processor Architecture for High-bandwidth Applications. In: Müller-Schloer, C., Ungerer, T., Bauer, B. (eds.) ARCS 2004. LNCS, vol. 2981, pp. 184–198. Springer, Heidelberg (2004)
3. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. 4th ACM Symposium on the Principles of Programming Languages, pp. 238–252. ACM (1977)
4. Hennessy, J.L.: VLSI processor architecture. IEEE Trans. on Computers 33(C), 1221–1246 (1984)
5. Oliva, D., Buchty, R., Heintze, N.: AES and the cryptonite crypto processor. In: Proc. CASES 2003: International Conference on Compilers, Architecture and Synthesis for Embedded Systems. ACM (2003)
6. Patterson, D.A.: Reduced instruction set computers. Communications of the ACM 28(1), 8–21 (1985)
7. Sun, M.C., Su, C.P., Huang, C.T., Wu, C.W.: Design of a scalable RSA and ECC crypto-processor. In: Proc. ASP-DAC 2003: Asia and South Pacific Design Automation Conference. ACM (2003)

Transversal Policy Conflict Detection*

Matteo Maria Casalino, Henrik Plate, and Slim Trabelsi

SAP Labs France
Mougins, France

Abstract. Declarative policies are a common means to manage the security of complex IT environments and they belong to different, heterogeneous classes (access control, filtering, data protection, etc.). Their enforcement requires the selection and configuration of appropriate enforcement mechanisms whose dependencies in a given environment may result in conflicts typically not foreseeable at policy design time. Such conflicts may cause security vulnerabilities and non compliance; their identification and correction is costly. Detecting transversal policy conflicts, i.e., conflicts happening across different policy classes, constitutes a challenging problem, and this work makes a step forward towards its formalization.

1 Introduction

Security management is a key task for organizations offering and consuming services through large IT infrastructures. Declarative security policies constitute the usual means to specify the intended behavior of the different security mechanisms that have to be put in place in such organizations. The process to design these policies starts from an analysis of the security requirements stemming from multiple sources like customers, business partners, internal regulations, etc. This analysis will point out the different security threats related to a system, hereby identifying a number of security requirements like: resource access control, information protection (confidentiality and integrity), privacy safeguarding, etc. Countermeasures are then selected and put in place in order to effectively meet the defined security objectives. The intended behavior of these countermeasures is described by different security policies. Policies are enforced by security mechanisms often spread over several different IT architectural layers, for instance network filtering, transport or message level data encryption, application specific access control, and must cooperate in a coherent fashion to avoid misbehaviors which may lead to limited functionality or security breaches.

We identify several types or "classes" of security properties: authorization, access control, usage control/obligation, authentication, data protection, filtering, etc. For each type several security policies may exist to describe the security rules. When these policies are deployed and enforced at the same time within the

* This work was supported by the European Union's 7th Framework Program through project PoSecCo, grant agreement no. 257129.

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 30–37, 2012.

same landscape there is a high probability to observe some misbehaviors during the runtime. This is due to the existence of incompatibilities or conflicts among security rules arising within the enforcement phase that are not predictable at design time, when the configuration of the landscape is only partially known. In addition, policies are likely to be authored by separate entities because (i) different organizations or domains have to interact and connect with each other, or (ii) they belong to different classes and thus concern distinct domains. Although the detection of policy conflicts is largely addressed in the literature especially when conflicts happen between the same class of policies, to our knowledge there is no solution focused on addressing the conflicts that can happen between different classes, for example when a data protection policy requires encrypting a channel with HTTPS/SSL and a firewall needs to perform payload inspection.

In this preliminary work we propose an approach to detect conflicts and incoherences between policies of different classes. Section 2 presents related work on policy conflict detection. Section 3 introduces a motivating scenario together with some examples of policies and possible transversal conflicts. In Section 4 we first define a Domain Description Model (DDM) describing the interaction between different classes of policies and capturing the interdependencies between them. Second we provide a formal representation of policies belonging to different classes. This representation called Class-Specific Policy Model (CSPM) tends to extract the inter-dependent parameters within the different classes of policies. Finally, we define a Conflict Specification Model (CSM) where we declare the different transversal conflicts that can be detected. We hereby formalize the transversal conflict detection problem as the satisfiability check of the CSM. A preliminary design for our framework is described throughout the paper, mapping all its parts to an illustrative first order language. Section 5 outlines conclusions and future work.

2 Related Work

Related work on policy conflict detection mainly focuses on conflicts happening within a single class of policies. We instead address conflicts among different classes. In [1] authors propose a classification of firewall rules anomalies and algorithms for detecting them. However the impact that firewall rules may have on other classes of security policies is not considered, and the scope is limited to network-level filtering, without considering application-layer packet inspection.

Access control policy analysis have been also extensively studied in literature. The Ponder policy language can be formalized in Event Calculus allowing for detecting conflicts by the means of abductive reasoning [3]. Formalizations for non-procedural fragments of standardization and industry oriented policy languages like XACML [8] have been as well proposed. These approaches allow for various kinds of policy analysis, including conflict detection. However they are mainly focused on access control, which constitutes only one of our possible policy classes. In [4] authors approach the problem of composing policies expressed in different languages providing a common generic language to express them

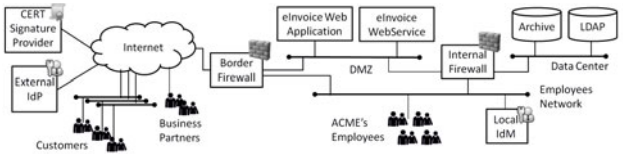


Fig. 1. ACME system

and a set of algebraic operators to specify the composition. In contrast we let the domain description determine the way different policies relate to each other, instead of explicitly relying on composition operators.

Data protection policies specifying confidentiality and integrity security requirements on data have also been analyzed for conflicts due to composition, for instance [11] leverage logic programming to check for consistency of WS-SecurityPolicy rules in the context of web services composition.

In [6] the definition of policy conflict is left up to the policy author. This work has similar characteristics to ours, providing a declarative approach for the definition of conflict types and encompassing conflicts among policies possibly belonging to different classes. Our approach is complementary as we explicitly focus on more kinds of transversal conflicts.

3 Use Case

A simplified but reasonably realistic scenario used to motivate and illustrate the contributions of this paper is as follows:

The imaginary service provider ACME offers an eInvoice service to B2B customers allowing them to securely exchange electronic invoices with their business partners, and to store them in a long term archive. Figure 1 outlines the system used to implement and deliver this service: ACME customers submit raw invoice data to a Web service, which is transformed into a PDF document, that is then digitally signed by the 3rd party service provider CERT, a company creating signatures compliant with respective national regulations. Business partners of ACME customers can access their invoices through a Web application. Both eInvoice components, Web service and Web application, run in an Internet-facing subnet. Other system components comprise the data center subnet with the long term archive and a user management system for eInvoice account credentials, and the employee subnet from where operations staff performs system maintenance. Traffic between subnets is mediated by application-level firewalls.

ACME faces many security and legal requirements stemming from a variety of different source, e.g., customers, suppliers, national regulations, or internal risk management. Table 1 presents a fraction of ACME's security policies addressing such requirements. The single policies already identify actors and components

Table 1. ACME security policy

Cod.	Policy	Policy Class	Motivation	Enforcement
Obl2	Archived invoices must be deleted according to maximum retention times	Obligation	Legal requirement	ACME account managers manually trigger the deletion process for their respective accounts
Fil1	Document exchanges with selected customers must be virus-checked	Filtering	Customer SLA	Perform virus scan at the border firewall for selected HTTP content-types
Fil3	It shall not be possible to establish outbound TCP connections from within internal networks	Filtering	Acceptable use policy	Block outbound connection requests at the border firewall
AuC1	Use of the CERT signature service requires prior authentication	Authentication	CERT service specific.	CERT's users must provide a 3rd party IDP SAML authentication token
AuZ1	Only authorized account managers shall have access to their customers' invoice archives	Authorization	ACME service specific.	File system access on the archive share is restricted to user groups maintained in ACME's local IDM
DaP1	Invoice information (raw, layouted, and signed) must remain confidential when sent over the Internet	Data protection	Data confidentiality	Web service communication is protected by WS-Security, access to critical elements of the Web application by SSL

Table 2. Conflicts in ACME policy and enforcement

Cod.	Conflict	Impact or Threat	Correction
Fil3, AuC1	Rigorous settings of the border firewall for outbound connections disrupt access to external authentication provider	Service availability or data integrity, depending on the application's error handling	Re-configure firewall
Fil1, DaP1	Border firewall fails virus scanning due to encrypted payload	Non-compliance with customer SLA	Partial system re-design
Obl2, AuZ1	Deletion of long term archives fails, due to wrong group membership of account managers in ACME's IDM	Non-compliance with customer SLA, information disclosure	Re-assign user groups

specific to ACME's organizational structure and service implementation, i.e., system. The policies expressed in natural language were specified by different stakeholders, and were taken as input for the selection and configuration of appropriate enforcement mechanisms, an activity concerning different system architecture layers, and again done by different stakeholders.

The specification of stated policies, the alignment of various stakeholders, and the selection and configuration of enforcement mechanisms is a tedious and error-prone process, often involving manual work and lacking tool-support. Any failure in this process can disrupt operation, violate required security properties or result in non-compliance, and on-going changes at both requirements and system-side do not facilitate this difficulty either. In fact, deficiencies of these processes are responsible for a significant share of IT operations costs [9], and for a significant share of vulnerabilities present in real-world systems [2]. In Tab. 2 we collected a selection of policy and enforcement conflicts that may occur at ACME, some of them can be fixed by means of configuration, others require partial system re-design.

4 Transversal Conflict Detection

Transversal conflicts are inconsistencies due to the implicit composition of security policies belonging to different classes and expressed in different languages, driven by the environment on which they apply. Informally, the `Obl2, AuZ1` conflict in Tab. 2 arises since, at the same time, a user *must access* a service because of the obligation policies, but *cannot access* it because of the filtering rule. Details like IP addresses, network ports or protocols, access control actions or targets are not directly used for conflict detection; in contrast they are essential in order to drive both the *must access* and *cannot access* conclusions according to (i) the conflicting policies and (ii) the status of their application domain. Our approach towards the formalization of transversal policy conflict detection therefore aims at providing a logic framework where this problem is equivalent to satisfiability checking. This framework is composed by a **Domain Description Model** (DDM), several **Class-Specific Policy Models** (CSPMs) and a **Conflict Specification Model** (CSM).

4.1 Domain Description Model

The DDM enables to specify the interlinked components of the policy-managed infrastructure. Figure 2 depicts a UML representation of the DDM meta-model.

We now describe a possible first order logic representation of such meta-model, in the following we will use bold font-face for predicates, italic for universally quantified variables and typewriter style for constants. Predicates $\mathbf{Proto}(p)$ and $\mathbf{DepOn}(p, p')$ can represent a generic network protocol stack, in which protocols belonging to upper ISO/OSI layers depend on (are encapsulated by) lower layers of the stack. Protocols have a number of *Fields* like the source and destination addresses for IP, the port number for TCP or the elements of a HTTP message (method, URI, headers, body, etc.). Predicates $\mathbf{Fld}(f)$, $\mathbf{Srv}(s)$ and $\mathbf{Bnd}(s, f, v)$ describe how services are linked to the set of protocols through which they are accessible on the network. This is done by specifying for each service s the actual values which protocol fields are bounded to ((f, v) couples), like the services' IP address when binding to the IP destination address field or their URI when binding to the respective HTTP field. Subjects are represented via the $\mathbf{Sbj}(s)$ predicate and can be as well bounded to protocol fields, hereby modeling their interactions with services over the network. Similar predicates are defined for roles, actions and resources. Services and users are located in network zones

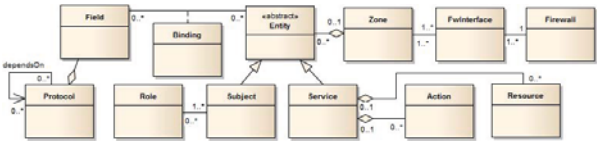


Fig. 2. Domain Description UML meta-model

by the means of the two predicates $\mathbf{Zone}(z, \{n_1, ..., n_n\})$ and $\mathbf{InZ}(s, z)$, specifying that the zone z encompasses a number of IP subnets having network addresses n_1 to n_n and harbors a service s. Zones connect to each other through firewalls ($\mathbf{Fw}(f)$ predicate) and firewall interfaces ($\mathbf{FwIf}(i)$ predicate). Finally $\mathbf{FwHasIf}(f, i)$ and $\mathbf{ZIf}(z, i)$ link together zones, interfaces and firewalls.

New predicates can be inferred providing meaningful deductive capabilities we require for our DDM. For instance $\mathbf{SrvIf}(i, s, f, v)$ expresses that a service s communicates through an interface i, hereby using a protocol who's field f is bound to value v. We also define the $\mathbf{UpLayer}^+$ predicate as the transitive closure of **DepOn** and therefore inferred for all the protocols p lying above the protocol p'' in the stack

$$\begin{aligned} \mathbf{Srv}(s) \wedge \mathbf{FwIf}(i) \wedge \mathbf{ZIf}(z, i) \wedge \mathbf{InZ}(s, z) \wedge \mathbf{Bnd}(s, f, v) &\rightarrow \mathbf{SrvIf}(i, s, f, v) \\ \mathbf{DepOn}(p, p') &\rightarrow \mathbf{UpLayer}^+(p', p) \\ \mathbf{DepOn}(p, p') \wedge \mathbf{UpLayer}^+(p'', p') &\rightarrow \mathbf{UpLayer}^+(p'', p) \ . \end{aligned} \tag{1}$$

Building such a domain model of a realistic operating infrastructure is in general a complex task, as heterogeneous pieces of information need to be fetched and integrated. However this activity can be partially automated, for instance information about network topology or user accounts can be retrieved from network discovery and configuration management tools [7]. Building the model of services protocols could instead be eased by leveraging the work carried on by [10], providing a language for representing the Internet protocol stack.

4.2 Class-Specific Policy Models

The purpose of the specific policy languages is to provide a declarative formal representation for each policy class, being as close as possible to the typical languages in which they are commonly expressed.

Authorization and Obligation policies. We derive the definition of authorization and obligation policies from the OASIS XACML specification [12]. We only consider declarative parts and we do not model XACML's grouping structures (like `PolicySet`), as well as the policy combination algorithms. We represent authorizations by the means of the $\mathbf{AuthZ}\big(s, a, r, [+|-]\big)$ predicate, stating that a subject s is either authorized or forbidden to perform action a on resource r, depending on the value of the result parameter being either *permit* $(+)$ or *deny* $(-)$. Obligations in XACML are attached to authorization policies, describing the actions that should be performed by the data controller after getting the access control decision (e.g., retention time, notification, etc). Since the XACML specification does not define the semantics of such actions, we refer to the PPL language [13], which extends XACML's obligation definition according to the type of access to data and the purpose of usage. We assert obligations as $\mathbf{Oblig}(s, a, r)$, meaning that subject s must perform action a on resource r.

Filtering. In this paper we focus on two classes of filtering policies, namely network layer and web application layer filtering, which are likely to appear in real scenarios. A network layer filtering rule $\mathbf{L3F}\big(i_s, i_d, a_s, a_d, p, p_s, p_d, [+|-]\big)$ applies between a source and destination interfaces (i_s and i_d) of a firewall and

state whether source and destination IP addresses (a_s and a_d) are allowed ($+$) or denied ($-$) to communicate on source and destination ports (p_s and p_d) for a given transport protocol (p). Web application filtering policies are instead stemmed from typical rule languages of web application firewall devices, like the Cisco ACE Web Application Firewall products [5], and are represented by the predicate $\mathbf{WAF}\big(i_s, i_d, f, o, v, [+|-]\big)$. Different parts of the application layer payload may be inspected by the means of rules comparing specific protocol fields (f) with fixed values (v) by the means of comparison operators (o).

Data Protection. The data protection policies we consider specify how services enforce confidentiality and integrity on data exchanged over the network. The $\mathbf{DP}\big(s, p, c, i\big)$ predicate states that a service s performs encryption and message authentication with a cipher suite identified by the couple of ground terms c (confidentiality) and i (integrity) over messages exchanged on protocol p.

Example. The following example shows a representation of the filtering and data protection policies stated in Tab. 1 according to the above formalization

$$\begin{aligned} &\texttt{Fil1:}\ \mathbf{WAF}\big(\texttt{ext}, \texttt{dmz}, \texttt{HTTP\_POST}, \texttt{sig}, \texttt{`signature regexp'}, -\big) \\ &\texttt{Fil3:}\ \mathbf{L3F}\big(\texttt{dmz}, \texttt{ext}, \texttt{any}, \texttt{any}, \texttt{TCP}, \texttt{any}, \texttt{any}, -\big) \\ &\texttt{Dap1:}\ \mathbf{DP}\big(\texttt{`eInvoice WebApp'}, \texttt{HTTPS}, \texttt{AES-256}, \texttt{SHA1}\big)\ . \end{aligned} \tag{2}$$

4.3 Conflict Specification Model

Transversal policy conflicts, as the ones listed in Tab. 2, can be seen as incompatibilities among properties abstracting away the concrete details of the different CSPMs. We describe these properties by the means of two CSM predicates, namely $(\neg)\mathbf{access}(u, s)$ and $(\neg)\mathbf{inspect}(i, p)$. The former states whether a subject u accesses or not a service s, the latter represents the ability or denial to perform traffic inspection on protocol p at the interface i. We then map transversal conflict detection to the consistency checking problem of CSM formulas.

In order to perform this mapping, we need to link each policy to a CSM predicate, leveraging the domain information as it provides the source of interdependencies between policy classes. The following rules exemplify the expressivity of such a language and capture the particular kind of transversal conflicts happening between web application filtering and data protection policies, such as the `Fil1, Dap1` conflict in Tab. 2

$$\begin{aligned} \mathbf{FwIf}(i) \wedge \mathbf{Proto}(p) \wedge \mathbf{Fld}(f) \wedge \mathbf{FldOf}(f, p) \wedge \mathbf{WAF}\big(\_, i, f, \_, \_, \_\big) &\rightarrow \mathbf{inspect}(i, p) \\ \mathbf{FwIf}(i) \wedge \mathbf{Proto}(p') \wedge \mathbf{UpLayer}^{+}(p', p) \wedge \mathbf{Srv}(s) \wedge \mathbf{Fld}(f) \wedge \mathbf{FldOf}(f, p') & \\ \wedge \mathbf{SrvIf}(i, s, f, \_) \wedge \mathbf{DP}(s, p', \_, \_) &\rightarrow \neg\mathbf{inspect}(i, p)\ . \end{aligned} \tag{3}$$

In case, given a set of DDM and CSPM predicates, both $\mathbf{inspect}(i, p)$ and $\neg\mathbf{inspect}(i, p)$ can be inferred, this kind of conflict is detected, being generated by the policies in the left hand side of the rules.

5 Conclusion

In this paper we proposed to address the problem of detecting transversal policy conflicts, happening whenever policies belonging to different classes cannot be enforced in the same environment because they imply contemporary conflicting behaviors. We herewith complement existing related work on the detection of conflicts between security policies within single classes of policies.

We proposed a framework composed by a Domain Description Model (DDM), several Class-Specific Policy Models (CSPMs) and a Conflict Specification Model (CSM). Policy conflicts are detected as inconsistencies between CSM predicates, inferred by rules linking the CSPMs with the DDM. Our approach is extensible with respect to both the set of encompassed policy classes and to the number of detected conflicts, since new CSPMs and new inference rules for conflict detection can be added independently to the framework. Policy designers can therefore specify new conflicts according to their specific security needs.

On-going and future work comprise the identification of a decidable fragment of logic able to capture the requirement of the proposed strawman design, such as Datalog (possibly with negation). We also plan to cover more realistic scenarios by enhancing the expressive power of our domain description and policy models.

References

1. Al-Shaer, E., Hamed, H., Boutaba, R., Hasan, M.: Conflict classification and analysis of distributed firewall policies. IEEE JSAC 23(10), 2069–2084 (2005)
2. Baker, W.: The 2009 data breach investigations report. Verizon Security Blog (2009), http://securityblog.verizonbusiness.com/2009/04/15/2009-dbir/
3. Bandara, A.K., Lupu, E.C., Russo, A.: Using event calculus to formalize policy specification and analysis. In: POLICY. IEEE (2003)
4. Bonatti, P., De Capitani di Vimercati, S., Samarati, P.: An algebra for composing access control policies. ACM Trans. Inf. Syst. Secur. 5, 1–35 (2002)
5. Cisco: Cisco ACE Web Application Firewall User Guide. Tech. Rep. OL-16661-01, Cisco Systems, Inc., San Jose, USA (2009)
6. Davy, S., Jennings, B., Strassner, J.: Using an information model and associated ontology for selection of policies for conflict analysis. In: POLICY (2008)
7. Garbani, J.P., Mendel, T.: Change and configuration management. Tech. rep., Forrester Research, Inc. (2004)
8. Kolovski, V., Hendler, J., Parsia, B.: Analyzing web access control policies. In: WWW, pp. 677–686. ACM (2007)
9. Plodík, P.: IBM cloud computing (2010), http://www.itforpeople.cz/wp-content/uploads/2010/09/IBM-Plodik-Cloud.pdf
10. Risso, F., Baldini, A., Bonomi, F.: Extending the netpdl language to support traffic classification. In: GLOBECOM. IEEE (2007)
11. Satoh, F., Tokuda, T.: Security policy composition for composite services. In: ICWE, pp. 86–97. IEEE (2008)
12. Simon Godik, T.: OASIS eXtensible Access Control Markup Language (XACML) Version1.0. Tech. rep., OASIS (February 2003)
13. Trabelsi, S., Njeh, A., Bussard, L., Neven, G.: The PPL Engine: A Symmetric Architecture for Privacy Policy Handling. In: W3C Workshop on Privacy and Data Usage Control (2010)

Challenges in Implementing an End-to-End Secure Protocol for Java ME-Based Mobile Data Collection in Low-Budget Settings

Samson Gejibo[2], Federico Mancini[1],
Khalid A. Mughal[1], Remi Valvik[1], and Jørn Klungsøyr[2]

[1] Department of Informatics, University of Bergen, Norway
{federico,khalid}@ii.uib.no, remi@valvik.org
[2] Centre for International Health, University of Bergen Norway
{samson.gejibo,mihjk}@cih.uib.no

Abstract. Mobile devices are having a profound impact on how services can be delivered and how information can be shared. Sensitive information collected in remote communities can be relayed to local health care centers and from there to the decision makers who are thus empowered to make timely decisions. However, many of these systems do not systematically address very important security issues which are critical when dealing with such sensitive and private information.

In this paper we analyze implementation challenges of a proposed security protocol based on the Java ME platform. The protocol presents a flexible secure solution that encapsulates data for storage and transmission without requiring significant changes in the existing mobile client application. The secure solution offers a cost-effective way for ensuring data confidentiality, both when stored on the mobile device and when transmitted to the server. In addition, it offers data integrity, off-line and on-line authentication, account and data recovery mechanisms, multi-user management and flexible secure configuration. A prototype of our secure solution has been integrated with openXdata.

Keywords: Mobile Data Collection Systems, Mobile Security, secure communication protocols, secure mobile data storage, secure mobile data transmission, Java ME, openXdata, HTTPS, JAD.

1 Introduction

There are already a number of systems that allow data collection in the health sector using mobile phones and provide a server component to manage the collected data. However, none of these systems has a complete security solution to guarantee data confidentiality, integrity, availability and privacy both on the client and on the server side. In this paper, we present the challenges in implementing the *openXSecureAPI* (which from now on we will refer to as simply API), based on the secure protocol proposed in [5], which can be used to add a security layer in existing Mobile Data Collection Systems (MDCS).

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 38–45, 2012.

Here, we focus on the mobile side of the API, which is developed for Java Mobile Edition(Java ME)[7] based applications and assumes that the main use of the application is the collection of data by an authorized user through predefined forms. In other words, we do not consider systems where data is gathered by automated sensing systems. The API is designed by considering several security challenges in mobile data collection where low-end mobile phones are deployed and the projects run on very constrained budgets. The details can be found in [5].

For this work we collaborated with openXdata [6], a MDCS that is primarily designed for data collection using low-end Java-enabled phones in low-budget settings. The openXdata community shared with us their field experience regarding the deployment of their mobile data collection tools and various technical details of their client and server applications.

The challenges and solutions are covered in Section 2, where, in order to make this article self-contained, we also mention how the different parts of the API reflect the underlying protocol, and which security and usability requirements are addressed. Finally, we present some experimental results we obtained by testing a basic data collection client that uses our API on various mobile devices. In this paper we assume that the reader is familiar with Java ME technology and terminology.

2 Implementation Challenges and Solutions

Most of the challenges we faced during the implementation required finding the right balance between flexibility, efficiency and usability, while not compromising security. In general, we decided to give more emphasis to flexibility, in order to create an API that is easy to use and integrate with different clients, at the cost of some efficiency. In the following sections, we discuss some of issues we consider to be highly relevant.

2.1 Cryptography API Providers

Early versions of Java ME did not support a cryptography API. However, since the introduction of MIDP 2.0, the Security and Trust Services API (SATSA) has been developed and added to the Java ME platform as an optional package that provides some basic cryptographic primitives. Besides, since it is implemented as part of the phone libraries, its use does not affect the memory footprint of the application. Unfortunately, very few low-end mobile phones actually support it. On the other hand, Bouncy Castle (BC)[4] provides a flexible lightweight cryptography API which is extensively used in Java ME applications. Since it is an external API, it allows us to develop device independent solutions, but its libraries can add a significant memory overhead.

In order to allow for future compatibility, we opted for an hybrid solution. Our API provides an interface that defines the required cryptographic operations, but leave the actual implementation open, with BC as default provider. However, if

the phone supports the SATSA package, our API can automatically switch to that implementation. So, even though memory footprint is not reduced (BC is always loaded anyway), one can gain in performance by using the phone built-in libraries. Using two different implementations, means also that we are forced to use only algorithms supported by both libraries. In particular: RSA for public key encryption, AES in padded CBC mode with initializing vector (IV) for symmetric cryptography, SHA1 digest, Hash-based Message Authentication Code (HMAC) based on SHA1 digest. Only BC provides an adequate Pseudo Random Number Generator (PNRG) and Password Based Encryption based on PKCS#5.

2.2 Key Generation

A critical issue when using cryptography on a mobile phone is the generation of good random keys, since mobile phones do not have good sources of entropy [1], and even if they have, J2ME might lack the necessary libraries to access them. In the proposed solution, we generate a strong seed on the server and send it securely to the client whenever possible, so that strong cryptographic keys can be generated. Every user will have their personal set of seeds stored encrypted in their key store, so that the PNRG can be seeded also at boot time, and in a different way for each user. This solution avoids putting the burden of generating the seed on the user by pressing random keys or playing a game, or turning on the camera or the microphone to collect entropy, as it has been suggested in the literature.

2.3 Secure Data Upload and Download

The API is designed to be flexible and support both HTTPS and the protocol proposed in [5]. We offer a `SecureHttpConnection` class that can be wrapped around a `HttpConnection`. If the connection is HTTPS, the `SecureHttpConnection` will behave in the same way as a normal `HttpsConnection` object would. If however it is not HTTPS, any request headers or data written to the connections output stream will be encrypted prior to being sent to the server by using the protocol presented in [5]. The API is designed so that the client developer would use the `SecureHttpConnection` object in the same way as an `HttpConnection` object. This makes for easy and transparent integration into existing systems. We are able to create a secure tunnel by changing only two lines of code in the existing openXdata client. The following snippet shows openXdata client before the integration (no encryption is used):

```
HttpConnection  con = (HttpConnection)Connector.open(URL);
((HttpConnection)con).setRequestMethod(POST);
```

This snippet shows openXdata client after integration with the secure API:

```
HttpConnection con = (HttpConnection)Connector.open(URL);
SecureHttpConnection secCon = new SecureHttpConnection(con,
                          SecureHttpConnection.RequestType);
secCon.setRequestMethod(POST);
```

By using the `SecureHttpConnection` class, the client can now provide a secure data transfer for any project whether they can afford to use SSL certificates (and therefore HTTPS) or not.

Initially we had thought to exploit the fact that data is encrypted on the phone, and send it as it was, to avoid further encryption and decryption operations. However, that could not be done without significant changes to the existing client code, and without exposing many cryptographic operations to the developers. Not to mention that the same key used for the storage would be re-used for transmission, raising security concerns and key management issues. Hence, even if this could give better performance, it could also affect the security of the API and its usability. We chose, therefore, to simply wrap the data sent from the client in a secure connection, which, despite some extra traffic, allows also for a complete decoupling between the secure layer and the client requests.

Notice also the second parameter of the `SecureHttpConnection` constructor: `SecureHttpConnection.RequestType`. When this parameter is specified, our API can automatically generate some predefined requests that can be used for various operations: user registration; password recovery and server authentication as described in [5].

2.4 Secure Storage

The storage has been designed to accommodate typical scenarios in mobile data collection. In particular that multiple users should be allowed to use the same mobile device, that the same user can use multiple mobile devices and that Internet access might always not be available. This means that mobile devices can no longer be considered private or personal to an user and that most of the data collection might have to be done off-line. From a security perspective this translates into the following concerns:

1. A mobile device must store some identification token to authenticate users off-line.
2. If a user loses the password, other users on the same device and their data should not be affected.
3. If users change their password on the server, possibly from a web application, the access to the mobile device should not be compromised.
4. Even if the password is lost, it should always be possible to recover the encrypted data stored on the mobile phone by some authorized entity.

A scheme that satisfies all the above requirements is described in [5] and implemented in the API, which offers tools to register a new user so that a new personal secure storage is initialized according to such scheme. The client application simply needs to pass username and password to our `registerUser()` method, and thereafter use our `login()` method to get access to a user's data. The login method authenticates the user and creates a session object with a user's key, that the API will use to handle the secure stores and secure HTTP sessions. This is also independent from the authentication method used on the server. The data is encrypted with symmetric encryption, and the encryption

key is protected by a password-based key. This means that losing the password does not prevent access to the data if the data encryption key has been saved, for example, on the server. Notice, however, that the overall security of the data still depends on the strength of the password, and as long as off-line local authentication is required on the mobile phone, and smart cards are not supported by the phone, this is a problem that cannot be solved. When it comes to the actual storage of data in the RMS analogue to the `SecureHttpConnection` class, we offer a `SecureRecordStore` class, that can be used to wrap the data in a secure way. Every write/read operation will, respectively, encrypt the input data before writing it in the actual RecordStore object, and decrypt it before returning it to the client. The API also takes care of checking whether the current user has permissions to write in that storage and handles the corresponding keys. All of this happens completely transparent way for the client.

The drawback of this approach is that the user has no control over the data encryption, so, every time something is read or written from the secure store, a cryptographic operation is performed. This can be a computational overhead if a search must be done across the stored data, since several decryption operations are required. This happens, for instance, when a menu must be generated to show the users which form values have been saved in the record store. To mitigate this problem, we offer to store the data with a label that describes it. All the labels are stored as a list in a single encrypted record, so that only this list needs to be decrypted to generate a menu, rather than all the records.

One advantage, instead, is that the client is not forced to pre-process the data and store it in a specific format or in a dedicated record store. The only assumption we make, is that each user has a dedicated record store, so that a unique key can be assigned to it. This makes the key and permission management much easier for the API.

An alternative solution we tested was to offer methods that took a byte stream and returned an object containing the encrypted stream plus a set of fields to manipulate it, so that the developer could have direct control over the encrypted data. However this idea was discarded because it would have required substantial refactoring in the existing client, and it could have potentially introduced security issues if the data were manipulated incorrectly.

2.5 Modularity of the API

While designing the API we focused also on making it modular. We tried to make the different packages that constitutes the API as independent as possible, so that a client using only the secure communication, would not import the secure storage libraries and vice-versa, thus minimizing the final size of the application.

2.6 API Integration

Figure 1 shows how our API creates a secure layer on top of the existing application layer, taking as example our work with the openXdata client.

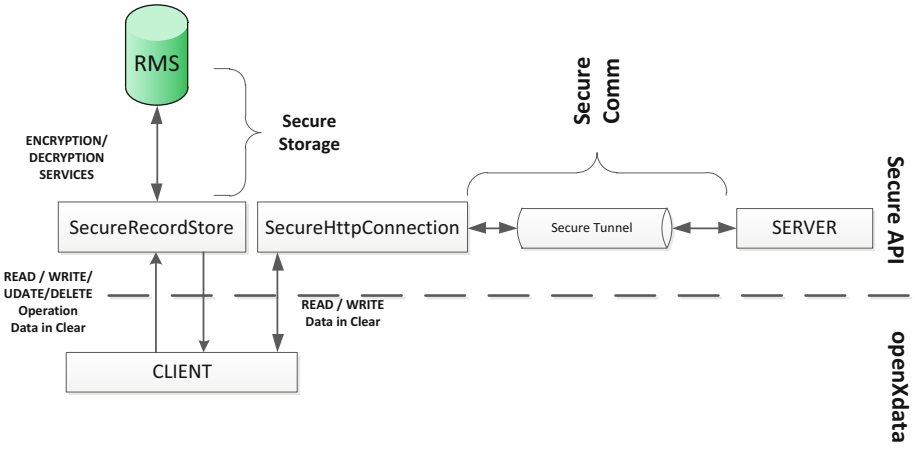


Fig. 1. Openxdata - secure API integration architecture

3 Preliminary Performance Test

In this section we report the results of some preliminary tests we ran in order to analyze the performance of the API on devices with different hardware specifications and price categories. The results are summarized in Table 1. Note that the phones used for the benchmark are phones that are most likely to be deployed on the field by openXdata. No smart phones are therefore considered. Also what we define as "powerful" phones, are only there to put the other results into perspective, since they are not likely to be used due to their high cost. It is clear that with the given parameters the performance of the API is barely acceptable on the least powerful phone (2760), but it already has a more than acceptable performance compared to an equally cheap and only slightly more powerful device (2330c). It is interesting that the processor speed (3rd row in the table) is not always the most important factor. The most expensive and powerful mobile phone (E-63) we used in the test, has very poor performance due to the high amount of time used to create new records in the record store (4th row in the table). We have not tested our protocol when a HTTPS connection is used, but a simple SSL handshake took on average 12 seconds on all the devices tested, which is comparable with a complete Server Authentication step of the protocol in [5] on the slowest phone. It is also clear that the bottle neck in the various transactions is the RSA encryption, but no much optimization can be done in this regard. The key cannot be reduced to less then 960 bits, i.e., the smallest size required to guarantee that all protocol requests can be encrypted, and, in general, it is not recommended to use less than 1024 bit anyway.

Table 1. Test results

Phone model (Nokia)	**2760**	**2330c-2**	**2730c**	**3120c**	**E-63**
Price ($)	50	<50	89	120	180
Processor speed (Mhz)	0,8	4,6	67,7	68,8	125,7
Time to create 20 records of 100 bytes on the phone (ms)	120	49	16	6	2573,9
RSA Encryption with 1024 bits key (ms)	3702	562	92	79	265
16 bytes AES encryption of a 100 bytes form (ms)	58	19	5	5	315
16 bytes AES decryption of a 100 bytes form (ms)	28	22	5	11	333
PKCS-5 password-based-encryption (100 iterations) (ms)	878	151	18	18	344
Processing time for Sever Authentication (ms)	11611	6306	5470	5021	11297
Processing time for User Registration (ms)	9226	6153	5392	3523	4186
Uploading 356 bytes of forms (ms)	3895	4989	5038	3295	1588
Downloading 2880 bytes of forms (ms)	4347	2868	4424	3023	1513

4 Related Work and Conclusions

In general, all modern smart phones equipped with operative systems like Blackberry, Android and iOS provide a crypto API to develop secure applications. However, we are developing a secure solution for the Java ME platform, which lacks support for any kind of data security [2,10], and we target low-end phones, so that solutions that might be adequate for high-end phone like smart phones, are not an option for our context.

The solution we implemented is based on a custom protocol developed by considering the specific constraints of MDCS [5], but it makes almost no assumptions about how or where data are stored, or how the communication layer of an existing application is implemented. This guarantees wide compatibility. Besides, the different secure solutions that it offers are very modular, and can be used independently to fit the needs of MDCS with different security requirements. We have also developed our own prototype MDCS using the API, and tested it on various phones with different settings in order to collect experimental data on the performance of the API. The results are encouraging, since the performance with the default security settings was acceptable also on very low-end phones, and the openXdata integration is proceeding smoothly.

Other approaches to secure applications having the Java ME platform as their target have been proposed in the literature [8,3,9], but it is easy to see that they are all tailored for specific target applications, and they are nowhere as extensive and flexible as our API. Besides, as far as we know, the solutions proposed in these works have not been employed in actual systems, while our

API is currently being used to develop a secure client for openXdata, that next year will be deployed in the field and carefully tested in a real project. The results from this test will be used to optimize the code and develop new features for the API. For example, mechanisms for automatically configuring the security settings on each device, in order to maximize security without compromising usability, are currently being studied.

References

1. Crocker, S., Schiller, J.: RFC 4086 - randomness requirements for security (2005), http://www.ietf.org/rfc/rfc4086.txt
2. Egeberg, T.: Storage of sensitive data in a Java enabled cell phone. Master's thesis, Høgskolen i Gjøvik (2006)
3. Itani, W., Kayssi, A.: J2ME application-layer end-to-end security for m-commerce. Journal of Network and Computer Applications 27(1), 13–32 (2004)
4. T. Legion Of the Bouncy Castle, http://www.bouncycastle.org/ (accessed March 2011)
5. Mancini, F., Mughal, K., Gejibo, S., Klungsoyr, J.: Adding security to mobile data collection. In: Proceedings of Healthcom 2011 - 13th IEEE International Conference on e-Health Networking Applications and Services, pp. 86–89 (June 2011)
6. openXdata, http://www.openxdata.org (accessed March 2011)
7. Oracle. Java ME, http://www.oracle.com/technetwork/java/javame/index.html (accessed March 2011)
8. Shah, S.M.A., Gul, N., Ahmad, H.F., Bahsoon, R.: Secure Storage and Communication in J2ME Based Lightweight Multi-Agent Systems. In: Nguyen, N.T., Jo, G.-S., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2008. LNCS (LNAI), vol. 4953, pp. 887–896. Springer, Heidelberg (2008)
9. Wang, Z., Guo, Z., Wang, Y.: Security research on j2me-based mobile payment. IEEE Communication Society 2(2), 644–648 (2008)
10. Whitaker, B.: Problems with mobile security #1 (July 2007), http://www.masabi.com/2007/07/13/problems-with-mobile-security-1/ (accessed March 2011)

Runtime Enforcement of Information Flow Security in Tree Manipulating Processes*

Máté Kovács and Helmut Seidl

Technische Universität München, Germany

Abstract. We consider the problem of enforcing information flow policies in XML manipulating programs such as Web services and business processes implemented in current workflow languages. We propose a runtime monitor that can enforce the secrecy of freely chosen subtrees of the data throughout the execution. The key idea is to apply a generalized *constant propagation* for computing the public effect of branching constructs whose conditions may depend on the secret. This allows for a better precision than runtime monitors which rely on tainting of variables or nodes alone. We demonstrate our approach for a minimalistic tree manipulating programming language and prove its correctness w.r.t. the concrete semantics of programs.

Keywords: Semi-structured data, information flow control, runtime enforcement.

1 Introduction

As the application of computer based workflow and information storage solutions becomes ubiquitous in today's practice, the successful operation of organizations depends heavily on the correct functionality of these systems. In particular, invaluable pieces of information are stored and manipulated by computer systems that are in the same time connected to the Internet, so it is our valid expectation that the security of our data should be guaranteed. Yet, there are numerous problems and challenges that need to be tackled before we can achieve this goal.

This paper is concerned with the problem of enforcing information flow security in contemporary business workflows implemented e.g. using the Web Services Business Process Execution Language (BPEL) [5]. One common property of these workflows is that the data they manipulate is inherently semi-structured. The common data representation and exchange format is XML document trees, especially if Web services are used for communication. In our opinion, the challenge in this environment is to provide sufficient flexibility to enable the specification of secrecy in terms of parts of document trees, besides enforcing information flow policies.

* This work was partially supported by the German Research Foundation (DFG) under the project SpAGAT (grant no. FI 936/2-1) in the priority program "Reliably Secure Software Systems – RS3".

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 46–59, 2012.

There are established solutions for providing information flow control in structured and object-oriented languages like the Java extension Jif [4, 23], and ValSoft/Joana [15]. Further results in this area include among others [8, 10, 13, 30]. These approaches define information flow policies in terms of variables. However, if a complex data structure consisting of pieces with different security levels is encoded into the value of a single variable, policies associating security levels with variables might be too restrictive, because they would be forced to consider the secrecy level of the complete data structure to be equivalent with that of the most confidential member of the complete structure. The solution we propose associates secrecy levels to specific positions in the tree structured data during runtime.

The authors of [28] note that runtime approaches [13, 26, 30] are on the rise again, because they can be more permissive than static solutions, while providing the same guarantees. In our case this statement especially holds, because our monitor takes advantage of the fact that during runtime data instances are available. In principle, our monitor executes programs in parallel to the operational semantics of the language, while maintaining a state which only depends on the public data. Accordingly, we call the state of the monitor the *public* or *low view*. The computation of the low view is challenging in the case, when the result of a branching construct, whose condition depends on the secret, is about to be computed. In this case we apply a dataflow analysis procedure, which is a refinement of constant propagation (see e.g. [29]) for handling semi-structured data. The key difference is the hierarchic nature of lattice elements, which aligns to our purpose of preventing information leakage in tree-manipulating programs. Moreover, we gain precision by only considering a modification of a subtree inside a secret-dependent branch as potentially secret, if it does not occur in the other alternative as well, and thus must be excluded from the public view. In summary, our paper provides the following innovations:

- A runtime monitor is introduced to support the specification of information flow policies in terms of tree-like data and their enforcement.
- The enforcement mechanism applies a generalized variant of constant propagation in order to compute the public view of the state at the end of branching instructions.

The paper is organized as follows. In Section 2 we introduce a minimalistic language for which the construction of the monitor is demonstrated. In Section 3 we illustrate the intuition behind our solution through an example, a fragment of a hypothetical paper submission system. We formalize the approach in Section 4, and in Section 5 we discuss the guarantees the monitor provides us. In Section 6 we relate our work to others and conclude.

2 Preliminaries

In order to minimize the formal overhead, we demonstrate our approach with a small programming language for manipulating trees. Here, we consider binary

trees only. This is no restriction in generality, since binary trees are in one-to-one correspondence with unranked trees. Unranked trees in turn can be considered as the natural internal representation of XML documents.

Definition 1 (Binary trees). *The set of binary trees* $\mathfrak{B}_{\Sigma,\{\#\}}$ *over the finite set of binary alphabet elements* Σ *and the set of nullary alphabet elements* $\{\#\}$ *is defined by the language:*

$$t ::= \# \mid \beta(t_1, t_2)$$

where $\beta \in \Sigma$ *and* $t_1, t_2 \in \mathfrak{B}_{\Sigma,\{\#\}}$.

The left hand side of Figure 1 displays an unranked document tree representing a scientific publication in a database, while the right hand side shows its binary equivalent in the *first-child/next-sibling* encoding. The binary tree $\beta(t_1, t_2)$ is interpreted as an unranked forest, where the root of the leftmost tree is labeled β. Its content is the unranked variant of t_1, while the forest on its right hand side is the unranked variant of t_2. The only nullary node labeled # represents the empty forest.

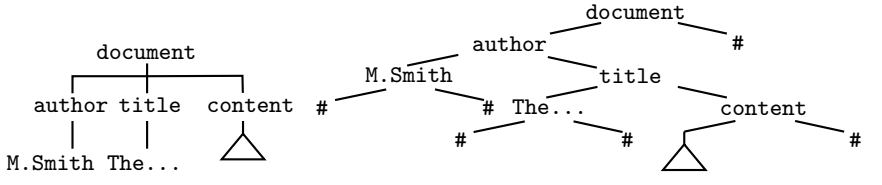


Fig. 1. Encoding unranked trees into binary trees

Our goal is to design a monitor which enforces confidentiality in programs manipulating tree structured data. In principle, the presented models may be applied also to interactive processes. In this paper, though, we disregard interactive aspects, like communication primitives. Instead, we focus on the key aspects of the monitor construction. Therefore, we assume that the input processed by programs is given in the initial configuration, and the result is presented in the final configuration.

(tree expressions)	$e ::= x \mid \texttt{\#} \mid \beta\texttt{(}x_1\texttt{,}x_2\texttt{)} \mid x\texttt{/1} \mid x\texttt{/2}$
(boolean expressions)	$b ::= \texttt{top(}x\texttt{)=}\alpha$
(program)	$p ::= \varepsilon \mid c\texttt{;}p$
(commands)	$c ::= x\texttt{<-}e \mid \texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ p \mid \texttt{while}\ b\ \texttt{do}\ p$

Fig. 2. A minimal language optimized for tree manipulation

A grammar for our minimalistic programming language is shown in Figure 2. A tree expression is the content of a variable x, the nullary node #, or a binary

tree composed of a new root labeled β having the contents of variables x_1 and x_2 as subtrees. The expressions x`/1` and x`/2` refer to the first and second subtree of the tree stored in variable x. Boolean expressions may test the label of the root of the tree stored in a variable. A program, generated by the nonterminal p, is a possibly empty sequence of commands. A command can be an assignment x`<-`e of the value of a tree expression e to the variable x, a conditional execution of alternative programs `if`, or an iteration `while`.

The semantics of the language is defined by transition relations $cfg \rightarrow_\rho cfg'$ between configurations of the form $\langle p, \sigma\rangle$, where p is the program to be executed on the state σ. In case of final configurations, where $p = \varepsilon$ we simply write σ instead of $\langle \varepsilon, \sigma\rangle$. The state $\sigma : (Var \rightarrow \mathfrak{B}_{\Sigma,\{\#\}}) \cup \{\lightning\}$ is a mapping from the set of variables of the program to binary trees, or the error state, denoted by $\lightning$, symbolizing that a runtime error has occurred during the execution.

The semantics of assignments and boolean expressions is shown in Figure 3. A boolean expression transforms a state into a boolean value (`t` or `f`), while an assignment transforms the state before the assignment into the state after the assignment. The new state σ' is equal to the original σ in all variables except for the one on the left hand side of the assignment. Such a modification of the mapping σ at argument x is denoted by $\sigma' = \sigma[x \mapsto v]$, where v is the new value. This new value for an assignment is obtained by evaluating the expression on the right hand side within the state σ.

$$[\![\texttt{top(}x\texttt{)=}\alpha]\!]\sigma = \begin{cases} \texttt{t} & \alpha \in \Sigma \text{ and } \sigma(x) = \alpha(t_1, t_2) \text{ for some } t_1 \text{ and } t_2 \text{ or} \\ & \alpha = \texttt{\#} \text{ and } \sigma(x) = \texttt{\#} \\ \texttt{f} & \text{otherwise} \end{cases}$$

$$[\![x\texttt{<-}y]\!]\sigma = \sigma[x \mapsto \sigma(y)] \qquad [\![x\texttt{<-\#}]\!]\sigma = \sigma[x \mapsto \texttt{\#}]$$

$$[\![x\texttt{<-}\beta\texttt{(}x_1\texttt{,}x_2\texttt{)}]\!]\sigma = \sigma[x \mapsto \beta(\sigma(x_1), \sigma(x_2))]$$

$$[\![x\texttt{<-}y\texttt{/1}]\!]\sigma = \begin{cases} \sigma[x \mapsto t_1] & \text{if } \sigma(y) = \beta(t_1, t_2) \text{ for some label } \beta \\ \lightning & \text{otherwise} \end{cases}$$

$$[\![x\texttt{<-}y\texttt{/2}]\!]\sigma = \begin{cases} \sigma[x \mapsto t_2] & \text{if } \sigma(y) = \beta(t_1, t_2) \text{ for some label } \beta \\ \lightning & \text{otherwise} \end{cases}$$

Fig. 3. The semantics of assignments and boolean expressions

When dealing with structured data, runtime errors cannot be excluded. When processing binary trees, an error occurs, if a subtree of a leaf is about to be accessed by expressions x`/1` or x`/2`. In this case the new state is the error state ($\lightning$). The error state is atomic, it does not map variables to values any more. We assume on the other hand, that each variable has already received a value in the initial configuration. Accordingly, no error can be caused by a variable access during the execution.

The semantics of the programming language is shown in Figure 4. In the condition parts of the rules we use the relation $\rightarrow_\rho^*$, which denotes the reflexive and transitive closure of $\rightarrow_\rho$. A central rule of the semantics is S, which is

responsible for executing a program, in other words a sequence of commands. The remaining rules define the effects of individual commands. By rule E the error state is not modified by any command. Instead, it is passed over to the next command in the sequence, or to the final configuration. In case the state is not erroneous, the execution of an assignment is specified by the rule A. The rules WT and WF execute iterations, IT and IF execute conditional selections of alternative programs as it is usual in other structured programming languages.

$$\text{E: } \frac{\sigma = \lightning}{\langle c, \sigma\rangle \to_\rho \lightning} \quad \text{A: } \frac{\sigma \neq \lightning}{\langle x\texttt{<-}e, \sigma\rangle \to_\rho [\![x\texttt{<-}e]\!]\sigma} \quad \text{S: } \frac{\langle c, \sigma\rangle \to_\rho^* \sigma'}{\langle c\texttt{;}p, \sigma\rangle \to_\rho \langle p, \sigma'\rangle}$$

$$\text{WT: } \frac{\sigma \neq \lightning \quad [\![b]\!]\sigma = \texttt{t} \quad \langle p_t, \sigma\rangle \to_\rho^* \sigma'}{\langle \texttt{while } b \texttt{ do } p_t, \sigma\rangle \to_\rho \langle \texttt{while } b \texttt{ do } p_t, \sigma'\rangle}$$

$$\text{WF: } \frac{\sigma \neq \lightning \quad [\![b]\!]\sigma = \texttt{f}}{\langle \texttt{while } b \texttt{ do } p_t, \sigma\rangle \to_\rho \sigma}$$

$$\text{IT: } \frac{\sigma \neq \lightning \quad [\![b]\!]\sigma = \texttt{t} \quad \langle p_t, \sigma\rangle \to_\rho^* \sigma'}{\langle \texttt{if } b \texttt{ then } p_t \texttt{ else } p_f, \sigma\rangle \to_\rho \sigma'} \quad \text{IF: } \frac{\sigma \neq \lightning \quad [\![b]\!]\sigma = \texttt{f} \quad \langle p_f, \sigma\rangle \to_\rho^* \sigma'}{\langle \texttt{if } b \texttt{ then } p_t \texttt{ else } p_f, \sigma\rangle \to_\rho \sigma'}$$

Fig. 4. The semantics of the programming language

3 The Runtime Monitor

Since the seminal paper of Denning [11], the secrecy level of data is usually captured in terms of lattices. The simplest form of this lattice is $L \sqsubseteq H$, which allows to specify that pieces of information belonging to the lattice element L should not depend on those belonging to H.

Similarly to other runtime monitors e.g. [13, 26, 28, 30], in order to enforce information flow properties, we extend the configuration of the semantics of the language with an additional member, which maintains the secrecy information of the data stored in the state. This new member $D : (Var \to \mathfrak{B}_{\Sigma,\{\#,\square\}}) \cup \{\bot, \top, \lightning\}$, referred to as *monitor state*, assigns to every variable either a binary tree having the extra nullary alphabet element $\square$, or is one of the symbols $\bot$, $\top$ and $\lightning$. Intuitively, $D(x)$ stores the *public* upper part i.e., an upper part of the current value of x, which belongs to the security lattice element L, where the symbol $\square$ indicates subtrees possibly depending on the secret, and thus belonging to H. For this reason, we also call D the *public view*, because it only contains definitely public information. The monitor recalculates this value in parallel to the semantics for each configuration, and the result of the computation for principals belonging to the security lattice element L is presented by the final public view.

In the next paragraphs, we informally illustrate the functionality of the runtime monitor by an example. The code fragment in Listing 1 could be part of a paper submission system distributing the papers to reviewers. Let us suppose that reviewer 2 declared a conflict of interest with the author A. Mustermann, and therefore the distribution system is not allowed to send information to him about the content. Therefore, from the point of view of reviewer 2, the information on the topic of the paper of A. Mustermann is secret too. Let us suppose that the runtime monitor reaches line 3 of Listing 1 with monitor state:

$$D_0 = \{..., \texttt{topic} \mapsto \square, \texttt{rev1} \mapsto \#, \texttt{rev2} \mapsto \texttt{listElem}(\texttt{document}(...), \#), ...\}$$

```
1  empty<-#;
2  if top(author)=A_Mustermann then
3  if top(topic)=Databases then
4          rev2<-conflict(empty,rev2); rev1<-listElem(doc,rev1);
5   else   rev2<-conflict(empty,rev2); rev3<-listElem(doc,rev1);
6  ; else ;
```

Listing 1. Branching on a secret value

Because the conditional expression depends on the secret, constant propagation is carried out on this branching command. We can identify the value $\square$ with the top element of constant propagation expressing that the value is not constant and therefore, may leak information about the secret. After executing the branches we get:

$$\begin{aligned} D_{\texttt{then}} = \{..., \texttt{rev1} \mapsto \texttt{listElem}(\texttt{document}(...), \#), \\ \texttt{rev2} \mapsto \texttt{conflict}(\#, \texttt{listElem}(\texttt{document}(...), \#)), ...\} \\ D_{\texttt{else}} = \{..., \texttt{rev1} \mapsto \#, \texttt{rev2} \mapsto \texttt{conflict}(\#, \texttt{listElem}(\texttt{document}(...), \#)), ...\} \end{aligned}$$

After the join computation we have:

$$D = \{..., \texttt{rev1} \mapsto \square, \texttt{rev2} \mapsto \texttt{conflict}(\#, \texttt{listElem}(\texttt{document}(...), \#)), ...\}$$

Computing the join of two states can be done by replacing the values of variables where the two states differ, with the symbol $\square$. In this way, it is guaranteed that the monitor state after the branching construct is independent of the secret.

For the join computation, however, it is not necessary to replace the entire value of a variable with $\square$ if the two values differ only for certain subtrees. Figure 5 illustrates the join computation for the values of variable `authors` in monitor states $D_{\texttt{then}}$ and $D_{\texttt{else}}$ in such a situation. The variable contains a list of authors and their documents that they submitted. Let us suppose that the order of two authors has been accidentally exchanged in one of the two secret-dependent conditional branches. By computing the join, we only need to replace those members of the list which were exchanged, but we can leave the others as they are. In this way, we take the semi-structured nature of data into account and gain additional precision.

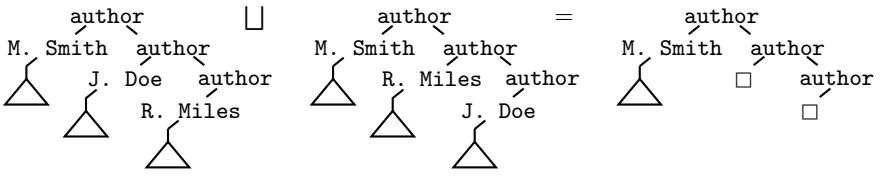


Fig. 5. The join on document trees, where the leaves labeled # are omitted for the sake of simplicity

Another advantage of our approach is the following. Because in the code fragment of Listing 1 variable `rev2` is assigned in a secret branch, many information flow analyzers would consider its value secret. The solutions similar to the type system of Volpano et al. [31] like Jif [4,23], SIF [10] and Paralocks [8] do so because the variable `rev2` has been assigned in an environment, where the program counter depends on the secret and therefore is high. Similarly behave runtime monitors [13,26,28,30] for the same reason. The program slicing [17] based solutions like Joana [15] do so, because of the control dependency edges from the conditional expressions to assignments. Our idea is based on the observation that in the final configuration the value of the variable `rev2` is independent of the value of `topic`. This could happen, perhaps, because the program noticed by the embedding branching decision, that the content of the paper is secret and behaved correctly. Accordingly, the observation of `rev2` does not give us information on the secret value. Our runtime monitor would consider the value of `rev2` as public, because it determines the confidential parts of values by means of the join computation after exiting from branching commands depending on secret values. There are approaches based on bisimulation, e.g. [18,21], allowing public assignments in secret branches, if the equivalence of the public effects of these branches is proved. Because program equivalence is in general undecidable, these solutions rely on syntactic approximations. In our solution if programs P and Q are equivalent, they do not read confidential variables, and they terminate, then the result of `if secret then` P `else` Q`;` is recognized public regardless of the syntactic representation of P and Q.

In the next section we formally elaborate the ideas introduced here.

4 Formal Treatment of the Monitor

In order to describe the runtime monitor formally, we need some more definitions. In the following, we view a tree t as a mapping from its positions $Pos(t)$ to the alphabet Σ of binary symbols or # and □, where the domain is a prefix closed subset of $\{1,2\}^*$ with the additional property that if $p_i 2 \in Pos(t)$ then $p_i 1 \in Pos(t)$ too. Accordingly, we use the notation $t(p)$ to refer to the alphabet element at position p of the tree t. If a node p of t has no successors, then $t(p)$ either equals # or □. We denote the *subtrees* rooted at the first and the second child of the root of t with $t/1$ and $t/2$ respectively.

Definition 2 (Preorder of trees). *If* $t_1, t_2 \in \mathfrak{B}_{\Sigma,\{\#,\square\}}$ *then* $t_1 \sqsubseteq t_2$ *holds if one of the following is true:*

- $t_2(\varepsilon) = \square$.
- $t_2(\varepsilon) \neq \square$ *and* $t_1(\varepsilon) = t_2(\varepsilon)$*, furthermore, if* $t_1(\varepsilon) \neq \#$ *then* $t_1/1 \sqsubseteq t_2/1$ *and* $t_1/2 \sqsubseteq t_2/2$.

In Definition 2 the symbol $\square$ occurs as an additional nullary element, which represents a secret subtree in the public view. Similarly to the state, the monitor state can also be erroneous, denoted by $\lightning$, meaning that the execution reached an inconsistent situation. It is also possible that the error state itself depends on the secret. This happens for instance, if one conditional branch of a decision depending on the secret exhibits an error, while the other does not. We have introduced the top element $\top$ to represent this case. For the monitor state, a dataflow analysis will be performed to approximate the public view after a secret-dependent branching construct. For this analysis, a bottom element (denoting unreachability) comes in handy to obtain a complete lattice (see Definition 3).

Definition 3 (Complete lattice of monitor states). *The complete lattice of monitor states is* $\mathbb{D} = (Var \to \mathfrak{B}_{\Sigma,\{\#,\square\}}) \cup \{\lightning, \top, \bot\}$*. For any* $D_1, D_2 \in \mathbb{D}$ *the relation* $D_1 \sqsubseteq D_2$ *holds if one of the following is true:*

- $D_1 = \bot$
- $D_2 = \top$
- $D_1 = \lightning$ *and* $D_2 = \lightning$
- *If* $D_1, D_2 \notin \{\lightning, \top, \bot\}$ *then for all variables* x *it holds that* $D_1(x) \sqsubseteq D_2(x)$ *according to Definition 2.*

The idea of the monitored execution is to execute the state transformations on the real state and the monitor state in parallel. In order to do so we need to specify the semantics of expressions on the monitor states. Considering tree expressions, we obtain the monitor semantics if we exchange σ with D in Figure 3 with the extension that $[\![x/1]\!]D = [\![x/2]\!]D = \top$ if $D(x) = \square$. This is inevitable for the following reason. In place of $\square$ in the monitor state, the real state may have any binary tree and thus, in particular, may equal to #. Therefore, depending on the secret, the result of the expression on the real state may be $\lightning$ or not. Therefore, the monitor state must be switched to $\top$.

As it is shown in Figure 6, a boolean expression is a transformation on the monitor state just as tree expressions are. Basically, if the boolean expression is true on the monitor state then it is the identity function, otherwise the result is $\bot$. The only exception is in the positive case ((1) and (2)) when the content of the variable x is transformed to the greatest tree not being $\square$, for which the boolean expression holds.

The semantics of the monitored execution is defined in the form of relations $cfg \to_\gamma cfg'$ between configurations of the form $\langle p, n, (\sigma, D)\rangle$ where p is a program to be executed and σ is the state of the execution. The member D is the public view of the state σ, which we call the monitor state, σ is called the real state.

$$[\![\texttt{top(}x\texttt{)=}\alpha]\!]D = \begin{cases} D & \text{if } D(x)(\varepsilon) = \alpha \\ D[x \mapsto \alpha(\square,\square)] & \text{if } D(x) = \square \text{ and } \alpha \neq \texttt{\#} \quad (1) \\ D[x \mapsto \texttt{\#}] & \text{if } D(x) = \square \text{ and } \alpha = \texttt{\#} \quad (2) \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\neg\texttt{top(}x\texttt{)=}\alpha]\!]D = \begin{cases} D & \text{if } D(x)(\varepsilon) \neq \alpha \text{ or } D(x) = \square \\ \bot & \text{otherwise} \end{cases}$$

Fig. 6. Semantics of boolean expressions on the monitor state

The member n is a natural number influencing the precision of the monitor when computing the public effect of branching constructs. Larger values correspond to enhanced precision and longer computation time. In the initial configuration $\langle p, n, (\sigma, D)\rangle$ it holds that $\sigma \sqsubseteq D$ and in case σ is not the error state, there are no nodes labeled $\square$ in the contents of its variables. The intuitive meaning of the relation $\sqsubseteq$ between the real state and the monitor state is that they agree on public values, and this is the property our monitor guarantees along the run.

As long as the monitored semantics does not execute a branching construct whose boolean expression depends on the secret, the monitored execution succeeds similarly to the original execution semantics shown in Figure 4. Assignments are executed in parallel on σ and on D according to the semantics in Figure 3 with the extension that $[\![x\texttt{/1}]\!]D = [\![x\texttt{/2}]\!]D = \top$ if $D(x) = \square$. The truth values of boolean expressions are determined based on the monitor state. If $[\![b]\!]D = \bot$, then we consider $\neg b$ to be true. In case $[\![b]\!]D \neq \bot$ and $[\![\neg b]\!]D \neq \bot$ simultaneously, we execute a branching construct, whose condition depends on the secret. In this case the result of the branching command on the real state is computed using the original semantics of Figure 4, the resulting monitor state is computed using a generalized constant propagation algorithm. This is the point, where the parameter n is used. Assume that the command c is a branching construct whose condition depends on the secret in configuration $\langle c\texttt{;}p, n, (\sigma, D)\rangle$. In this case we apply the generalized constant propagation on the command $c(n)$, which we construct based on c by replacing all occurrences of the command `while` b `do` p in the program text of c by `while(`n`)` b `do` p.

The *generalized constant propagation* algorithm is defined in Figure 7, which is basically the rule-based formalization of a syntax directed fixed point computation algorithm on the program text as it is presented in [7]. The lattice is the set of possible monitor states according to Definition 3.

The rules defining the functionality of assignment (MA), sequential execution of commands (MS), and the propagation of the states $\top$, $\bot$ and $\lightning$ (MCE) are very similar to rules A, S and E of the original semantics. The only difference is at rule MCE, which propagates the states $\bot$ and $\top$ unmodified as well.

The rule MI is responsible for computing the monitor state transformation belonging to an `if` command. It evaluates both branches with initial states $[\![b]\!]D$ and $[\![\neg b]\!]D$ and then joins the results.

The rules MWT, MWF, MWH and MWX are used to compute the public effect of iterations. If the parameter n is zero, or the condition is secret-dependent,

$$\text{MCE: } \frac{D \in \{\top, \bot, \lightning\}}{\langle c, D\rangle \to_\mu D} \quad \text{MA: } \frac{D \notin \{\top, \bot, \lightning\}}{\langle x\texttt{<-}e, D\rangle \to_\mu [\![x\texttt{<-}e]\!]D} \quad \text{MS: } \frac{\langle c, D\rangle \to_\mu^* D'}{\langle c\texttt{;}p, D\rangle \to_\mu \langle p, D'\rangle}$$

$$\text{MI: } \frac{\begin{array}{c}\langle p_t, [\![b]\!]D\rangle \to_\mu^* D_1 \\ \langle p_f, [\![\neg b]\!]D\rangle \to_\mu^* D_2 \\ D \notin \{\top, \bot, \lightning\} \quad D' = D_1 \sqcup D_2\end{array}}{\langle \texttt{if } b \texttt{ then } p_t \texttt{ else } p_f, D\rangle \to_\mu D'} \quad \text{MWF: } \frac{\begin{array}{c}D \notin \{\top, \bot, \lightning\} \\ [\![b]\!]D = \bot\end{array}}{\langle \texttt{while(}n\texttt{) } b \texttt{ do } p_t, D\rangle \to_\mu D}$$

$$\text{MWT: } \frac{D \notin \{\top, \bot, \lightning\} \quad [\![\neg b]\!]D = \bot \quad n > 0 \quad \langle p_t, [\![b]\!]D\rangle \to_\mu^* D'}{\langle \texttt{while(}n\texttt{) } b \texttt{ do } p_t, D\rangle \to_\mu \langle \texttt{while(}n-1\texttt{) } b \texttt{ do } p_t, D'\rangle}$$

$$\text{MWH: } \frac{\begin{array}{c}([\![\neg b]\!]D \neq \bot \wedge [\![b]\!]D \neq \bot) \vee n \leq 0 \\ D \notin \{\top, \bot, \lightning\} \quad \langle p_t, [\![b]\!]D\rangle \to_\mu^* D_1 \quad D' = D_1 \sqcup D \quad D' \not\sqsubseteq D\end{array}}{\langle \texttt{while(}n\texttt{) } b \texttt{ do } p_t, D\rangle \to_\mu \langle \texttt{while(}n-1\texttt{) } b \texttt{ do } p_t, D'\rangle}$$

$$\text{MWX: } \frac{\begin{array}{c}([\![b]\!]D \neq \bot \wedge [\![b]\!]D \neq \bot) \vee n \leq 0 \\ D \notin \{\top, \bot, \lightning\} \quad \langle p_t, [\![b]\!]D\rangle \to_\mu^* D_1 \quad D' = D_1 \sqcup D \quad D' \sqsubseteq D\end{array}}{\langle \texttt{while(}n\texttt{) } b \texttt{ do } p_t, D\rangle \to_\mu D}$$

Fig. 7. Generalized constant propagation

a fixed point is computed by rules MWH and MWX. If, however, the condition is independent of the secret, and n is greater then zero, the monitor executes the body of the loop iteratively by applying rules MWT and MWF. In the same time this might not terminate. So the purpose of n is to allow the user to specify how many times the monitor should apply the rule MWT before switching to the fixed point computation. In particular, setting the parameter n to zero in the initial configuration of the monitored execution $\langle p, n, (\sigma, D)\rangle$ amounts to choosing to omit the application of rules MWT and MWF, and use only the fixed point computation offered by rules MWH and MWX. In the same time this might result in an unnecessarily inaccurate monitor state.

Because the complete lattice of Definition 3 has the ascending chain condition, as Theorem 1 states below, the fixed point computation always terminates.

Theorem 1. *If there is σ' so that $\langle p, \sigma\rangle \to_\rho^* \sigma'$ and $\sigma \sqsubseteq D$, then there is a D' so that $\langle p, n, (\sigma, D)\rangle \to_\gamma^* (\sigma', D')$.*

Proof. The idea behind the proof is the following: If there is no branching construct executed having condition depending on the secret along the monitored execution, then the state transitions are carried out simultaneously on the real state and on the monitor state. In the case of a branching construct having secret-dependent condition, the public effect is computed by the algorithm in Figure 7. The only rule, which could be applied an unbounded number of times is MWH, but because the lattice of Definition 3 has the ascending chain condition, the fixed point computation terminates. For the detailed proof please refer to [19].

5 Guarantees

In this section we formally discuss the guarantees provided by the runtime monitor.

Similarly to other language-based information flow controlling solutions [4,12, 13,15,23,26,27,30], our approach enforces a variant of Termination Insensitive Noninterference [6] tailored for our computational model. Accordingly, we do not consider covert channels like the timing channel, the heat channel, or the memory consumption channel, or any other channel that could result from the properties of a specific implementation or runtime environment.

Definition 4. *Program p satisfies Termination Insensitive Noninterference relative to the initial and final public views D and D' if and only if for all $\sigma_1, \sigma_2 \sqsubseteq D$ it is true that if*

- $\langle p, \sigma_1 \rangle \rightarrow_\rho^* \sigma_1'$ *and*
- $\langle p, \sigma_2 \rangle \rightarrow_\rho^* \sigma_2'$

then $\sigma_1' \sqsubseteq D'$ and $\sigma_2' \sqsubseteq D'$. In this case we say that D' is an appropriate final public view belonging to the program p and the initial public view D.

The monitored execution $\langle p, n, (\sigma, D) \rangle \rightarrow_\gamma^* (\sigma', D')$ computes a pair (σ', D') based on (σ, D), where D' is an appropriate final public view belonging to p and D. We may consider the public view D as an indistinguishability relation between any initial states σ^*, for which it holds that $\sigma^* \sqsubseteq D$. The meaning of the resulting monitor state D' is that by observing it, we do not gain information on which σ^* was the initial state. Accordingly, we can communicate parts or the entire final public view D' to principals having low security clearance (L), and we can consider nodes labeled $\square$ as a default value that secret pieces of information have been replaced with. If $D' = \top$ then the observers of the public do not gain more information than the fact that the execution of the program terminated. Still, principals with high security clearance (H) may observe the resulting real state σ', and use the computed values.

The following theorem assures us that our monitor indeed computes an appropriate final public view belonging to the program and the initial public view:

Theorem 2. *If there are two initial states σ_1 and σ_2 so that $\sigma_1, \sigma_2 \sqsubseteq D$, then if*

- $\langle p, n, (\sigma_1, D) \rangle \rightarrow_\gamma^* (\sigma_1', D_1')$ *and*
- $\langle p, n, (\sigma_2, D) \rangle \rightarrow_\gamma^* (\sigma_2', D_2')$

then $D_1' = D_2' = D'$ and $\sigma_1' \sqsubseteq D'$ and $\sigma_2' \sqsubseteq D'$.

Proof. For the proof please refer to [19].

6 Related Work and Conclusion

This paper is related to two research areas. One is language based information flow security as we have discussed in Section 3, the other is formalization and verification of Web service compositions and business workflows. Much effort has been invested to find adequate formalisms to describe the functionality of service orchestrations and choreography, in particular, the BPEL [5] language, in order to enable formal rigor for development and to allow verification. The majority of the publications in this topic can be sorted into two groups. One [1,2,16,24] applies formalisms based on Petri-nets to model workflows, the other [9,14,20,25,32] prefers algebraic calculi like the Π-calculus [22] as the basis for investigations. The authors of [3] and [33] present security related results using Petri-net based formalisms. A common property of these approaches is that they mostly focus on the control flow of orchestrations, sometimes with emphasis on error handling, whereas data values undergo severe abstractions. Data are either considered as atomic values, or completely disregarded by handling branching decisions as nondeterminism.

At the same time practical business processes mostly use XML documents for data representation, which are naturally semi-structured and unbounded in size. In the case of these processes, security is especially important, because of their distributed nature. For this reason we showed for a minimalistic, yet Turing-complete language, how to provide security guarantees without carrying out major abstractions on the data.

As a result we have presented an approach to enforce information flow control in tree manipulating processes. Since practical information flow policies refer to the structure of data, approaches where security specifications are bound to variables such as [4,8,10,13,15,23,30] may not suffice. In their paper [26], Russo et al. aim at a similar goal like us. In their formal model of JavaScript, they consider one DOM tree representing the data in a Web browser. However, their formalism is still quite different to ours. Their computational model operates on a single unranked tree using a pointer on one specific working node, and supports operations like insertion, modification and removal. Since their monitor maintains the security levels of the positions of the DOM tree during the run, it can be considered as a generalization of the idea of binding secrecy levels to variables towards trees. Our monitor, on the other hand, maintains the concrete values of public nodes making it possible to take the semantics of branches into account where the conditional depends on the secret and compute the public effect by means of a value-based comparison of the resulting states.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: yet another workflow language. Information Systems 30(4), 245–275 (2005)
2. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press (2002)

3. Accorsi, R., Wonnemann, C.: Static information flow analysis of workflow models. In: Abramowicz, W., Alt, R., Fähnrich, K.P., Franczyk, B., Maciaszek, L.A. (eds.) ISSS/BPSC. LNI, vol. 177, pp. 194–205. GI (2010)
4. Myers, A.C., Zheng, L., Zdancewic, S., Chong, S., Nystrom, N.: Jif:java + information flow (July 2001-2011), Software release. Located at, `http://www.cs.cornell.edu/jif`
5. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., Koenig, D., Marin, M., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., Yiu, A.: Web services business process execution language version 2.0 (OASIS standard). WS-BPEL TC OASIS (2007), `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
6. Askarov, A., Hunt, S., Sabelfeld, A., Sands, D.: Termination-Insensitive Noninterference Leaks more than Just a Bit. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 333–348. Springer, Heidelberg (2008)
7. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. CoRR abs/cs/0701193 (2007)
8. Broberg, N., Sands, D.: Paralocks – role-based information flow control and beyond. In: POPL 2010: Proceedings of the 37th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (2010)
9. Bruni, R., Melgratti, H.C., Montanari, U.: Theoretical foundations for compensations in flow composition languages. In: Palsberg, J., Abadi, M. (eds.) POPL, pp. 209–220. ACM (2005)
10. Chong, S., Vikram, K., Myers, A.C.: SIF: enforcing confidentiality and integrity in web applications. In: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium, pp. 1:1–1:16. USENIX Association, Berkeley (2007)
11. Denning, D.E.: A lattice model of secure information flow. Commun. ACM 19(5), 236–243 (1976)
12. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. Commun. ACM 20(7), 504–513 (1977)
13. Guernic, G.L.: Automaton-based confidentiality monitoring of concurrent programs. In: CSF, pp. 218–232. IEEE Computer Society (2007)
14. Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G.: Sock: A Calculus for Service Oriented Computing. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 327–338. Springer, Heidelberg (2006)
15. Hammer, C., Snelting, G.: Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. International Journal of Information Security 8(6), 399–422 (2009)
16. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri Nets. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 220–235. Springer, Heidelberg (2005)
17. Horwitz, S., Prins, J., Reps, T.: On the adequacy of program dependence graphs for representing programs. In: POPL 1988: Proceedings of the 15th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages, pp. 146–157. ACM, New York (1988)
18. Köpf, B., Mantel, H.: Transformational typing and unification for automatically correcting insecure programs. Int. J. Inf. Sec. 6(2-3), 107–131 (2007)
19. Kovács, M., Seidl, H.: Runtime enforcement of information flow security in tree manipulating processes (proofs). Tech. rep., Technische Universität München, Institut für Informatik (2011)

20. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. J. Log. Algebr. Program. 70(1), 96–118 (2007)
21. Mantel, H., Sands, D.: Controlled Declassification Based on Intransitive Noninterference. In: Chin, W.N. (ed.) APLAS 2004. LNCS, vol. 3302, pp. 129–145. Springer, Heidelberg (2004)
22. Milner, R.: Communicating and Mobile Systems: the Π-calculus. Cambridge University Press (1999)
23. Myers, A.C.: JFlow: Practical mostly-static information flow control. In: Proc. 26th ACM Symp. on Principles of Programming Languages (POPL), pp. 228–241 (1999)
24. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: WofBPEL: A Tool for Automated Analysis of BPEL Processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 484–489. Springer, Heidelberg (2005)
25. Rouached, M., Godart, C.: Requirements-driven verification of WSBPEL processes. In: IEEE International Conference on Web Services, ICWS 2007, pp. 354–363 (July 2007)
26. Russo, A., Sabelfeld, A., Chudnov, A.: Tracking Information Flow in Dynamic Tree Structures. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 86–103. Springer, Heidelberg (2009)
27. Sabelfeld, A., Myers, A.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
28. Sabelfeld, A., Russo, A.: From Dynamic to Static and Back: Riding the Roller Coaster of Information-flow Control Research. In: Pnueli, A., Virbitskaite, I., Voronkov, A. (eds.) PSI 2009. LNCS, vol. 5947, pp. 352–365. Springer, Heidelberg (2010)
29. Seidl, H., Wilhelm, R., Hack, S.: Compiler Design: Analysis and Transformation. Springer, Heidelberg (2011)
30. Venkatakrishnan, V.N., Xu, W., DuVarney, D.C., Sekar, R.: Provably Correct Runtime Enforcement of Non-Interference Properties. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 332–351. Springer, Heidelberg (2006)
31. Volpano, D.M., Irvine, C.E., Smith, G.: A sound type system for secure ow analysis. Journal of Computer Security 4(2/3), 167–188 (1996)
32. Wirsing, M., Clark, A., Gilmore, S., Hölzl, M., Knapp, A., Koch, N., Schroeder, A.: Semantic-Based Development of Service-Oriented Systems. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 24–45. Springer, Heidelberg (2006)
33. Wolter, C., Miseldine, P., Meinel, C.: Verification of Business Process Entailment Constraints using SPIN. In: Massacci, F., Redwine Jr., S.T., Zannone, N. (eds.) ESSoS 2009. LNCS, vol. 5429, pp. 1–15. Springer, Heidelberg (2009)

Formalisation and Implementation of the XACML Access Control Mechanism*

Massimiliano Masi[1,2], Rosario Pugliese[2], and Francesco Tiezzi[3]

[1] Tiani "Spirit" GmbH, Guglgasse, 6 - 1110 Vienna, Austria
[2] Università degli Studi di Firenze, Viale Morgagni, 65 - 50134 Firenze, Italy
[3] IMT Advanced Studies Lucca, Piazza S. Ponziano, 6 - 55100, Lucca, Italy

Abstract. We propose a formal account of XACML, an OASIS standard adhering to the Policy Based Access Control model for the specification and enforcement of access control policies. To clarify all ambiguous and intricate aspects of XACML, we provide it with a more manageable alternative syntax and with a solid semantic ground. This lays the basis for developing tools and methodologies which allow software engineers to easily and precisely regulate access to resources using policies. To demonstrate feasibility and effectiveness of our approach, we provide a software tool, supporting the specification and evaluation of policies and access requests, whose implementation fully relies on our formal development.

Keywords: PBAC, XACML, formal semantics, CASE tools.

1 Introduction

Nowadays, web services are increasingly used by enterprises and organizations to expose their data to business partners. In this context, resources and services are spread among different administrative domains, thus controlling accesses to them has become a crucial issue. Access control mechanisms are currently used to mitigate the risks of unauthorized access to resources and systems, which could jeopardise the secrecy of sensitive data and cause loss of competitive advantages. These mechanisms may take several forms, use different technologies and involve varying degrees of complexity. Anyway, they are implementations of one of the several access control models proposed in the literature (see, e.g., [1,2]).

We focus on the Policy Based Access Control (PBAC) model [2], that is by now the de-facto standard model for enforcing access control policies in service-oriented architectures. In this model, a resource is governed by a document that exactly specifies what subject credentials and requirements must be fulfilled in order to obtain access. A widely used implementation of PBAC is given by the eXtensible Access Control Markup Language (XACML) [3], an OASIS standard now at version 2.0[1]. It defines a language for the definition of policies and access requests, and a workflow to achieve policy enforcement. XACML is currently

* This work has been partially sponsored by the EU project ASCENS (257414).
[1] We will refer from now on to [3] as *the standard*.

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 60–74, 2012.

used as a basis for enforcing access control in many large scale projects (see, e.g., [4,5]) and standards (see, e.g., [6,7]).

However, designing XACML access control policies is a difficult and error-prone task. The language has an XML syntax which makes writing XACML policies awkward by using common editors. To make the definition of XACML policies easier also for those end users that are not accustomed with the complexity of the overall policy language, many companies have equipped their products with ad-hoc policy editors (e.g. [8,9]). Such editors are certainly suitable to develop simple and repetitive policies, but might turn out to be cumbersome and ineffective when dealing with complex policies as indeed they tend to hide all the possibilities available in the policy language. Most of all, XACML comes without a formal semantics. The standard is written in prose and contains quite a number of loose points that may give rise to different interpretations and lead to different implementation choices. Some of these loose points are due to an extensive use of the keyword "SHOULD", as per the IETF rfc2119 [10], to indicate recommended requirements that can be for some reason ignored. This leaves the difficult task of understanding the full implications of the various choices to the implementers. Of course, this has to be avoided, since otherwise the portability of XACML policies across different platforms would be considerably undermined.

In this paper we introduce a formal semantics of XACML 2.0[2] that clarifies all ambiguous and intricate aspects of the standard. To hide the complexity introduced by XML, we propose an alternative syntax. This way, we get a tiny language with solid mathematical foundations that lays the basis for developing tools and methodologies that can be easily used by software engineers to precisely define access controls policies on resources. To demonstrate feasibility and effectiveness of our approach, by relying on the formal semantics, we have implemented our language using Java. We have thus obtained a software tool that supports the specification and evaluation of policies and access requests.

Related work. As a result of the widespread use of XACML in (web) service-oriented systems and international projects, many attempts of formalisation have been made. A largely followed approach is based on 'transformational' semantics, where XACML policies are translated into terms of some formalism. For example, [11] uses description logic expressions as target formalism, [12] exploits the process algebra CSP [13], and [14] the model-oriented specification language VDM++ [15]. The main advantage of this approach is the possibility of analysing policies by means of off-the-shelf reasoning tools that may be already available for the considered formalisms. From the semantics point of view, this approach provides some alternative high-level representations of policies, which in their turn have their own semantics. This makes it more difficult to understand the formal meaning of policies with respect to our formal semantics, which directly associates mathematical objects (i.e. 4-tuples of request sets) to policies. These concepts are easier and more understandable than terms, like e.g. description

[2] At the time of writing, the new version XACML 3.0 is under first review and, hence, is continuously changing. We suppose that a full adoption of this new version in production projects will take quite some time.

logic expressions, resulting from automatic translations, also because such translations unavoidably produce terms more complex than necessary. Therefore, our semantics can be conveniently exploited by software engineers to drive XACML implementations. At the same time, its mathematical foundations enable the development of reasoning tools (as we briefly discuss in Section 6).

A similar approach is proposed in [16], where the policies are first specified by means of the description language RW [17], then are analysed through a model checking technique, and finally are translated in XACML. Advantages and disadvantages with respect to our approach are as before.

Other formalisation approaches, more similar to ours, defines the semantics of XACML policies in a more direct way. For example, [18] proposes a semantics based on (multi-terminal) binary decision diagrams, which permit efficiently carrying out the proposed analysis techniques (i.e. property verification and change-impact analysis), but are not suitable as an implementation guide. Instead, [19] formalises a subset of XACML, called Core XACML. The semantics is given through an inductively defined policy evaluation function. Differently from our approach, each policy is evaluated only w.r.t. a single request and, most of all, Core XACML ignores some important XACML features, such as rule conditions, matching functions, some combining algorithms, and the indeterminate value.

There are by now many XACML implementations (see e.g. [20]). In particular, SUN XACML [21] and HERAS^{AF} [22], that are widely used in software in production, implement a Policy Decision Point (PDP) and a library for the development of Policy Enforcement Point (PEP)s. Differently from our implementation, they parse policies in XML format deployed in the policy repository. Moreover, they evaluate each request by visiting parts of the generated DOM tree, while we evaluate the requests by executing Java classes implementing the semantics representations of the policies. XEngine [23] is another notable implementation. It aims at highly efficient request processing, achieved by converting XACML policies into numerical representations. Instead, our main goal is the development of an XACML implementation driven by a formal semantics. Another implementation of an access control mechanism is PERMIS [24], a modular infrastructure specifically devised for Grid systems and integrated in modern toolkits (like, e.g., [25,26]). However, PERMIS relies on an ad-hoc, non-standard policy language which is less expressive than XACML [27].

To sum up, differently from related works, our formalisation has a twofold aim: it serves as a guide for implementers and, at the same time, paves the way for the development of analysis tools.

Summary of the rest of the paper. In Section 2, we give a glimpse of the XACML standard by describing the underlying access control model and the main features of the policy language. In Section 3, we introduce an alternative syntax for XACML, which we then use in Section 4 as the basis to define the formal semantics. We illustrate our approach through an example from an healthcare project. In Section 5, we describe our Java-based implementation of the formal semantics. Finally, in Section 6, we touch upon directions for future work.

2 The XACML Standard

In the access control model underlying XACML, each resource can be paired with one or more policies, namely XML documents expressing the capabilities that a requestor needs to have for accessing the resource. Specifically, policies and policy sets are retrieved from a Policy Administration Point (PAP) by a PDP, which is on duty to decide whether to give access to resources or not. The policies and policy sets retrieved by the PDP represent the complete policy for the specified resources.

A request to access a resource is created by a PEP, which reuses claims within the service invocation made by an *access requester*. PEPs can have many different forms, e.g. they may be part of a remote-access gateway, a Web server, an email user-agent, etc. Thus, we cannot expect that in an enterprise all PEPs issue access requests to a PDP directly in a common format. Therefore, the requests and responses handled by the PDP must be converted in a canonical form, i.e. the so-called XACML *context*. The obvious benefit of this approach is that policies may be written and analyzed independently of the specific environment in which they have to be enforced.

The authorization decision is made by the PDP by checking the *matching* between values of the request and values from the retrieved policies. The decision taken by the PDP can be one among permit, deny, not-applicable and indeterminate: the meaning of the first two values is obvious, while the third means that the PDP does not have any policy that applies to the request and the fourth means that the PDP is unable to evaluate the access request (reasons for such inability include, e.g., missing attributes, network errors, evaluation errors).

Let us now consider the languages for expressing policies and requests provided by the standard. The basic element of the policy language is Policy. A Policy is composed of a Target, which identifies the set of capabilities that the requestor must expose, and some Rules. Every Rule contains the facts for the access control decision and has an Effect, which can be either Permit or Deny. A Policy also specifies a combining algorithm that defines what is the final decision for a request when there are (permit/deny) conflicts in the rule decisions.

A Target is composed of four sub-elements: Subjects, Actions, Resources, and Environments. Each category is composed of a set of target elements, each of which contains an attribute identifier, a value and a matching function. Such information is used to check whether the policy is applicable to a given request. Specifically, the matching function retrieves a value from the designed attribute in the request and matches it with the values specified in the target element, according to the function's semantics. If, for all four categories, at least a matching of a target element succeeds, then the policy is applicable to the request.

Besides the Effect, a Rule may specify a Target and some Conditions, i.e. a set of standardly-defined functions that operate on values coming from the request. The Effect is propagated to the upper level policy if the Target of the rule matches and if the Conditions are satisfied.

Policies can be combined together into a PolicySet, which specifies an algorithm that defines the policy set decision in case the contained policies cause

permit/deny conflicts. A PolicySet also contains a Target, which is checked for matching with the access request before the targets of the included policies are.

A Policy/PolicySet can also contain a set of Obligations indicating the actions that the PEP shall enforce after receiving the response. However, since such actions do not play any role in the evaluation procedure, Obligations are not considered in this paper.

An XACML Request, instead, is the request in a canonical form (created by the PEP or the context handler) made of attribute/value pairs. The elements specifying such pairs are grouped according to the same four categories used for the policies, i.e. Subject, Action, Environment and Resource.

3 An Alternative Syntax of XACML

The XACML standard, as explained in the previous section, defines an XML-based language that permits both writing policies [3, Section 5] and representing contexts (i.e. access requests and responses) [3, Section 6] in a way independent of the specific formats used by PEPs. However, the XML syntax of this language, on the one hand, can make the task of writing policies difficult and error-prone, and, on the other hand, is not adequate for formally defining the semantics of the language and reasoning on it. Therefore, in this section, we provide an alternative syntax of the XACML policy language through a BNF-like grammar (a similar grammar for context representation can be found in [28]).

Our alternative syntax of the XACML policy language is reported in Table 1. As usual, square brackets are used to indicate optional items (that is, everything that is set within the square brackets may be present just once, or not at all).

The manipulable values, ranged over by value, can have simple types (e.g. boolean, string, integer) or complex types (i.e. the values are XML elements that may contain other elements and/or attributes). For the sake of simplicity, we present an untyped version of the language, because the treatment of types would be standard and, anyway, their addition is not relevant for our studies.

To base an authorization decision on some characteristics of the request, like e.g. the subject's identity or the resource's identifier, XACML provides facilities to identify specific values (called *attribute* values) contained in the request context. This approach is supported by means of *attribute designators* and *attribute selectors*. The former ones are pointers to specific attributes of targets (e.g. subjects or resources) in the request context, while the latter ones provide a more general retrieval mechanism based on XPath [29] expressions over the request context. For the sake of presentation, in our XACML's syntax, we represent both designators and selectors by means of *(structured) names*, ranged over by name. For example, the following designator (drawn from [28])

```
<SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
                            DataType="http://www.w3.org/2001/XMLSchema#string" />
```

is represented by the name subject.role.

To permit specifying conditions, the language is also equipped with *expressions*, ranged over by expression, which are defined by functions that operate

Table 1. XACML policies syntax

$PDPpolicies$	$::= \{Palg\,;Policies\}$	(Retrieved policies)
$Palg$	::= only-one-applicable $\mid$ $Ralg$	(Policy-combining alg.)
$Ralg$	::= deny-overrides $\mid$ permit-overrides $\mid$ first-applicable $\mid$ ordered-deny-overrides $\mid$ ordered-permit-overrides	(Rule-combining alg.)
$Policies$	::=	(Policies)
	$\{Palg\,;$target :$\{\,[\,Targets\,]\,\}\,;Policies\}$	(policy set)
	$\mid \langle Ralg\,;$target :$\{\,[\,Targets\,]\,\}\,;$rules :$\{Rules\}\rangle$	(policy)
	$\mid Policies \quad Policies$	
$Targets$	::= $MatchId$(value,name)	(Targets)
	$\mid Targets \vee Targets$	
	$\mid Targets \wedge Targets \quad \mid \quad Targets \sqcap Targets$	
$MatchId$	::= string-equal $\mid$ integer-equal	(Match functions)
	$\mid$ string-regexp-match	
	$\mid$ integer-greater-than $\mid$ ...	
$Rules$	::= ($Effect$ [;target :$\{Targets\}$] [;condition :{expression}])	(Rules)
	$\mid Rules \quad Rules$	
$Effect$	::= permit $\mid$ deny	(Effects)

on values and names. The complete list of functions provided by XACML is reported in [3, Appendix A.3], while the examples shown in the rest of the paper will exploit the syntax of expressions (reported in [28]) implemented by the tool described in Section 5.

For efficiency of evaluation and ease of management, the overall security policy in force across an enterprise is expressed as multiple independent components. Then, the top-level term $\{Palg\,;Policies\}$ of the XACML policy syntax is a simplified form of policy set (i.e. without target). Given a request, the PDP evaluates the policies in $Policies$ (possibly retrieved from a repository or a PAP) as if they are organised as a single policy set, according to a specified policy-combining algorithm $Palg$. The algorithms provided by XACML for combining the values resulting from policies evaluation – which can be permit, deny, not-applicable and indeterminate – are the following (we refer to [28] for a more precise account):

- deny-overrides: if any policy in the considered set evaluates to deny, then the result of the policy combination is deny;
- permit-overrides: it is similar to the previous algorithm, but this time permit takes precedence over the other results;
- first-applicable: the combined result is that resulting from the evaluation the first policy whose target is applicable to the request;

- ordered-deny-overrides/ordered-permit-overrides: like deny-overrides/permit-overrides, but policies are evaluated in the same order as they occur;
- only-one-applicable: it only applies to policies/policy sets and ensures that one and only one policy is applicable by virtue of its target.

The policies that can be evaluated by the PDP, and hence aggregated by a policy set, can be simple policies of the form $\langle Ralg\,;\mathsf{target}:\{\,[\,Targets\,]\,\};\mathsf{rules}:\{Rules\}\rangle$ or, recursively, policy sets of the form $\{Palg\,;\mathsf{target}:\{\,[\,Targets\,]\,\};Policies\}$. Both polices and policy sets specify the algorithm for combining the results of the evaluation of the contained elements and a target to which the policy/policy set applies. The algorithms for simple policies are the same as those for policy sets (but for only-one-applicable) and behave similarly.

A *target* permits identifying the set of access requests that a rule, a policy or a policy set is intended to evaluate. Specifically, a target specifies the set of *subjects*, *resources*, *actions* and *environments* to which the corresponding rule/policy/policy set applies. In the original XML-based syntax of XACML, the target element may contain four elements, one for each of the above categories. However, the evaluation of these separate blocks of information shall be performed in the same way. In fact, in the XACML specification document, the evaluations of subjects, resources, actions and environments are defined by the same 'match table' [3, Section 7.6] and, also, the set of designators for each category is not fixed in advance. Therefore, to obtain a more compact notation, we have decided to represent a target as an expression built from *match elements*, i.e. terms of the form $MatchId(\mathsf{value},\mathsf{name})$, by exploiting an operator for logical disjunction, $\vee$, and two operators for logical conjunction, $\wedge$ and $\sqcap$. Each match element spells out a specific value that the subject/resource/action/environment in the decision request (identified by a name) must match, according to a given matching function. Anyway, this target representation does not lead to a loss of information, because names can be structured and hence, as shown before in the designator example, can include the corresponding category. In a match element, *MatchId* specifies the (boolean) matching function to be used to compare the given literal value with the value of the attribute identified by the given name. XACML supports a wide range of (standard) matching functions (we refer to [3, Appendix A.3] for a complete account and to [28] for the list of functions supported by the tool described in Section 5). Notably, if the target of a policy (resp. policy set) is empty, the policy (resp. policy set) applies to any request context. Instead, if the target of a rule is absent, the rule inherits the target of its enclosing policy.

The three logical operators used for expressing targets are defined over the set $\{\mathsf{match},\mathsf{no\text{-}match},\mathsf{indeterminate}\}$. Basically, they behave as standard conjunction and disjunction operators over $\{\mathsf{match},\mathsf{no\text{-}match}\}$ (where match and no-match are dealt with as true and false, respectively) and the behaviours of the two conjunction operators $\wedge$ and $\sqcap$ only differ for the treatment of the value indeterminate. The decreasing order of precedence among them is as follows: $\wedge$, $\vee$ and $\sqcap$. A disciplined use of structured names and these logical operators permits properly expressing XACML targets: a target must be a term of the form

$Subjects \sqcap Resources \sqcap Actions \sqcap Environments$, where each subterm, say $Subjects$, must have the form $Subject_1 \vee Subject_2 \vee \ldots \vee Subject_n$ and, finally, each $Subject_i$ must have the form $MatchId_1(\mathsf{value}_1,\mathsf{name}_1) \wedge \ldots \wedge MatchId_m(\mathsf{value}_m,\mathsf{name}_m)$. We believe our approach has many advantages like, e.g., a more compact syntax and a more intuitive and clearer semantics.

A single policy contains a (non-empty) set of rules of the form (*Effect* [; target :{ $Targets$ }][; condition :{expression}]), each specifying: 1. an *effect*, which indicates the rule-writer's intended consequence of a positive evaluation for the rule (the allowed values are permit and deny), 2. a *rule target*, which refines the applicability established by the target of the enclosing policy, and 3. a *condition*, which is a boolean expression that may further refine the applicability of the rule. Notably, in a rule, target and condition may be absent.

Regarding context requests, they are represented as terms of the form request :{ *Attributes* }, where *Attributes* consists of a set of (name,value) pairs. Such information indicate the subjects associated to the request, the resources for which the access is being requested, the action to be performed on the resources and the environmental properties. Again, to avoid dealing with separate blocks of information, we exploit structured names. As a matter of notation, we will use R_{all} to denote the set of all possible requests.

We conclude by showing the syntax of a policy[3], which expresses the *patient privacy consent* [30] for the EU Project epSOS [4]. In this project, each role (e.g. doctor, nurse, pharmacist) has *permissions* for performing a certain *coded action* [31] for a certain purpose (e.g. healthcare treatment, statistics, emergency).

```
⟨permit-overrides ;
   target :{ string-equal("medical doctor", subject.role)
                ∧ string-equal("TREATMENT", subject.purposeofuse)
             ⊓ string-equal("34133-9", resource.resource-id) } ;
   rules :{(permit ; target :{ string-equal("Read", action.action-id) } ;
            condition :{ string-subset(
                           string-bag("PRD-003","PRD-005","PRD-010","PRD-016"),
                           subject.permission) })
           (deny)  } ⟩
```

The policy specifies a subject and a resource in its target, according to which the policy applies to requests issued by a medical doctor with the purpose of accessing to a resource with a code identifier 34133-9[4] for an healthcare TREATMENT. If these capabilities are met, the rules enclosed in the policy are evaluated. The first rule has effect permit if the requestor aims at performing a Read action and has at least the permissions PRD-003, PRD-005, PRD-010 and PRD-016[5] for accessing the resource. The second rule has always effect Deny and is combined

[3] Due to lack of space, the corresponding XML description is relegated to [28].

[4] In the international code system LOINC [32], 34133-9 identifies a *patient summary*.

[5] These permissions are values from the Hl7 RBAC catalogue [31] grouped together by using a *bag*, i.e. an unordered collection that may contain duplicate values.

with the previous one in such a way that if the first rule evaluates to Permit then the policy permits the access to the resource, otherwise the access is denied.

4 XACML Formal Semantics

We present in this section a semantics of XACML policies that formalises the informal one provided by the the standard.

Our semantics is given in a denotational style, i.e. it is defined by a function $[\![\cdot]\!]_R$ that, given a policy/policy set (or a *PDPpolicies* term) and a set R of context requests (with $R \subseteq R_{all}$), returns a *decision* tuple of the form

$$(\,\mathsf{permit} : R_p\,;\ \mathsf{deny} : R_d\,;\ \mathsf{not\text{-}applicable} : R_n\,;\ \mathsf{indeterminate} : R_i\,)$$

where $R_p \cup R_d \cup R_n \cup R_i = R$. Intuitively, R is partitioned into four sets according to the results of the requests evaluation. Notably, R is a subset of the set R_{all} of all requests, thus it can contain e.g. all possible requests, only requests with a given structure or a single request. The definition of $[\![\cdot]\!]_R$ relies on an auxiliary function $(\!|\cdot|\!)_R$ that, given a target, returns a *matching* tuple of the form

$$(\,\mathsf{match} : R_m\,;\ \mathsf{no\text{-}match} : R_n\,;\ \mathsf{indeterminate} : R_i\,)$$

where $R_m \cup R_n \cup R_i = R$, i.e. R is partitioned according to the results of the target evaluation. We will use a projection operator $\cdot\downarrow_{\mathsf{v}}$ that, given a tuple, returns the set corresponding to the value v. Moreover, we will use r to denote a context request and, when convenient, we shall regard r as a set, writing e.g. $(\mathsf{name},\mathsf{value}) \in r$ to mean that $(\mathsf{name},\mathsf{value})$ is an attribute of the request r. As shown in [28], this representation of requests easily permits dealing with multivalued attributes and with the fact that attribute designators and selectors may select bags of values from a request context.

The semantics of a match element $\mathit{MatchId}(\mathsf{value},\mathsf{name})$ of a target is a matching tuple determined by comparing value with the values within the request attributes by means of the matching function $\mathit{MatchId}$.

$$\begin{aligned}
&(\!|\mathit{MatchId}(\mathsf{value},\mathsf{name})|\!)_R = \\
&\quad(\,\mathsf{match} : \{r \in R \mid \exists\,(\mathsf{name},\mathsf{value}') \in r : \mathit{MatchId}(\mathsf{value},\mathsf{value}') = \mathsf{true}\}; \\
&\quad\ \mathsf{no\text{-}match} : \{r \in R \mid \forall\,(\mathsf{name},\mathsf{value}') \in r : \\
&\qquad\qquad\qquad \mathit{MatchId}(\mathsf{value},\mathsf{value}') = \mathsf{false}\}; \\
&\quad\ \mathsf{indeterminate} : \{r \in R \mid \exists\,(\mathsf{name},\mathsf{value}') \in r : \\
&\qquad\qquad\qquad\qquad \mathit{MatchId}(\mathsf{value},\mathsf{value}') = \mathsf{indeterminate} \\
&\qquad\qquad \wedge\ \nexists\,(\mathsf{name},\mathsf{value}') \in r : \\
&\qquad\qquad\qquad\qquad \mathit{MatchId}(\mathsf{value},\mathsf{value}') = \mathsf{true}\}\,)
\end{aligned}$$

Notably, the use of the universal quantification in the definition of the no-match set implies that requests that do not contain attributes named name are inserted into the no-match set[6]. The definitions of the matching functions supported by

[6] We assume that the `MustBePresent` parameter of every selector/designator has always the default value `false`, which prescribes to return an empty bag when the specified attribute is absent from the request.

XACML are reported in [3, Appendix A.3]. For example, the function string-equal returns true if and only if both argument values are strings of equal length and are equal byte-by-byte according to the function integer-equal (defined by the IEEE standard [33]); otherwise the function string-equal returns false. The matching tuples returned by the evaluation of the match elements within a given target are then combined according to the semantics of the operators $\vee$, $\wedge$ and $\sqcap$, as e.g. in

$$
\begin{array}{l}
(\!|Targets_1 \vee Targets_2|\!)_R = \\
\quad (\ \mathsf{match} : (\!|Targets_1|\!)_R \downarrow_{\mathsf{match}} \ \cup (\!|Targets_2|\!)_R \downarrow_{\mathsf{match}}; \\
\quad\ \ \mathsf{no\text{-}match} : (\!|Targets_1|\!)_R \downarrow_{\mathsf{no\text{-}match}} \ \cap (\!|Targets_2|\!)_R \downarrow_{\mathsf{no\text{-}match}}; \\
\quad\ \ \mathsf{indeterminate} : ((\!|Targets_1|\!)_R \downarrow_{\mathsf{indeterminate}} \setminus (\!|Targets_2|\!)_R \downarrow_{\mathsf{match}}) \\
\qquad\qquad\qquad\qquad \cup ((\!|Targets_2|\!)_R \downarrow_{\mathsf{indeterminate}} \setminus (\!|Targets_1|\!)_R \downarrow_{\mathsf{match}})\ )
\end{array}
$$

The semantics of a rule with effect permit is defined as follows:

$$
\begin{array}{l}
[\![\ (\mathsf{permit}\,;\mathsf{target}:\{\ Targets\ \};\mathsf{condition}:\{\mathsf{expression}\})\ ]\!]_R = \\
\quad (\ \mathsf{permit} : \{r \in (\!|Targets|\!)_R \downarrow_{\mathsf{match}} \mid\ \mathsf{expression} \cdot r = \mathsf{true}\}\ ; \\
\quad\ \ \mathsf{deny} : \emptyset\ ; \\
\quad\ \ \mathsf{not\text{-}applicable} : \{r \in (\!|Targets|\!)_R \downarrow_{\mathsf{match}} \mid\ \mathsf{expression} \cdot r = \mathsf{false}\} \\
\qquad\qquad\qquad\qquad \cup\ (\!|Targets|\!)_R \downarrow_{\mathsf{no\text{-}match}}\ ; \\
\quad\ \ \mathsf{indeterminate} : \{r \in (\!|Targets|\!)_R \downarrow_{\mathsf{match}} \mid\ \mathsf{expression} \cdot r = \mathsf{indeterminate}\} \\
\qquad\qquad\qquad\qquad \cup\ (\!|Targets|\!)_R \downarrow_{\mathsf{indeterminate}}\ )
\end{array}
$$

where $\mathsf{expression} \cdot r$ denotes the evaluation of the expression expression w.r.t. the request r according to the function definitions reported in [3, Appendix A.3]. The semantics of a rule with effect deny is similar, except that in the decision tuple the permit and deny sets are swapped. Notably, in a rule, the target and the condition are optional; if one or both of them are absent, the semantics of the rule is determined by the above definitions where $\mathsf{expression} \cdot r$ is true for any r if the expression is omitted, and $(\!|Targets|\!)_R \downarrow_{\mathsf{match}} = R$, $(\!|Targets|\!)_R \downarrow_{\mathsf{no\text{-}match}} = \emptyset$ and $(\!|Targets|\!)_R \downarrow_{\mathsf{indeterminate}} = \emptyset$, if the target is omitted.

The semantics of a policy is defined as follows:

$$
\begin{array}{l}
[\![\ \langle Ralg\,;\mathsf{target}:\{\ Targets\ \}\,;\mathsf{rules}:\{Rules\}\rangle\ ]\!]_R = \\
\quad (\ \mathsf{permit} : Ralg(Rules)_{R_m} \downarrow_{\mathsf{permit}}\ ; \\
\quad\ \ \mathsf{deny} : Ralg(Rules)_{R_m} \downarrow_{\mathsf{deny}}\ ; \\
\quad\ \ \mathsf{not\text{-}applicable} : Ralg(Rules)_{R_m} \downarrow_{\mathsf{not\text{-}applicable}}\ \cup\ (\!|Targets|\!)_R \downarrow_{\mathsf{no\text{-}match}}\ ; \\
\quad\ \ \mathsf{indeterminate} : Ralg(Rules)_{R_m} \downarrow_{\mathsf{indeterminate}}\ \cup\ (\!|Targets|\!)_R \downarrow_{\mathsf{indeterminate}}\ )
\end{array}
$$

where R_m stands for $(\!|Targets|\!)_R \downarrow_{\mathsf{match}}$. Basically, the requests for which the policy's target does not match are evaluated as not-applicable, while those for which the policy's target is indeterminate are evaluated as indeterminate. The remaining requests, i.e. those for which the policy's target matches, are partitioned by applying the algorithm $Ralg$ specified by the policy to the policy's rules. Similarly to the evaluation of rules, if the policy's target is empty then the policy is

evaluated as above by letting $(\!|Targets|\!)_R \downarrow_{\mathsf{match}} = R$, $(\!|Targets|\!)_R \downarrow_{\mathsf{no\text{-}match}} = \emptyset$ and $(\!|Targets|\!)_R \downarrow_{\mathsf{indeterminate}} = \emptyset$. Functions $Ralg(Rules)_R$, given a set $Rules$ of rules and a set R of requests, return decision tuples of the form

$$\begin{array}{l}(\ \mathsf{permit} : \{r \in R \mid Ralg(Rules, r) = \mathsf{permit}\}\,; \\ \ \ \mathsf{deny} : \{r \in R \mid Ralg(Rules, r) = \mathsf{deny}\}\,; \\ \ \ \mathsf{not\text{-}applicable} : \{r \in R \mid Ralg(Rules, r) = \mathsf{not\text{-}applicable}\}\,; \\ \ \ \mathsf{indeterminate} : \{r \in R \mid Ralg(Rules, r) = \mathsf{indeterminate}\}\ )\end{array}$$

Basically, such tuples are calculated by relying on the auxiliary functions $Ralg(Rules, r)$ whose definitions are given in [3, Appendix C].

The semantics definition of a policy set is similar to that of a single policy:

$$\begin{array}{l}[\![\,\{Palg\,;\mathsf{target}:\{\,Targets\,\}\,; Policies\}\,]\!]_R = \\ \ \ (\ \mathsf{permit} : Palg(Policies)_{R_m} \downarrow_{\mathsf{permit}}\,; \\ \ \ \ \mathsf{deny} : Palg(Policies)_{R_m} \downarrow_{\mathsf{deny}}\,; \\ \ \ \ \mathsf{not\text{-}applicable} : Palg(Policies)_{R_m} \downarrow_{\mathsf{not\text{-}applicable}} \cup (\!|Targets|\!)_R \downarrow_{\mathsf{no\text{-}match}}\,; \\ \ \ \ \mathsf{indeterminate} : Palg(Policies)_{R_m} \downarrow_{\mathsf{indeterminate}} \cup (\!|Targets|\!)_R \downarrow_{\mathsf{indeterminate}}\ )\end{array}$$

where R_m stands for $(\!|Targets|\!)_R \downarrow_{\mathsf{match}}$, and function $Palg(Policies)_R$ returns a decision tuple calculated by applying the algorithm $Palg$ to the enclosed policies. It is worth noticing that the definitions of the policy combining algorithms slightly differ from the corresponding rule combining algorithms.

Finally, given a set R of access requests, the semantics of a top-level term $\{Palg\,; Policies\}$ is determined by applying the definition for policy sets and by letting $R_m = R$, $(\!|Targets|\!)_R \downarrow_{\mathsf{no\text{-}match}} = \emptyset$, and $(\!|Targets|\!)_R \downarrow_{\mathsf{indeterminate}} = \emptyset$.

We conclude the section by showing how the semantics definitions presented so far apply to the policy example from the epSOS project, introduced in Sections 3. Given a set R of requests, the permit set of the decision tuple returned by the application of function $[\![\cdot]\!]_R$ to this policy is as follows:

$$\begin{array}{l}\{r \in R \mid (\mathsf{subject.role}, \text{“medical doctor”}) \in r \\ \qquad \wedge\, (\mathsf{subject.purposeofuse}, \text{“TREATMENT”}) \in r \\ \qquad \wedge\, (\mathsf{resource.resource\text{-}id}, \text{“34133-9”}) \in r \\ \qquad \wedge\, (\mathsf{action.action\text{-}id}, \text{“Read”}) \in r \\ \qquad \wedge\, (\mathsf{string\text{-}subset}(\mathsf{string\text{-}bag}(\text{“PRD-003”}, \text{“PRD-005”}, \text{“PRD-010”}, \text{“PRD-016”}), \\ \qquad\qquad\qquad \mathsf{subject.permission})) \cdot r = \mathsf{true}\ \}\end{array}$$

As expected, these are all those requests in R that are issued by a medical doctor, with appropriate permissions, for read accessing a patient summary for treatment purpose. The deny set of the decision tuple consists of all requests in R that match with the policy's target but are not in the set above, i.e. they do not satisfy the target and condition of the first rule. The remaining requests in R belong to the not-applicable set, since the policy never evaluate to indeterminate. We refer the interested reader to [28] for a step-by-step computation of the above decision tuple and further examples.

5 Tools

The implementation of the formalisation presented in the previous sections is made in Java, by also using the ANTLR tool [34] for parsing generation. Our tool "compiles" a policy written in the syntax proposed in Section 3 into a Java class following the semantics rules defined in Section 4. Thus, a repository storing some policies consists of a Java archive containing all the Java classes generated from the policies. A policy decision is then computed by executing the generated code with the requests passed as parameters to an entry method.

For long-lasting repositories where policy changes are infrequent, this approach is convenient, since no policy's XML Document trees need to be loaded in memory and parsed for each request. Instead this approach does not fit well in situations where the policy repository changes on-the-fly.

Specifically, we have defined two separate parsers: one for the proposed XACML syntax and another one for the rule condition expressions. Each parser is defined so that, every time a syntactic category is identified within a policy term, the corresponding Java method is included into the class under generation. The generated class exploits three lists for representing the matching tuples computed during the evaluation of targets. Indeed, when a target is found, the corresponding matching function is retrieved from a specific data structure, i.e. a 'function table' containing the code implementing all functions defined by the standard. The operators $\wedge$, $\vee$, and $\sqcap$ are used to maintain the lists of requests.

Rules are created according to the corresponding rule combining algorithm: if targets and conditions are satisfied, the algorithm is applied and the decision tuples are returned to the caller. Here, to deal with conditions, a factory method is used to load the current implementation of the expression evaluator. The strategy used in this version of the tool follows the same paradigm as the XACML syntax implementation: when a new condition is satisfied, a Java file is created on-the-fly and compiled. Policies and policy sets are implemented in a way similar to the implementation of rules, relying on the policy-combining algorithms. When targets, rules, and policies are evaluated, the resulting lists representing the decision tuples will be returned to the caller.

A web interface to the tool is available online at `http://rap.dsi.unifi.it/xacml_tools`. It permits to practice with the implementation by using sample policies. The web interface gives the possibility to create XACML requests and, then, to obtain the decision computed by the engine.

6 Concluding Remarks

We defined a formal semantics of XACML that aims at clarifying all ambiguous and intricate aspects of the XACML standard and, hence, at conveniently driving implementations. To demonstrate the feasibility and effectiveness of our approach, we fully implemented the semantics as a Java tool.

Another significant advantage of our formalisation is that it paves the way for the development of reasoning tools supporting the analysis of XACML policies.

For example, *equivalences* and *preorders* among (syntactically) different policies could be defined based on their semantics denotations and then used to more compactly store the policies or to more efficiently compute a decision. Thus, two policies could be considered as equivalent if their associated decision tuples coincide or, simply, have the same permit set (indeed, sometimes it does not matter the reason why the access is not permitted, as e.g. with a *deny-biased* PEP [3, Section 7.1.2] that allows the access if the decision taken by the PDP is permit and denies the access in all other cases). We leave the investigation of policy relations as a future work.

We also intend to develop techniques, based on our formal semantics, for studying the application of the *least-privilege* concept [35], in order to determine the requests using the least amount of privilege necessary to satisfy a given XACML policy. To this aim, we will consider an approach where *weights* (indicating the access privilege level[7]) are associated to request data and are used to identify, within the permit set of the decision tuple associated to the considered policy, the requests with minimum total weight. We will also exploit our semantics as a basis for studying *separation of duty* aspects of XACML policies.

We also plan to extend our Java-based framework with other tools, e.g. for translating XACML policies written in the original XML format into polices written in our syntax, and vice versa, and for generating XACML requests, as variations of a template, to be input by the evaluation tool already available. We intend to determine the performances of our tool and to compare them with those of the most notable XACML implementations.

References

1. Ferraiolo, D., Kuhn, R.: Role-based access control. In: NIST-NCSC National Computer Security Conference, pp. 554–563 (1992)
2. NIST: A survey of access control models (2009), http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf
3. OASIS XACML TC: eXtensible Access Control Markup Language (XACML) version 2.0 (2005), http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip
4. The epSOS project: A european ehealth project, http://www.epsos.eu
5. The Nationwide Health Information Network (NHIN): an American eHealth Project (2009), http://healthit.hhs.gov/portal/server.pt
6. OASIS: Cross-Enterprise Security and Privacy Authorization (XSPA) Profile of XACML v2.0 for Healthcare v1.0 (2009), http://www.oasis-open.org
7. OASIS Security Services TC: Assertions and protocols for the OASIS security assertion markup language (SAML) v2.02 (2005), http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
8. Namli, T., Dogac, A.: Implementation Experiences On IHE XUA and BPPC. Technical report, Software Research and Development Center, Middle East Technical University Ankara (December 2006)

[7] For example, the privilege level corresponding to datum "head physician" would be higher than the level of "nurse", which would be higher than that of "anonymous".

9. Universidad de Murcia: UMU-XACML-Editor (2008), http://sourceforge.net/projects/umu-xacmleditor/
10. Bradner, S.: Key words for use in rfcs to indicate requirement levels (1997)
11. Kolovski, V., Hendler, J.A., Parsia, B.: Analyzing web access control policies. In: WWW, pp. 677–686. ACM (2007)
12. Bryans, J.: Reasoning about XACML policies using CSP. In: SWS, pp. 28–35. ACM (2005)
13. Hoare, C.: Commmunicating Sequential Processes. Prentice-Hall (1985)
14. Bryans, J., Fitzgerald, J.S.: Formal Engineering of XACML Access Control Policies in VDM++. In: Butler, M., Hinchey, M.G., Larrondo-Petrie, M.M. (eds.) ICFEM 2007. LNCS, vol. 4789, pp. 37–56. Springer, Heidelberg (2007)
15. Fitzgerald, J., Larsen, P., Mukherjee, P., Plat, N., Verhoef, M.: Validated Designs for Object-oriented Systems. Springer, Heidelberg (2005)
16. Zhang, N., Ryan, M., Guelev, D.P.: Evaluating Access Control Policies through Model Checking. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 446–460. Springer, Heidelberg (2005)
17. Zhang, N., Ryan, M., Guelev, D.P.: Synthesising verified access control systems in XACML. In: FMSE, pp. 56–65. ACM (2004)
18. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. In: ICSE, pp. 196–205. ACM (2005)
19. Tschantz, M.C., Krishnamurthi, S.: Towards reasonability properties for access-control policy languages. In: SACMAT, pp. 160–169. ACM (2006)
20. OASIS XACML TC: Available XACML Implementations (2011), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#other (last visited September 21, 2011)
21. Proctor, S.: SUN XACML (2011), http://sunxacml.sf.net (last visited September 21, 2011)
22. The Herasaf consortium: HERAS^{AF}, http://www.herasaf.org
23. Liu, A.X., Chen, F., Hwang, J., Xie, T.: Xengine: a fast and scalable XACML policy evaluation engine. In: SIGMETRICS, pp. 265–276. ACM (2008)
24. ISSRG: The Modular PERMIS Project, http://sec.cs.kent.ac.uk/permis/
25. Foster, I.T.: Globus toolkit version 4: Software for service-oriented systems. J. Comput. Sci. Technol. 21(4), 513–520 (2006)
26. Barton, T., et al.: Identity federation and attribute-based authorization through the globus toolkit, shibboleth, gridshib, and myproxy. Technical report, National Center for Supercomputing Applications, University of Illinois (2006)
27. Chadwick, D.W., Zhao, G., Otenko, S., Laborde, R., Su, L., Nguyen, T.A.: Permis: a modular authorization infrastructure. Concurrency and Computation: Practice and Experience 20(11), 1341–1357 (2008)
28. Masi, M., Pugliese, R., Tiezzi, F.: Formalisation and Implementation of the XACML Access Control Mechanism (full version). Technical report, Dipartimento di Sistemi e Informatica, Univ. Firenze (2011), http://rap.dsi.unifi.it/xacml_tools
29. Clark, J., DeRose, S.: XML Path Language (XPath) version 1.0 (1999), http://www.w3.org/TR/xpath
30. The IHE Initiative: IT Infrastructure Technical Framework (2009), http://www.ihe.net
31. Health Level Seven organization: Hl7 standards (2009), http://www.hl7.org

32. The Regenstrief Institute: Logical observation identifiers names and codes (LOINC), http://www.loinc.org
33. IEEE Computer Society: IEEE Standard for Binary Floating-Point Arithmetic IEEE Product No. SH10116-TBR (1985)
34. Parr, T.J., Quong, R.W.: ANTLR: A Predicated-LL(k) Parser Generator. Software Practice and Experience 25, 789–810 (1994)
35. Saltzer, J.H.: Protection and the Control of Information Sharing in Multics. Commun. ACM 17, 388–402 (1974)

A Task Ordering Approach for Automatic Trust Establishment*

Francisco Moyano, Carmen Fernandez-Gago, Isaac Agudo, and Javier Lopez

Department of Computer Science, University of Malaga, 29071, Málaga, Spain
{moyano,mcgago,isaac,jlm}@lcc.uma.es

Abstract. Trust has become essential in computer science as a way of assisting the process of decision-making, such as access control. In any system, several tasks may be performed, and each of these tasks might pose different associated trust values between the entities of the system. For instance, in a file system, reading and overwriting a file are two tasks that pose different trust values between the users who can carry out them. In this paper, we propose a model for automatically establishing trust relationships between entities considering an established order among tasks.

1 Introduction

Trust has become an issue of paramount importance when considering systems security. Despite of its importance, a clear, standard definition of trust has not been provided yet. However, it is wide accepted that trust might assist decision-making processes, such as those involved in authorization schemes.

If establishing a definition of trust is very important, how to measure it is also a matter of research and can vary depending on the context where it is applied and the problem that the trust model is meant to solve. What it is mostly common among all the definitions of trust is that it involves a trustor (entity that trusts) and a trustee (the entity on which the trustor places its trust). Thus, the trustor places some trust on the trustee to perform a given task. A usual example to understand this can be given by the fact that one might trust his mechanic for repairing his car but not for fixing his teeth.

This example shows very unrelated tasks but in some cases entities of a system perform tasks that sometimes are overlapping or related. In this paper, we address the issue of how to derive trust values for entities using an established order between tasks. We consider that the trust relationships among entities in a system can be expressed as a trust graph where the edges are the trust values between two of them for a given task. Considering an order on the tasks will

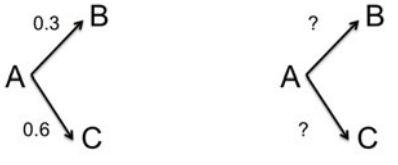
* The research leading to these results have received funding from the European Community's Seventh Framework Programme FP7/2007-2013 as part of the Network of Excellence NESSoS (www.nessos-project.eu) under grant agreement number 256980. The first author is funded by the Spanish Ministry of Education through the National F.P.U. Program.

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 75–88, 2012.

allow us to determine the trust graph of one task based on the trust graph of the other task.

Consider, for example, a file system with several users each of them able to perform two different tasks on it: *read a file* and *write a file*. Users of the system might trust other users to perform these tasks. The organization that owns this file system estimates that, due to privacy regulations, reading a file poses a high risk, and decides that reading a file should be more restrictive than writing a file. Two questions arise: on the one hand, how and under which criteria to define an order between these tasks? On the other hand, let us assume that the organization assigns trust values to the users of the systems for the task *read a file*. Suppose that user A trusts user B to read a file with trust value 0.3, while the same user A trusts user C with value 0.6 for the same task, as shown in Figure 1a. Is it possible to automatically derive trust values for these users regarding the task *write a file* (Figure 1b), if we know the relationship between this task and the task *read a file*? We intend to address these two questions by first defining an order on the tasks of a system and then defining a trust model that can be used for establishing the unkown trust relationships.

The paper is organized as follows. Section 2 gives an overview of similar work carried out prior to this paper. Then, in Section 3, a graph-based trust evaluation model is introduced, as it encompasses the foundations required for the rest of the paper. The order between tasks is explained in Section 4, and in Section 5 our proposed model is presented. In Section 6, our model is applied to an e-Health scenario. Finally, Section 7 presents the conclusions, as well as some relevant lines for future research.



(a) Trust Graph for Task *Read a File*

(b) Trust Graph for Task *Write a File*

Fig. 1. Trust Graphs for a File System with three Users

2 Related Work

To the best of our knowledge, the idea of exploiting an order among tasks to perform automatic trust evaluations is new. However, enough work has been previously carried out in related areas. Trust models have become very important for many computer systems and networks (see [6] for a survey). One of the most

critical and important issues for these models include how to quantify trust values between the entities of the systems, i.e, trust metrics. There are many ways to define trust metrics, ranging from simple, discrete models with *trusted* and *not trusted* values, to complex models using logical formulae like BAN logic [5], vector like approaches [8], or probabilities ([9] provides a survey) and fuzzy logic [13]. Flow models such as Advogato's reputation system [10] or Appleseed [17,18] use trust transitiveness.

In [3], a formal model is proposed to compute trust metrics between two any entities in a trust graph, using sequential operators to compute a value for a given graph path, and parallel operators to compute a final value from several graph paths. Our paper follows one of its future research proposals and builds upon some of its definitions (see Section 3).

Other approaches focus on delegation and recommendation purposes, such as [14] and [7], respectively. The former proposes a formal approach to assess the trustworthiness of potential delegatees in the context of the task to be delegated, ensuring that the choice does not cause a security breach. The latter provides an approach to take subjectivity into account when performing recommendations, in such a way that the same scalar value might mean different things to two different users.

The notions of risk and trust, as well as how they relate to each other, have also been paid attention in the literature. In [16], the authors declare that trust delegation implies risk assumptions, and propose a semirring-based trust model that takes into account the evaluation of risk and privacy for trust establishment. A mechanism that takes both trust and risk into account in order to make access control decisions is presented in [11]. Likewise, [12] considers risk and trust as two important, independent factors to strengthen interactions in e-commerce. Finally, in [4], user trust is defined as an asset. Then, by using asset-oriented risk analysis, the authors analyze which threats and vulnerabilities may cause a reduction in user trust.

3 A Graph-Based Trust Metric Model

Trust can be defined as the level of confidence that an entity e_1 places on another entity e_2 for perfoming a task in a honest way. As explained in [3], trust for one task can be modelled using a weighted graph where the vertices are identified with the entities of the community and the edges correspond to trust relationships between entities regarding the task. This graph actually is a weighted digraph, since any two entities in the graph do not need to have the same level of trust in each other. Now, if we consider having n tasks instead of just one, then we would have n weighted digraphs and as a result, a multigraph.

Next we provide some definitions based on those given in [3].

Definition 1 (Trust Domain). *A trust domain is a partially ordered set* $(TD, <, 0)$ *where every finite subset of* TD *has a minimal element in the subset and* 0 *represents the minimal element of* TD.

The trust domain represents the set of all possible trust values that any entity of the system might hold or might place on other entities.

Each entity in the system makes trust statements about the rest of the entities, regarding the task considered for each case. Those trust statements are defined as follows,

Definition 2 (Trust Statement). *A trust statement is an element (Trustor, Trustee,Task,Value) in $E \times E \times T \times TD$ where, E is the set of all entities in the system; T is a partially ordered set representing the possible tasks, where the order established on tasks is $\preceq$; and TD is a Trust Domain.*

The set $G_x = \{(e_1, e_2, x, t) \in E \times E \times T \times TD\}$ allows building the graph for task x.

Definition 3 (Trust Evaluation). *A trust evaluation for a task $x \in T$ is a function $\mathcal{F}_x : E \times E \longrightarrow TD$*

This function provides, for each task x and for each pair of entities e_1 and e_2, a value t of the trust domain that e_1 places on e_2 to perform x.

4 Tasks Dependencies

An order $\preceq$ amongst tasks is mentioned in Definition 2. However, it was not specified there how to define it. The main contribution of this paper consists of exploring how this order can be used in order to calculate trust values between entities. The order between tasks imposes certain conditions on the trust values of one task regarding another task. For the sake of completeness, a possible criteria to determine the order between tasks is provided in section 4.2, but any other possible criteria could be taken into account.

4.1 Motivation

Providing a task order might make easier for a trust manager the assignment of trust values to the entities which are to perform these tasks. Although a formal definition is given later, let us for now think of x_0 and x as two tasks in T that we would like to classify w.r.t. an order. Let us assume that x_0 is lower w.r.t. this order. This means that x_0 is a reference for x in order to build its trust graph. Thus, if we consider that we can deduce the gaph of the highest tasks from the lowest tasks graphs, we could semi-automatize the process of trust values assignment. Thus, starting from the lowest task/s, we could build the graphs and trust values for the rest of tasks repeating the process downwards through the order chain, saving the trust manager the tedious work of sketching one trust graph for each task.

4.2 Task Domain

Definition 4 (Task Domain). *A Task Domain is a finite partially ordered set* $(T, \preceq)$*, where* $\preceq$ *is defined as follows: let* x_0*,* x *be two tasks in* T*.* $x_0 \preceq x$ *if and only if* $R(x_0) > R(x)$*, where* $R(x) : T \longrightarrow \mathbb{R}^+$ *is a function that given a task, returns a positive real number.*

Lemma 1. *For any given task* x*, either it is minimal or there is a minimal task* x_0 *such that* $x_0 \preceq x$

This can be easily proved if we consider the finite set $R(T) \in \mathbb{R}^+$, which represents the set of the images of the function R. Note that in $\mathbb{R}$, finite sets have a minimum element. Since T is a finite set, $R(T)$ is finite as well.

A risk assessment process, focusing on the scope and context of the task, would analyze the threats, vulnerabilities, possible losses, and other issues regarding the impact of the task on the system, and would return a number between 0 (no risk at all) and an established upper bound. Given that any risk assessment process is a rather subjective concept, we can refine this value as $R_e(x)$, which refers to the risk assessed by entity e for task x.

However, the final order between tasks should consider the opinions of all entities of the system. Thus, the final risk function (the one used for ordering the tasks) should involve the local risk functions of each entity. Let us consider an example of a system with n entities, namely e_1, e_2, ..., $e_n \in E$, and $x \in T$. $R(x)$ could be the average risk assessment, that is,

$$\frac{R_{e_1}(x) + R_{e_2}(x) + ... + R_{e_n}(x)}{n}$$

where $R_{e_i}(x)$ is the risk assessment performed by entity e_i for task x.

Once we have this value for all tasks, we can order the tasks. Let us assume that we have the order among tasks of Figure 2, where x_1, x_2, ... , x_n are the tasks in the system, and the graph declares that $x_1 \preceq x_2$ ... $x_{n-1} \preceq x_n$. Now, we can take advantage of this order to automatically build the trust graph for x_i from the trust graph of x_{i-1}, $i > 1$.

Fig. 2. Task Order

4.3 Trust Assumptions

Some assumptions about trust have been made in the definition of our model. They are listed next:

1. **Higher risky tasks should impose lower trust values among the entities.** This assumption can be reformulated as follows: if entity e_1 trusts entity e_2 with value t_1 to perform task x_1 (which is highly risky), entity e_1 could trust entity e_2 with value $t_2 \geq t_1$ to perform task x_2 (which is less risky). This also happens in real life. We would implicitly trust someone unknown higher to perform a simple, safe task than a risky, complex one. As a consequence of this assumption and our order definition, the trust values for x_i should be lower than the trust values for x_j if $x_i \preceq x_j$.
2. **Trust in entity e to perform task x is proportional to the amount of risk that entity e assigned to x.** If an entity considers that a task is risky, it will more likely take the necessary measures to ensure its successful execution.
3. **People tend to trust similar persons.** If we know that someone has a similar set of values or think in a very similar way as we do, we tend to trust that person more. In the case of entities, those entities which perform a similar risk assessment for most of the tasks will probably have similar worries and similar goals, thus they will be able to trust more each other.
4. **Mistrust should be preserved.** If entity e_1 does not trust entity e_2 to perform a task, there is not reason a priori to assume that it should trust it to perform another related task.

5 Model for Automatic Trust Values Computation

Up to now, definitions have been provided in order to establish an order on the tasks of a system. Now, we will use this order to automatically compute trust values for the entities which execute these tasks, while respecting the assumptions of Section 4.3. For this purpose, one more definition is required.

Definition 5 (Entities Divergence). *Let $e_1, e_2 \in E$ be two entities of the system. Let x_1, x_2, ..., $x_n \in T$ be the tasks of the system. We define the Entities Divergence (ED) between e_1 and e_2 as* $ED(e_1, e_2) = |R_{e_1}(x_1) - R_{e_2}(x_1)| + |R_{e_1}(x_2) - R_{e_2}(x_2)| + ... + |R_{e_1}(x_n) - R_{e_2}(x_n)|$.

Definition 5 provides a way to measure how close two entities are between them. This is the way how we incorporate assumption 3 of Section 4.3 into our model.

As we mentioned in Section 4.2, we would like to automatically generate the trust values of x_j from those trust values of x_i, being $x_i \preceq x_j$. Using the notation and definitions introduced in Section 3, what we want is to compute, for every trustor (e_1) and for every trustee (e_2), $\mathcal{F}_{x_j}(e_1, e_2)$ from $\mathcal{F}_{x_i}(e_1, e_2)$.

We have to consider all the minimal tasks as well. If we think of our model as a recursive model, in which the trust values for the current task depend on the trust values of the previous task, the trust graph for the minimal tasks would represent the base case. A graph for these minimal tasks should be sketched in order to assign the initial trust values. This assignment is made beforehand, since entities are unknown between them, and have neither knowledge nor experience with the other entities to make an informed decision on the initial trust values.

However, if such information exists, it could be taken it into account during this assignment.

Note that assumption 1 in Section 4.3 states that the trust values for x_j should be higher than the trust values of x_i if $x_i \preceq x_j$. We can model it declaring that $\mathcal{F}_{x_j}(e_1, e_2)$ represents an increment over the value $\mathcal{F}_{x_i}(e_1, e_2)$. Thus, we could say that the former trust values depend on the latter ones, and at the same time, monotony property would hold: if $x_i \preceq x_j$, $\mathcal{F}_{x_i}(e_1, e_2) \leq \mathcal{F}_{x_j}(e_1, e_2)$ for any pair of entities e_1, e_2.

The question that arises is how this increment should be done, and here is where assumptions 2 and 3 of Section 4.3 come into play.

Definition 6 (Trust Incremental Value). *We define the Trust Incremental Value as a function* $TIV : E \times E \times T \longrightarrow \mathbb{R}^+$ *that holds the following properties:*

1. *For all entities* e_1 *(trustor),* e_2 *(trustee)* $\in E$*, and for all tasks* $x \in T$*,* $TIV(e_1, e_2, x) \geq 1$
2. TIV *should be inversely proportional to the divergence between entities, thus more similar entities will tend to trust each other.*
3. TIV *should be directly proportional to the assessed risk by the trustee. A trustee that considers a task to be very risky will be more trusted by the rest of entities for performing such task.*

As it can be noticed from the above definition, the TIV depends on both the task risk assessed by the trustee, (e_2), as stated by assumption 2, and the similarity between the trustor (e_1) and the trustee (e_2), represented by the ED (see Definition 5), as declared by assumption 3. It is out of the scope of this paper to provide a concrete definition for this value, although it would constitute an interesting future research.

Next definition explains how to compute the actual trust values:

Definition 7 (Order-dependent Trust Evaluation). *Let* $x_i, x_j \in T$ *be two tasks of the system, in such a way that* $x_i \preceq x_j$*. Let* $e_1, e_2, ..., e_m \in E$ *be the entities of the system. Then, for any pair of entities* e_u*,* e_v*,* $\mathcal{F}_{x_j}(e_u, e_v) = TIV(e_u, e_v, x_j)\mathcal{F}_{x_i}(e_u, e_v)$

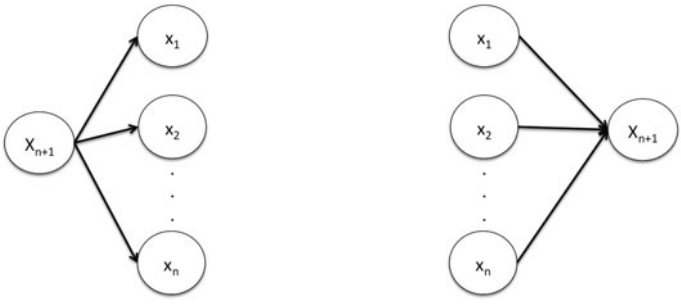
Note that assumption 4 is preserved as well, as in the case of mistrust for task x_i (i.e. $\mathcal{F}_{x_i}(e_u, e_v) = 0$), mistrust is preserved for task x_j (i.e. $\mathcal{F}_{x_j}(e_u, e_v) = 0$). Likewise, if x_i is a minimal task, the trust values for every pair of entities should be explicitly provided by the trust manager.

Also note that the purpose of our proposed trust model is to assign initial trust values. This is an important consideration, since this model does not cope with malicious entities, which might rate tasks with a high risk in order to gain higher trust. Thus, once the initial trust values have been calculated, the subsequent values are calculated with an appropriate trust model for the application, considered for each case.

5.1 Tasks Comparison

Up to now, we have only considered the case where we have a simple order configuration, as depicted in Figure 2. However, it might happen that two or more tasks are not comparable. Recall our task order definition (Definition 4). It implicitly stated that two tasks are comparable only when the total risk assessed for one of them is higher than the total risk assessed for the other one. Thus, when the risk assessment for both tasks is the same, they are non-comparable.

Consider the examples depicted in Figure 3. Figure 3a shows the case in which more than one edge comes out from task x_{n+1} (that is, n non-comparable tasks have an unique precedent task), whereas Figure 3b shows the situation in which several edges arrive in x_{n+1} (that is, x_{n+1} has many non-comparable precedent tasks). The first case does not represent any problem, since every task with an incoming arrow computes its trust values from those of x_{n+1} following Definition 7. However, the second case is more interesting, since the trust values of x_{n+1} can



(a) *1- n* Order (b) *n-1* Order

Fig. 3. Task Dependencies Configurations

be computed from the trust values of tasks x_1, x_2, ..., x_n. Remember that our model is subjected to some assumptions in Section 4.3, and that assumption 1 stated the monotony property. Thus, if $x_1 \preceq x_{n+1}$, $\mathcal{F}_{x_1}(e_1, e_2) \leq \mathcal{F}_{x_{n+1}}(e_1, e_2)$ for all entities e_1, e_2. In addition, as $x_2 \preceq x_{n+1}$, then $\mathcal{F}_{x_2}(e_1, e_2) \leq \mathcal{F}_{x_{n+1}}(e_1, e_2)$, and so forth for the other tasks. Thus, if we want to guarantee the preservation of assumption 1 we should take as the reference task that one of maximum trust value for entities e_1 and e_2.

Informally, the process would be as follows: let us assume that our system in Figure 3b has two entities, namely e_1 and e_2. We want to compute the trust value $\mathcal{F}_{x_{n+1}}(e_1, e_2)$ from the trust values of all precedent tasks x_1, x_2, ..., x_n. First, we should inspect the trust values $\mathcal{F}_{x_1}(e_1, e_2)$, $\mathcal{F}_{x_2}(e_1, e_2)$, ..., $\mathcal{F}_{x_n}(e_1, e_2)$. Then, we would choose the maximum of these values. If there are more than one maximum, we choose one of them in a indeterministic way. Assume that x_i is the task with a maximum $\mathcal{F}_{x_i}(e_1, e_2)$. Then, according to our model,

$\mathcal{F}_{x_{n+1}}(e_1, e_2) = TIV(e_1, e_2, x_{n+1})\mathcal{F}_{x_i}(e_1, e_2)$. In a system with more than two entities, we would repeat this process for every pair of entities.

This is further explained in the next section, where a case study is presented.

6 Case Study: Electronic Health Records Management

In this section, we present a case study in order to provide a clearer vision on the applicability of our model. The case study has been extracted from one of the NESSoS [1] application scenarios. These scenarios are further described in [2].

6.1 e-Health

Electronic Health, or more commonly e-Health, is defined by the World Health Organization as the *use of information and communication technology for health* [15]. e-Health covers a wide range of technologies and scenarios that include interaction between patients and health service providers, as well as peer-to-peer communication and transmission of data between institutions and health professionals.

Systems that manage Personally Identifiable Information (PII) about patients require strict security measures. These systems are often referred to as Electronic Health Records (EHRs), which include patient information created by a health professional, such as laboratory reports, X-ray films, correspondence between health professionals, and so forth.

EHR repositories might be managed by different entities: General Practitioners (GPs) in their office systems, ward or hospital departments, or even by a group of hospitals that could build a circle of trust that share, under some regulations, patient information. As explained in [2], there are several scenes that arise from the EHR management problem, ranging from how to administrate policies in the parameters of EHR access control policies (e.g. groups, roles, etc), to EHR Single-Sign On and transfer of EHR data within an administrative domain. Two very interesting and frequent scenes include reading and writing into an EHR, that could be done in *emergency* mode, setting a flag that may relax the access control policies.

There are other scenarios in the context of e-Health beyond EHR management, as discussed in [2]. These scenarios might entail the use of Internet of Things (IoT) technologies to provide ubiquitous patient monitoring, and the management of patient consent for the transfer of his or her information to other administrative domains.

Since EHR management is a scoped scene, we have chosen this for the application of our model, which is presented next.

6.2 Application of the Model

We proceed to explain the model with the chosen case study.

Entities and Tasks. Access control decisions might be made based on the trust level between the entities in a system. Entities might place trust on each other so that the trust system can decide whether an entity is granted access to a resource. In our example, this resource is the EHR. As explained in the preceeding section, EHR might be managed and accessed by different entities, including general practitioners, hospital departments, patients, or even groups of hospitals. For the sake of generalization, we do not make any assumptions about the type of entities, as we only consider that several entities want to access the EHR with different purposes. From now on, we refer to this entites as e_1, e_2, and e_3.

Regarding the tasks that these entities perform on the EHR, we have chosen *reading EHR*, *writing into EHR*, both in regular and *emergency* mode. We could have chosen other groups of tasks, for example, related to the EHR internal transfer scene. This would include tasks such as *transfer a record*, *transfer a group of records*, *transfer x-ray images*, *transfer hand-written scans*, and so forth. For our example, let x_1 = Reading EHR, x_2 = Writing into EHR, x_3 = Reading EHR in emergency mode, and x_4 = Writing into EHR in emergency mode.

Ordering the Tasks. Up to now, we have identified the entities and the tasks of our system. Now we assume, as described in Definition 4, that a risk assessment process is carried out by each entity e_i for each task x_j by a function $R_{e_i}(x_j)$. Once each entity has assessed the risk for a task x, a final risk function $R(x)$ is applied to compute the final risk value for the task.

The left side of Figure 4 shows a possible outcome of this process. Each position in the table represents the risk assigned by the entity in that row to the task in that column. For example, the number 3 in the position (1,2) represents $R_{e_1}(x_2)$. The last row is the final risk computation for each task, that is, $R(x_i)$. We have assumed that the function R is the average risk, which is rounded down. We have also assumed that entities follow the same numeric range to assign risk values to the task (e.g. a discrete number between 0 and 10). The value R

	x_1	x_2	x_3	x_4
e_1	6	3	8	7
e_2	5	5	7	7
e_3	7	4	8	4
R	6	4	7	6

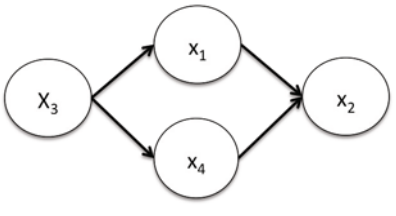


Fig. 4. E-Health Tasks Dependencies

(average risk in this example) determines the order $\preceq$ among the tasks, which is depicted on the right side of Figure 4. The following relations are established: $x_3 \preceq x_1$, $x_3 \preceq x_4$, $x_1 \preceq x_2$, and $x_4 \preceq x_2$.

Sketching the Graph of the Lowest Task. Since task x_3 (i.e. reading EHR in emergency mode) is the lowest task (i.e. the highest risky task), the trust manager has to sketch an initial trust graph for this task. For this purpose, the trust manager could query the entities. Let us assume that the resulting trust graph for task x_3 is the one shown in Figure 6a. Note that e_2 does not place trust on e_1, nor does e_3 on e_2. This is why there is not an arrow in these directions, although a trust relation does exist in the other way around.

Calculating the Entities Divergence. The goal of the model is to compute the trust graphs for the rest of the tasks from the trust graph of the lowest task, namely x_3. The first step is to calculate how much the divergence between entities is, according to Definition 5. Considering the risk values from Figure 4, the divergence values are computed and shown in Table 1. Also note that the concept of Entities Divergence is symmetric.

Table 1. Entities Divergence Table

ED	e_1	e_2	e_3
e_1		4	5
e_2	4		7
e_3	5	7	

Just as an example, let us compute $ED(e_1, e_3)$. According to Definition 5, $ED(e_1, e_3) = |R_{e_1}(x_1) - R_{e_3}(x_1)| + |R_{e_1}(x_2) - R_{e_3}(x_2)| + |R_{e_1}(x_3) - R_{e_3}(x_3)| + |R_{e_1}(x_4) - R_{e_3}(x_4)| = |6 - 7| + |3 - 4| + |8 - 8| + |7 - 4| = 5$.

Calculating the Trust Incremental Values. The Trust Incremental Values (see Definition 6) represent the core concept of the model. Each pair of entities have a TIV for each task. Since mistrust is preserved in our model (see Definition 7), it is only required to compute the TIV for entities that place some level of trust on another entity in a lower task. For example, it is not necessary to compute $TIV(e_2, e_1, x_i)$ for any task x_i, as e_2 does not place trust on e_1 in the lowest task (see Figure 6a).

Our model does not impose a concrete way to compute TIV, but it just provides some criteria that should hold. According to these criteria, some possible TIVs are shown in Figure 5. These values have been established considering both the risk assessed by the trustee (represented as the column), and the entities divergence.

Computing the Final Trust Values. We have all the required information to apply Definition 7, that is, the actual model. At this moment, we have to remember the task order depicted in Figure 4.

We have to compute the trust values for x_1 and x_4 from the trust values of x_3 (see Figure 6a). These trust graphs are depicted in Figure 6b and Figure 6c, respectively.

TIV for x_1	e_1	e_2	e_3
e_1		1.3	1.4
e_2			1.1
e_3	1.2		

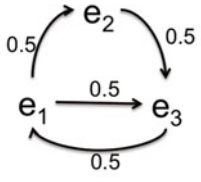
TIV for x_4	e_1	e_2	e_3
e_1		1.8	1.1
e_2			1.05
e_3	1.4		

TIV for x_2	e_1	e_2	e_3
e_1		1.3	1.1
e_2			1.05
e_3	1.05		

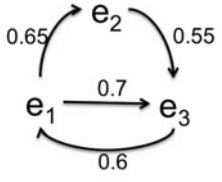
Fig. 5. TIVs for x_1, x_4, and x_2

Just as an example, let us do the calculation to obtain the trust that e_3 places on e_1 to perform x_4. Applying Definition 7,

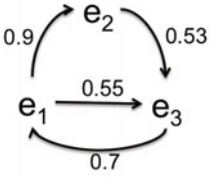
$$\mathcal{F}_{x_4}(e_3, e_1) = TIV(e_3, e_1, x_4)\mathcal{F}_{x_3}(e_3, e_1) = 1.4 * 0.5 = 0.7$$



(a) Trust Graph for Task x_3



(b) Trust Graph for Task x_1



(c) Trust Graph for Task x_4

Fig. 6. Trust Graphs for EHR Scenario

Finally, we should compute the trust values for x_2 from those of x_1 and x_4. Here we have to cope with the case described in Section 5.1, that is, to compute the trust values for one task from the trust values of more than one task. As explained in that section, if we want to ensure the preservation of assumption 1 (see Section 4.3), we have to proceed as follows: first, for a pair of entities, we examine their trust values in x_1 and x_4. Then, we chose the higher trust value, and apply the model to it, multiplying by the TIV of x_2. The final graph for task x_2 is depicted in Figure 7.

For example, to compute the trust value that e_3 places on e_1, we would chose the trust value for that entities in x_4 (0.7), thus:

$$\mathcal{F}_{x_2}(e_3, e_1) = TIV(e_3, e_1, x_2)\mathcal{F}_{x_4}(e_3, e_1) = 1.05 * 0.7 = 0.73$$

Otherwise, to compute the trust value that e_1 places on e_3, we would chose the trust value for that entities in x_1 (0.7), thus:

$$\mathcal{F}_{x_2}(e_1, e_3) = TIV(e_1, e_3, x_2)\mathcal{F}_{x_1}(e_1, e_3) = 1.1 * 0.7 = 0.77$$

After applying the model, we have the initial trust values for all the entities and tasks. From this point onward, another trust model should be in charge of updating the trust values according to the interactions between the entities. For this purpose, it might be required to map the trust values generated in our model to those trust values in the range of the latter model.

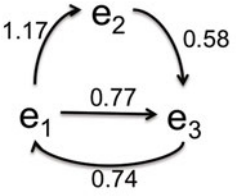


Fig. 7. Trust Graph for x_2

7 Conclusions and Future Work

We have proposed a model to compute trust values in a multi-task system. A multi-task system is characterized because many tasks can be performed by many entities. These tasks, in turn, impose different trust conditions between the entities. The first step in order to achieve our goal has consisted on defining a partial order between these tasks. Then, we automatize the trust evaluation process along this order while respecting some trust assumptions.

Our proposal assumes that the trust graphs for the lowest tasks are provided. Given that there is not information about the entities, this initial assignment might have an influence on the rest of the process. Further research on how to avoid or minimize the impact of this step on the whole process would be very relevant.

Furthermore, a concrete definition of TIV that respects its properties is open to future research, analyzing different alternatives and their impact on the trust values calculation.

An efficient tool implementation, through which to model the tasks order and to automatically generate and draw the trust graphs, would be interesting as future work. Finally, a validation of this tool with the NESSoS scenarios would be very relevant, as this validation could be further used to refine the model and the tool implementation.

References

1. NESSoS FP7 Project: Network of Excellence on Engineering Secure Future Internet Software Services and Systems, `http://www.nessos-project.eu/`
2. Selection and Documentation of the Two Major Application Case Studies. NESSoS Deliverable 11.2 (October 2011)
3. Agudo, I., Fernandez-Gago, C., Lopez, J.: A Model for Trust Metrics Analysis. In: Furnell, S.M., Katsikas, S.K., Lioy, A. (eds.) TrustBus 2008. LNCS, vol. 5185, pp. 28–37. Springer, Heidelberg (2008)
4. Brændeland, G., Stølen, K.: Using Risk Analysis to Assess user Trust: A Net-Bank Scenario. In: Jensen, C., Poslad, S., Dimitrakos, T. (eds.) iTrust 2004. LNCS, vol. 2995, pp. 146–160. Springer, Heidelberg (2004)
5. Burrows, M., Abadi, M., Needham, R.M.: A Logic of Authentication. ACM Trans. Comput. Syst. 8(1), 18–36 (1990)

6. Gil, Y., Artz, D.: A survey of trust in computer science and the semantic web. Web Semantics: Science, Services and Agents on the World Wide Web 5(2) (2011)
7. Hasan, O., Brunie, L., Pierson, J.-M., Bertino, E.: Elimination of Subjectivity from Trust Recommendation. In: The 3rd IFIP International Conference on Trust Management (TM 2009), pp. 65–80 (June 2009)
8. Jøsang, A., Ismail, R.: The Beta Reputation System. In: 15th Bled Electronic Commerce Conference e-Reality: Constructing the e-Economy, Bled, Slovenia (June 2002)
9. Jøsang, A., Ismail, R., Boyd, C.: A Survey of Trust and Reputation Systems for Online Service Provision. Decision Support Systems 43, 618–644 (2007)
10. Leiven, R.: Attack Resistant Trust Metrics. PhD thesis, University of California at Berkeley (2003)
11. Li, Y., Sun, H., Chen, Z., Ren, J., Luo, H.: Using trust and risk in access control for grid environment. In: Proceedings of the 2008 International Conference on Security Technology, SECTECH 2008, pp. 13–16. IEEE Computer Society, Washington, DC, USA (2008)
12. Li, Y., Zhao, M., Sun, H., Chen, Z.: A trust and risk framework to enhance reliable interaction in e-commerce. In: IEEE International Conference on E-Business Engineering, pp. 475–480 (2008)
13. Seo, Y., Han, S.: Local scalar trust metrics with a fuzzy adjustment method. TIIS 4(2), 138–153 (2010)
14. Toahchoodee, M., Xie, X., Ray, I.: Towards Trustworthy Delegation in Role-Based Access Control Model. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 379–394. Springer, Heidelberg (2009)
15. World Health Organization. eHealth Tools and Services: Needs of Member States (2006)
16. Zhang, M., Yang, B., Zhu, S., Zhang, W.: Ordered semiring-based trust establish model with risk evaluating. I. J. Network Security 8(2), 101–106 (2009)
17. Ziegler, C.N., Lausen, G.: Spreading Activation Models for Trust Propagation. In: IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE 2004), Taipei (March 2004)
18. Ziegler, C.-N., Lausen, G.: Propagation Models for Trust and Distrust in Social Networks. Information Systems Frontiers 7(4-5), 337–358 (2005)

An Idea of an Independent Validation of Vulnerability Discovery Models*

Viet Hung Nguyen and Fabio Massacci

Università degli Studi di Trento, I-38100 Trento, Italy
{vhnguyen,fabio.massacci}@disi.unitn.it

Abstract. Having a precise vulnerability discovery model (VDM) would provide a useful quantitative insight to assess software security. Thus far, several models have been proposed with some evidence supporting their goodness-of-fit. In this work we describe an independent validation of the applicability of these models to the vulnerabilities of the popular browsers Firefox, Google Chrome and Internet Explorer. The result shows that some VMDs do not simply fit the data, while for others there are both positive and negative evidences.

1 Introduction

The vulnerability discovery process normally refers to the post-release stage where people identify and report security flaws of a released software. Vulnerability discovery models (VDM) operate on the known vulnerability data to do a quantitative estimation of the vulnerabilities present in the software. Successful models can be useful hints for both software vendors and users in allocating resources to handle potential breaches, and tentative patch update. For example, we do not exactly know the day of major snow falls but cities expect it to fall in winter and therefore plan resources for road clearing in that period.

In this paper we consider six proposed VDMs. The first model is Anderson's Thermodynamic(AT) [5]. Rescorla proposed two other models [11]: Quadratic (RQ) and Exponential (RE). The fourth model considered here is Alhazmi & Malaiya's Logistic (AML) model [2]. The fifth is directly derived from a software reliability model, Logistic Poisson (LP) (a.k.a Musa-Okumoto model). The last model is the simple linear model (LN).

Among these models, the AML model has been subject to a significant experimental validation: from operating systems [1, 2, 3, 4] (*i.e.,* Windows NT/95/98/2K/XP, Redhat 6.2/7.1 and Fedora) to browsers [14] (*i.e.,* IE, Firefox, Mozilla), and web servers [15] (*i.e.,* ISS, Apache). The results reported in the literature show that there is not enough evidence to neither reject nor accept AML. Three browsers were considered: one is strongly accepted by AML (Mozilla), one is strongly rejected (IE), and another one is unknown (Firefox).

* This work is supported by the European Commission under projects EU-FET-IP-SECURECHANGE.

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 89–96, 2012.

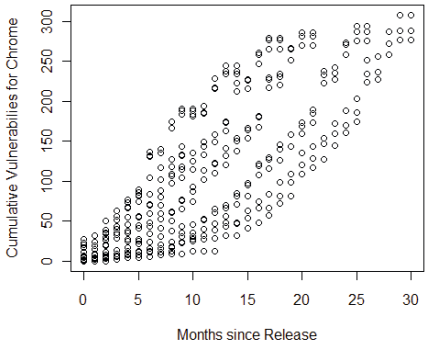


Fig. 1. Google Chrome Firework of Vulnerability Discovery Trends

These inconsistent results may be caused by a combination of factors. First, the authors did not clearly mention what a vulnerability is. For example, the National Vulnerability Database (NVD) reports a number of vulnerabilities which the security bulletin of the vendors do not classify as such. By considering different database we could get different trends.

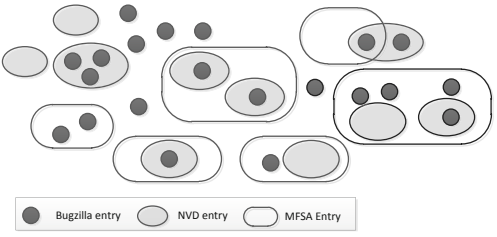
The second problem is that the authors considered all versions of software as a single application, and counted vulnerabilities for this "application". Massacci *et al.* [8] has shown that each Firefox version has its own code base, which may differ by 30% or more from the immediately preceding one. Therefore, as time goes by, we can no longer claim that we are counting the vulnerabilities of the same application. To explain visually this problem, Fig. 1 shows in one plot the cumulative vulnerabilities of the different versions of Chrome in which we restart the counters for each version. It is immediate to see that there is not a single "trend" but a "firework" effect where each version determines its own trajectory.

1.1 Contribution of This Paper

This paper presents an independent validation experiment on the goodness-of-fit of six existing VDMs against the three most popular browsers: Firefox, Google Chrome and Internet Explorer.

- We show that some model (AT) does not work completely. Some (LN, RE, RQ, LP) might not work, and some (AML) may work.
- We find an interesting phenomenon that cumulative vulnerabilities of a life long software may have many saturation points (where the number of vulnerabilities is stable) which might falsify all of existing VDMs.

The rest of the paper is organized as follows. In the subsequent section (§2), we describe our research questions and how to find out the answers. Next we briefly discuss existing VDMs and their formulae (§3). After that, we discuss the



This illustrates different abstract levels of vulnerability: from technical level (Bugzilla) to abstract level (MFSA, Bugzilla). Bugzilla entry denotes technical programming issues (both security and non-security ones). Security bugzilla are ones reported in an MFSA, or referenced by an NVD. The overlaps between notations denote that an report might reference to another report.

Fig. 2. The vulnerability space of Firefox

methodology to conduct the experiment, and a discussion about the result in our experiment (§4). Finally, we present potential threats (§5) to the validity of our work and conclude the paper with future work (§6).

2 Research Questions

The primary question is "does this model fit the observed data?". Frequently when a new VDM is proposed, the authors have done some experiment to validate the applicability of this VDM. Mostly, in their reports the proposed VDMs often have good goodness-of-fit measures. As time goes by, the goodness-of-fit may improve or deteriorate as more data become available (either in terms of data point for the same software or new software to be considered as an instance). This motivate our first research question:

RQ1. *Are existing VDMs able to fit cumulative numbers of vulnerabilities of the popular browsers (i.e., IE, Firefox, and Chrome)?*

To find the answer, we touched another, major and almost foundational issue: "what is a vulnerability?". Most related work did not explicitly discuss this question. Normally, a vulnerability is a security report describing a particular problem of a particular application, for instance: a report in Mozilla Foundation Security Advisories (a.k.a an MFSA entry), or an NVD report of NIST (NVD entry). In the wisdom of many people, an NVD entry is a vulnerability, but there are many other definitions [6, 7, 12].

Fig. 2 illustrates the vulnerability space of Firefox, in which different 'kinds' of Firefox vulnerabilities are coexisted at different level of abstraction.

- *Mozilla Bugzilla* (or *bug*): contains very technical reports for vulnerabilities, but also other normal programming bugs.
- *NVD*: holds high level third-party security reports for several applications, including Firefox. Many NVD entries (gray ovals) mentioning Firefox maintain references to Bugzilla (black circles inside ovals).

Table 1. Formal definitions of six VDMs in the study

Model	*Formula*
Alhazmi-Malaiya Logistic (AML)	$\Omega(t) = \frac{B}{BCe^{-ABt} + 1}$
Anderson Thermodynamic (AT)	$\Omega(t) = \frac{K}{\gamma} \ln(t) + C$
Linear (LN)	$\Omega(t) = At + B$
Logistic Poisson (LP)	$\Omega(t) = \beta_0 \ln(1 + \beta_1 t)$
Rescorla Exponential (RE)	$\Omega(t) = N(1 - e^{-\lambda t})$
Rescorlar Quadratic (RQ)	$\Omega(t) = \frac{At^2}{2} + Bt$

– *MFSA*: are set of vendor's high level security reports for Mozilla's products. Each MFSA entry (rounded rectangle) always references to one or more bugs (black circles inside) responsible for this security flaw. MFSA also holds links to corresponding NVD entries (overlapped ovals).

Depend on the judgement of analysts, different numbers of vulnerabilities are observed and collected. Here, in Fig. 2, if we define a vulnerability is an MFSA, or NVD, or Bugzilla, these numbers are respectively six, ten and fourteen. The fact that we can have a large variance in numbers raise another research problem *"How do different definitions of vulnerability impact the VDMs' goodness-of-fitness?"*. However, we do not present it here due to the limit of space.

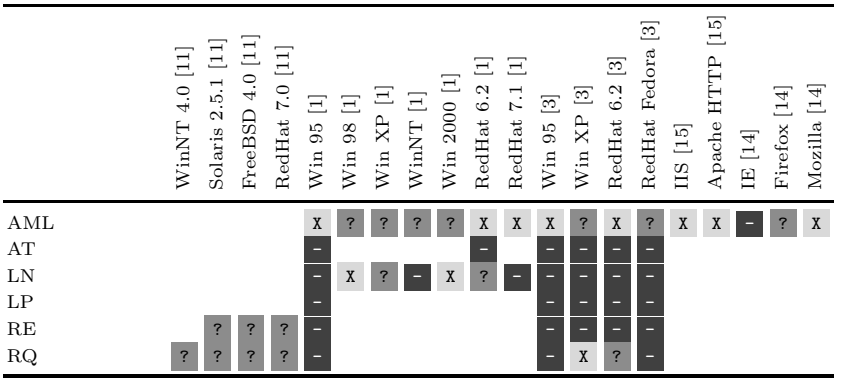
3 Vulnerability Discovery Models

This section provides a quick glance about six VDMs. As denoted in [3], these VDMs are main features of the vulnerability discovery models. Here, only the formulae of these six models are discussed. The detail rationale of models as well as the meaning of each parameter can be found in the original work or in [3]. All these parameters are estimated using non-linear regression on observed data.

– *Alhazmi-Malaiya Logistic (AML)*: proposed by Alhazmi & Malaiya [1], inspired by the s-shape curve.
– *Anderson Thermodynamic (AT)*: the application of this model to vulnerabilities is proposed in [5]. The term *thermodynamic* originates by the analogy from thermodynamics, in which γ accounts for the lower failure rate during beta testing compared to higher rates during alpha testing.
– *Linear model (LN)*: this is the simplest model, and well known by most people. Linear model is often used to express the trend line of data.
– *Logistic Poisson (LP)*: is originated from the field of reliability engineering, also known as Musa-Okumoto model.
– *Rescolar Exponential (RE)*: is proposed by Rescorla [11] while attempting to identify trends in the vulnerability discovery using statistical tests.
– *Rescolar Quadratic (RQ)*: this model is also a work of Rescorla [11], inspired by the Goel-Okumoto in software reliability engineering.

Table 2. The goodness-of-fit of VDMs in other studies

The table reports goodness-of-fit from previous studies. Columns are applications of which vulnerabilities are fitted. The number next to each application is citation to the corresponding study.



4 Validation of VDMs

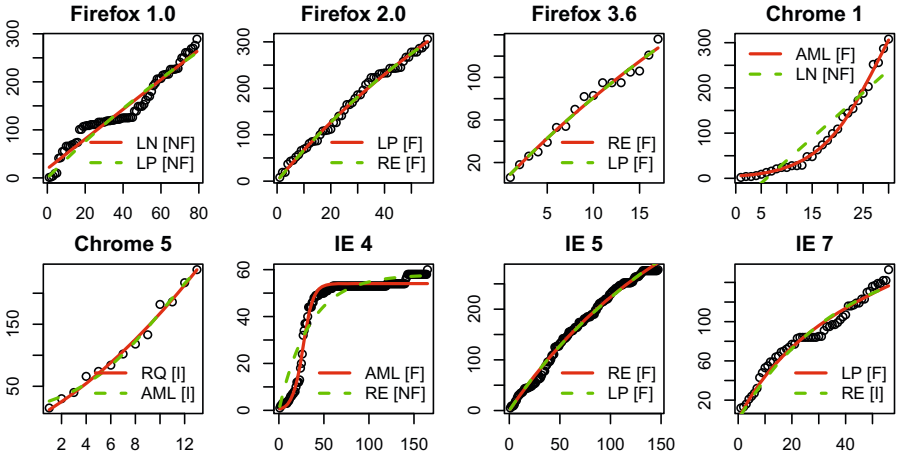
The validation is quire straight forward. We fit VDMs into the data set using R [10] tool. The differences between expected values of each generated model and observed values are calculated and tested by the chi-square (χ^2) goodness-of-fit test. This test is based on χ^2 statistics calculated as follows.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{1}$$

O_i and E_i orderly denote the observed values and expected values generated by VDMs. The smaller χ^2, the higher goodness a VDM gains. In practice, a VDM is acceptably fitted if the χ^2 is less than a critical value, given a significant level (α) and degrees of freedom. The *p-value* here represents the significance of the differences between observed values and expected values. If the *p-value* is small, differences are significant, not by chance. Thus, the smaller *p-value*, the stronger evidence a VDM does not fit the data. Hence, we interpret the goodness-of-fit based on the ranges of *p-value* as follows

- *Not Fit*: $p - value \in [0 \sim 0.05)$, the difference are not by chance. So, this evidence is strong enough to reject the model.
- *Good Fit*: $p - value[0.95 \sim 1.0]$, the difference, in opposite with the previous, is significant small. It is a strong evidence to accept the model.
- *Inconclusive Fit*: $p - value \in [0.05 \sim 0.95)$, there is not enough evidence neither to reject nor accept this model.

Applying goodness-of-fit interpretation discussed above, Table 2 show the goodness-of-fit of VDMs in other studies. In these studies, Windows 95/XP and



This figure illustrates feature goodness in Table 3. The circles indicate cumulative vulnerabilities at a certain time. The horizontal axis (X) is time-in-market measured by the number of months since officially released. The vertical axis (Y) is the cumulative vulnerabilities.

Fig. 3. Goodness of VDMs on browsers in database NVD

RedHat 6.1 have been tested in two studies (*i.e.,* [1,3]) of the same authors, but they seem to be duplicated. Thus we will consider these two experiments as one. According to the table, AML is the one that has been tested in various kinds of applications, *e.g.,* operating systems, web servers, and browsers. Most of the cases, AML shows its outstanding performance (only 1 *Not Fit* over 13 tests). On the contrary, AT model also did not work in all cases. For the other models, the ratio between *Not Fit* and *Inconclusive + Good Fit* is more or less fifty-fifty. Therefore, we cannot conclude anything about the performance of these models.

We run our experiment of five VDMs on seventeen releases. The experiment produces 102 curves, which are impossible to show all of them. Fig. 3 shows the some fitted plots of VDMs for releases using NVD data set. For Firefox v1.0, the cumulative number of vulnerabilities has more than one linear periods. This trend is against the recently three-phase model (*i.e., learning, linear* and *saturation*) proposed by Alhazmi *et al.* [1]. So none of VDM is able to fit (either *Good Fit* or *Inconclusive Fit*) this version. This trend of Firefox vulnerabilities is caused by a large portion of v1.0's code base is inherited in later releases. Thus lots of vulnerabilities applied to Firefox v1.0 are discovered in the newer releases [8]. This phenomenon slightly appears in IE v4.0, but the stable period of this release is long enough[1] for the AML model to obtain a *Good Fit*, nonetheless. For Firefox v3.6 and Chrome v3.5, these releases are still young (less than 16 months old), the vulnerability discovery is in the linear phase. So any VDM that supports linear modeling (or nearly linear), *i.e.,* AML, LN, LQ, RQ, RE, has a chance to fit the data.

[1] A long stable period has larger degree of freedom in the χ^2 test, thus there is more chance that the *p-value* is less than the significant level 0.05.

Table 3. The goodness-of-fit of VDMs using data set NVD

The goodness of fit of a VDM is based on *p-value* in the χ^2 test. *p-value* < 0.05: not fit (–), *p-value* ≥ 0.95: good fit (X), and inconclusive fit (?) otherwise.

	Firefox						Chrome						IE				
Model	v1.0	v1.5	v2.0	v3.0	v3.5	v3.6	v1.0	v2.0	v3.0	v4.0	v5.0	v6.0	v4.0	v5.0	v6.0	v7.0	v8.0
AML	–	–	?	?	?	?	X	?	?	?	?	?	X	?	?	–	X
AT	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	?	–
LN	–	–	X	–	X	?	–	–	–	?	–	–	–	–	–	?	?
LP	–	–	X	?	X	X	–	–	–	–	?	?	–	X	–	X	?
RE	–	–	X	?	X	X	–	–	–	–	?	?	–	X	–	?	?
RQ	–	–	–	?	?	X	–	–	?	?	?	?	–	–	–	–	X

Table 3 reports the goodness-of-fit for 102 curves. Here, instead of reporting a big table of numbers, Table 3 shows the interpretation of *p-value* of the χ^2 tests. This presentation also helps to study at higher abstract level than the raw *p-value*s. Basically, our result is consistent with others. In this table, there are 47 times VDMs can either well fit or inconclusively fit the data, and 55 times they do not work. Roughly speaking, the chance of not fit is about 50%. If we look at each VDM particularly, the AML model appears to be the best one as it obtains more positive results than others. In contract, the AT model seems to be the worst because it could only fit one release (IE v7.0). Meanwhile, other models are equivalent in number of times being rejected and accepted, except the LP model which is likely a bit better. Even though our result confirms the conclusion in previous studies, we still could not claim any strong argument about the goodness-of-fit of these VDMs since the goodness-of-fit might change overtime as more data will be available. We can only say that at the time when we collect data, AML is the best model that can fit most releases in our study; AT model apparently does not applicable; and other models work in haft way.

5 Threats to Validity

Bias in Data Collection. This work employs the same technique discussed in [9] to parse HTML pages of MFSA, and process the XML data of NVD and Bugzilla. Even though the collector tool has been checked for multiple times, it might contain bugs affecting to data collection.

Error in Curve Fitting. We estimate the goodness-of-fit of VDMs by using the Nonlinear Least-Square technique implemented in R (`nls()` function). This might not produce the most optimal solution. That essentially impacts the validity of this work. To mitigate this issue, we additionally employ a commercial tool *i.e.,* CurveExpert Pro[2] to cross check the goodness-of-fit.

6 Conclusion and Future Work

In this work we validated the goodness-of-fit of several VDMs on the three most popular browsers: IE, Firefox and Chrome. Our validation took into account

[2] `http://www.curveexpert.net/`, site visited on 16 Sep, 2011.

the definition of vulnerability which is not adequately considered in previous studies. However we have not enough room to report the result. Even though our experiment is consistent with other studies, but all the experiments so far have only reported the goodness-of-fit of these VDMs at certain time points of a software life cycle. Meanwhile, we need to analyze the evolution of each model in a long period and see how the goodness-of-fit evolves to have a better insight.

Additionally, we have shown the potential impact of different understanding about what a vulnerability is. Hence, it would be interesting to study which one is more appropriate for VDMs in general.

References

1. Alhazmi, O., Malaiya, Y.: Modeling the vulnerability discovery process. In: Proc. of the 16th IEEE Int. Symp. on Software Reliab. Eng., ISSRE 2005 (2005)
2. Alhazmi, O., Malaiya, Y.: Quantitative vulnerability assessment of systems software. In: Proc. of RAMS 2005 (2005)
3. Alhazmi, O., Malaiya, Y.: Application of vulnerability discovery models to major operating systems. IEEE Trans. on Reliab. 57(1), 14–22 (2008)
4. Alhazmi, O., Malaiya, Y., Ray, I.: Security Vulnerabilities in Software Systems: A Quantitative Perspective. In: Jajodia, S., Wijesekera, D. (eds.) Data and Applications Security 2005. LNCS, vol. 3654, pp. 281–294. Springer, Heidelberg (2005)
5. Anderson, R.: Sec. in open versus closed systems - the dance of Boltzmann, Coase and Moore. In: Proc. of Open Source Soft.: Economics, Law and Policy (2002)
6. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
7. Krsul, I.: Software Vulnerability Analysis. PhD thesis, Purdue University (1998)
8. Massacci, F., Neuhaus, S., Nguyen, V.H.: After-Life Vulnerabilities: A Study on Firefox Evolution, its Vulnerabilities and Fixes. In: Erlingsson, Ú., Wieringa, R., Zannone, N. (eds.) ESSoS 2011. LNCS, vol. 6542, pp. 195–208. Springer, Heidelberg (2011)
9. Massacci, F., Nguyen, V.H.: Which is the right source for vulnerabilities studies? an empirical analysis on mozilla firefox. In: Proc. of MetriSec 2010 (2010)
10. R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing (2011) ISBN 3-900051-07-0
11. Rescorla, E.: Is finding security holes a good idea? IEEE S&P 3(1), 14–19 (2005)
12. Schneider, F.B.: Trust in cyberspace. National Academy Press (1991)
13. Sliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: Proc. of the 2nd Int. Working Conf. on Mining Soft. Repo. MSR 2005 (2005)
14. Woo, S., Alhazmi, O., Malaiya, Y.: An analysis of the vulnerability discovery process in web browsers. In: Proc. of 10th IASTED SEA 2006 (2006)
15. Woo, S., Joh, H., Alhazmi, O., Malaiya, Y.: Modeling vulnerability discovery process in apache and iis http servers. C&S 30(1), 50–62 (2011)

A Sound Decision Procedure for the Compositionality of Secrecy*

Martín Ochoa[1,3], Jan Jürjens[1,2], and Daniel Warzecha[1]

[1] Software Engineering, TU Dortmund, Germany
[2] Fraunhofer ISST, Germany
[3] Siemens AG, Germany
firstname.lastname@cs.tu-dortmund.de

Abstract. The composition of processes is in general not secrecy preserving under the Dolev-Yao attacker model. In this paper, we describe an algorithmic decision procedure which determines whether the composition of secrecy preserving processes is still secrecy preserving. As a case-study we consider a variant of the TLS protocol where, even though the client and server considered separately would be viewed as preserving the secrecy of the data to be communicated, its composition to the complete protocol does not preserve that secrecy. We also show results on tool support that allows one to validate the efficiency of our algorithm for multiple compositions.

1 Introduction

The question of compositional model-checking [5] is crucial for achieving scalable verification of systems. Moreover, compositionality of secure protocols can cause unforeseen problems (see for example problems on the SAML based single-sign-on used by Google in [3]). Although this question has been studied extensively in the literature, in this paper we propose a novel methodology to specify protocols such that given a finite set of session variables, compositionality is decidable. This is equivalent to restrict the analysis of processes to finitely many runs. Indeed vulnerabilities in authentication protocols have been shown to be limited to finitely many parallel instantiations [14]. Technically, our analysis generates finite *dependency trees* that can be stored for further deciding on future compositions. The process of merging such trees can be shown to be empirically more efficient than re-analysing the composition from scratch, and constitutes our central contribution. Moreover, this process is relatively sound and complete with respect to the First Order Logic analysis of [10].

To validate our approach we have implemented our algorithm as an extension to the UMLsec Tool Suite. This validates the usability of the approach in a formally sound Software Development process, and has allowed us to measure

* This research was partially supported by the MoDelSec Project of the DFG Priority Programme 1496 "Reliably Secure Software Systems – RS^3" and the EU project NESSoS (FP7 256890).

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 97–105, 2012.

$E ::=$	expression
d	data value ($d \in \mathcal{D}$)
N	unguessable value ($N \in$ **Secret**)
K	key ($K \in$ **Keys**)
$\mathsf{inp}(c)$	input on channel c ($c \in \mathsf{Channels}$)
x	variable ($x \in$ **Var**)
$E_1 :: E_2$	concatenation
$\{E\}_e$	encryption ($e \in$ **Enc**)
$\mathcal{D}ec_e(E)$	decryption ($e \in$ **Enc**)
$\mathcal{S}ign_e(E)$	signature creation ($e \in$ **Enc**)
$\mathcal{E}xt_e(E)$	signature extraction ($e \in$ **Enc**)

Fig. 1. Grammar for simple expressions in the Domain-Specific Language

the efficiency of our approach given the derivation trees for up to 500 small components (amounting to about 1000 messages).

This paper is organized as follows: Sect. 2 presents some preliminaries about stream processing functions, composition and secrecy. Sect. 3 describes the main verification strategy, whereas Sect. 4 shows its application to an insecure variant of TLS. Sect. 5 reports on efficiency of the decision procedure compared to re-verification. Finally, Related Work is discussed on Sect. 6 and we conclude with Sect. 7.

2 Preliminaries

In [10] the underlying process model used to model component communication is based on Broy's stream-processing functions [4]. A *process* is of the form $P = (I, O, L, (p_c)_{c \in O \cup L})$ where $I \subseteq \mathsf{Channels}$ is called the set of its *input channels* and $O \subseteq \mathsf{Channels}$ the set of its *output channels* and where for each $c \in \tilde{O} \stackrel{\text{def}}{=} O \cup L$, p_c is a closed program with input channels in $\tilde{I} \stackrel{\text{def}}{=} I \cup L$ (where $L \subseteq \mathsf{Channels}$ is called the set of *local channels*). From inputs on the channels in $\tilde{I}$ at a given point in time, p_c computes the output on the channel c. Each channel defines thus a stream processing function based on its input variables allowing for a rigorous notion of sequential composition, which is denoted by $\otimes$. For cryptographic protocol analysis, the programs are specified in a domain specific language defined by the expressions as in Fig. 1 and a simple programming language with non-deterministic choice (where loops can be modelled by using local channels). To proceed with the Dolev-Yao secrecy analysis, one defines rules to translate programs to first-order logic formulas. With the predicate $\mathsf{knows}(E)$ we can express the fact that an adversary may know an expression E during the execution of the protocol, therefore it models the *man in the middle*. For example, if-constructs are translated by the following formula:

$$\begin{aligned}\phi(\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p') = \forall i_1, \ldots, i_n. \big[&\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow \\ &[E(i_1, \ldots, i_n) = E'(i_1, \ldots, i_n) \Rightarrow \phi(p)] \\ &\wedge [E(i_1, \ldots, i_n) \neq E'(i_1, \ldots, i_n) \Rightarrow \phi(p')]\big]\end{aligned}$$

To verify the secrecy of data $s \in \mathbf{Secret}$, one then has to check whether the adversary can derive $\mathsf{knows}(s)$, given the formulas that arise from the evaluation ϕ of the single program constructs and the following axioms:

$$\forall E_1, E_2.$$
$$\left[\mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2) \;\Rightarrow\; \mathsf{knows}(E_1 :: E_2) \wedge \mathsf{knows}(\{E_1\}_{E_2}) \wedge \mathsf{knows}(\mathcal{S}ign_{E_2}(E_1))\right]$$
$$\wedge \left[\mathsf{knows}(E_1 :: E_2) \;\Rightarrow\; \mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2)\right]$$
$$\wedge \left[\mathsf{knows}(\{E_1\}_{E_2}) \wedge \mathsf{knows}(E_2^{-1}) \;\Rightarrow\; \mathsf{knows}(E_1)\right]$$
$$\wedge \left[\mathsf{knows}(\mathcal{S}ign_{E_2^{-1}}(E_1)) \wedge \mathsf{knows}(E_2) \;\Rightarrow\; \mathsf{knows}(E_1)\right]$$

The conjunction of the formulae ϕ for all channel programs of a process is called ψ. In the following, we will discuss composition at the level of this First Order Logic translation and not at the underlying stream processing function level because the FOL translation contains implicitly all the possible actions an adversary process could perform (defined by the structural formulas). Moreover, and adversary that completely controls the communication channels between processes, might act as an adaptor creating unforeseen compositions between input and output channels. Therefore we want to approximate the knowledge an adversary can gain given all possible outputs of the processes (considering all possible well-formed inputs).

3 Decision Procedure

If we assume that both P and P' preserve the secrecy of the data value s, our goal is to show a procedure so that we can decide if $\psi(P \otimes P') \nvdash \mathsf{knows}(s)$. In general this does not hold. For example consider a process P which outputs $\{s\}_K$ and a process P' which outputs K^{-1}. Independently this both processes preserve the secrecy of s, but when composed an adversary could trivially compute s. To achieve this, we will construct proof artifacts on each single process called *derivation trees*. Moreover, in order ensure that this trees are finite, we will require that the number of *keys* and *nonces* are also finite and that the conditions in the "if" constructs of the process programs admit only variables that are of type *key* or *nonce*. This will imply the decidability of our approach.

Definition 1 (Subterm). *We say that a symbol* x *is a subterm of the symbol* T *and denote it* $x \hat{\in} T$ *when one of the following holds:*

$x = T$
$T = \{T'\}_K$ *and* $x \hat{\in} T'$
$T = \mathcal{S}ign_K\{T'\}$ *and* $x \hat{\in} T'$
$T = h :: k$ *and* $x \hat{\in} h$ *or* $x \hat{\in} k$

Example $\mathsf{s} \hat{\in} \{\mathsf{s}\}_\mathsf{K}$ but is not true that $\mathsf{K} \hat{\in} \{\mathsf{s}\}_\mathsf{K}$. We denote this by $\mathsf{K} \hat{\notin} \{\mathsf{s}\}_\mathsf{K}$. This means that an adversary could potentially compute s from $\{\mathsf{s}\}_\mathsf{K}$ using the structural formulas with the necessary previous knowledge, but he could not compute K.

Definition 2 (Inverse). *Let $x \hat{\in} J$. We define the cryptographic inverse of a symbol J with respect to x and denote it $\mathsf{J}^{-1}(\mathsf{x})$ in the following way:*

$\mathsf{x}^{-1}(\mathsf{x}) = \epsilon$
If $J{=}h{::}k$ *and* $x \hat{\notin} h$ *then* $\mathsf{J}^{-1}(\mathsf{x}){=}\mathsf{k}^{-1}(\mathsf{x})$
If $J{=}h{::}k$ *and* $x \hat{\notin} k$ *then* $\mathsf{J}^{-1}(\mathsf{x}){=}\mathsf{h}^{-1}(\mathsf{x})$
If $J{=}h{::}k$ *and* $x \hat{\in} k$, $x \hat{\in} h$ *then* $\mathsf{J}^{-1}(\mathsf{x}) = \mathsf{and}(\mathsf{h}^{-1}(\mathsf{x}), \mathsf{k}^{-1}(\mathsf{x}))$
If $J{=}\{J'\}_{\mathsf{K}}$ *or* $J{=}\mathcal{S}ign_{\mathsf{K}}\{\mathsf{J}'\}$ *then* $\mathsf{J}^{-1}(\mathsf{x}) = \mathsf{or}(\mathsf{J'}^{-1}(\mathsf{x}), K^{-1})$.

Example Let $\mathsf{J} = \{\{\mathsf{s}\}_{\mathsf{K}_1}\}_{\mathsf{K}_2}$. Then $\mathsf{J}^{-1}(\mathsf{s}) = \mathsf{or}(K_1^{-1}, K_2^{-1})$ which we will interpret later as "to preserve the secrecy s we need to preserve either K_1^{-1} or K_2^{-1}".

Let $\psi(P)$ be the first order logic formula associated to P. We define $\bar{\psi}(P)$ to be the set of instantiated formulas of $\psi(P)$ with all possible values satisfying the constraints in $\psi(P)$. Since we require that all constraints only contain variables of type *key* or *nonce*, and that the respective sets are finite, then $\bar{\psi}(P)$ is also finite. It is possible to show by induction on the program constructs that $\bar{\psi}(P)$ consists of formulas F_i of the form $\mathsf{knows}(\mathsf{E}_i) \Rightarrow \mathsf{knows}(\mathsf{J}_i)$ for closed expressions E_i and J_i. Let $\mathsf{Pres(x,P)}$ be the following inductively defined predicate:

$$
\begin{aligned}
&[(\,\forall \mathsf{F}_i \in \bar{\psi}(P)\ \mathsf{x} \hat{\notin} J_i\,) \Rightarrow \mathsf{Pres(x,P)})\\
&\wedge\ (\ \forall \mathsf{F}_i \in \bar{\psi}(P)\ (\mathsf{x} \hat{\in} \mathsf{J}_i) \Rightarrow ((\mathsf{Pres}(\mathsf{E}_i,\mathsf{P}) \vee \mathsf{Pres}(\mathsf{J_i}^{-1}(\mathsf{x}),\mathsf{P}))\\
&\wedge\ (\ (\mathsf{x} = \{\mathsf{x'}\}_{\mathsf{K}} \vee \mathsf{x} = \mathcal{S}ign_{\mathsf{K}}\{\mathsf{x'}\}) \Rightarrow (\mathsf{Pres(x',P)} \vee \mathsf{Pres(K,P)})\\
&\wedge\ ((\mathsf{x} = \mathsf{h{::}k} \Rightarrow (\mathsf{Pres(h,P)} \vee \mathsf{Pres(k,P)})\\
&\wedge\ ((\mathsf{x} = \mathsf{and(h,k)} \Rightarrow (\mathsf{Pres(h,P)} \wedge \mathsf{Pres(k,P)})\\
&\wedge\ ((\mathsf{x} = \mathsf{or(h,k)} \Rightarrow (\mathsf{Pres(h,P)} \vee \mathsf{Pres(k,P)})\,]\\
&\Rightarrow \mathsf{Pres(x,P)}
\end{aligned}
$$

and $\neg\mathsf{Pres}(\epsilon,\mathsf{P})$. If we can not derive $\mathsf{Pres(x,P)}$ for some x, it follows $\neg\mathsf{Pres(x,P)}$.

Theorem 1. *If it is possible to derive Pres(x,P) (conversely ¬Pres(x,P)) then* $\psi(P) \nvdash \mathsf{knows}(x)$ *(*$\psi(P) \vdash \mathsf{knows}(x)$*).*

Proof idea In case $\neg\mathsf{Pres}(\epsilon,\mathsf{P})$ since $\mathsf{knows}(\epsilon) \in \bar{\psi}(P)$ for all P. If $\forall \mathsf{F}_i \in \bar{\psi}(P)$ $\mathsf{x} \hat{\notin} J_i$ that means that there is no formula in $\bar{\psi}(P)$ containing x in a conclusive position, and therefore there is no way to derive $\mathsf{knows(x)}$ from the structural formulas. Now assume it is possible to derive $\mathsf{Pres(x,P)}$. We have already covered the base cases so we can assume that $\psi(P) \nvdash \mathsf{knows}(y)$ for all the $\mathsf{Pres(y,P)}$ $\mathsf{y} \neq \mathsf{x}$ needed in the precondition. Since in this formulas all the cases where we could apply the Structural Formulas are covered, it is impossible to derive $\mathsf{knows(x)}$. The case $\neg\mathsf{Pres(x,P)}$ is similar. □

Notice that the converse does not hold, that is $\psi(P) \nvdash \mathsf{knows}(x)$ does not mean we can derive $\mathsf{Pres(x,P)}$, because for some pathological cases we will have an infinite loop, for example for $\bar{\psi}(P) = \mathsf{knows(x)} \Rightarrow \mathsf{knows(x)}$. It is although easy to detect and avoid this loops in a machine implementation of the preservation predicate by running an initial check on the formulas. This makes the verification of the $\mathsf{Pres(x,P)}$ predicate sound and complete with respect to the First Order Logic embedding of the process programs.

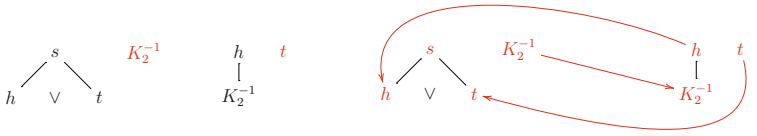


Fig. 2. Processes P and P' before and after composition

As we derive Pres(s,P) for some symbol s and formulas P, we can build a *derivation tree* consisting of the symbols we need to consider to be able to conclude the preservation status of s. If we generate and store the derivation tree for every symbol x appearing in a process P in a relevant position (that is $x \hat{\in} J_i$ for some i), then we can decide whether the composition with process P' will preserve the secrecy of any given symbol if we also have the derivation trees for P'. Consider for example $P = (\{s\}_{h::t}, K_2^{-1})$ and $P' = (\{\mathsf{h}\}_{\mathsf{K}_2}, t)$. The symbol dependency trees of both process are depicted in Fig. 2 (the symbols in red are the ones which secrecy is compromised). Clearly both processes preserve separately the secrecy of s. To see if the composition also does, we update the information on the tree of s by checking whether the truth values of h and t are altered by the composition as depicted in Fig. 2.

4 An Insecure Variant of the TLS Protocol

As an example we apply our approach to a variant of TLS [2] (not the version of TLS in current use) that does not preserve secrecy as a composition of the client C, the server S and the authority CA . We have that the predicate for C and S after the programs are translated to F.O.L are (for details on the translation see [10]) :

$$\psi(\mathsf{C}) = \mathsf{knows}(N_C :: K_C ::\mathcal{S}ign_{\mathsf{K_C}^{-1}}\{\mathsf{C} :: \mathsf{K_C}\}) \wedge (\mathsf{knows}(s_2) \wedge \mathsf{knows}(s_3) \Rightarrow \mathsf{knows}(\{m\}_y))$$
$$\psi(\mathsf{S}) = \mathsf{knows}(c_1) \wedge \mathsf{knows}(c_2) \wedge \mathsf{knows}(c_3) \Rightarrow \mathsf{knows}(N_S :: \{\ \mathcal{S}ign_{\mathsf{K_S}^{-1}}\{\mathsf{k_{CS}} :: \mathsf{c}_1\}\ \}_{c_2})$$

where $\{s_3\}_{K_{CA}} = S :: x \wedge \{Dec_{K_C^{-1}}(s_2)\}_x = y :: N_C$ and $\{c_3\}_{c_2} = C :: c_2$ where $\mathsf{key}(c_2), \mathsf{key}(x)$ and $\mathsf{key}(y)$. The set of keys is $\mathsf{Keys} = \{K_A, K_A^{-1}, k_{CS}, k_A, K_C, K_C^{-1}, K_S, K_S^{-1}, K_{CA}, K_{CA}^{-1}\}$ where k_{CS} and k_A are symmetric keys. The nonces are $\mathsf{Nonces} = \{N_C, N_S, N_A\}$. We assume that the authority CA has already distributed certificates to all parties and that the adversary is in possession of this information: $\mathsf{knows}(K_{CA}) \wedge \mathsf{knows}(\mathcal{S}ign_{\mathsf{K_{CA}}^{-1}}\{\mathsf{S} :: \mathsf{K_S}\}) \wedge \mathsf{knows}(\mathcal{S}ign_{\mathsf{K_{CA}}^{-1}}\{\mathsf{A} :: \mathsf{K_A}\})$.

We further assume that an adversary posses a key pair $\mathsf{knows}(K_A) \wedge \mathsf{knows}(K_A^{-1})$. Now we show that $\mathsf{C} \otimes \mathsf{S}$ does not preserve the secrecy of m although C and S separately do. First of all, in order to be able to apply our approach and generate the dependency tree, we have to solve the constraints for all the processes involved. So we have:

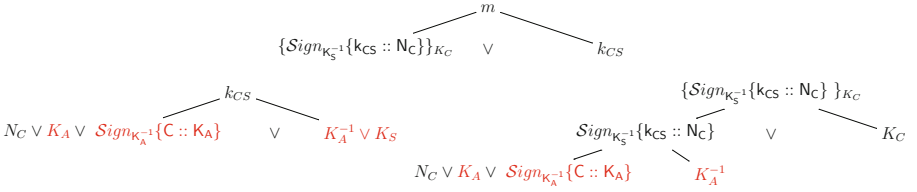


Fig. 3. Partial trees for m in C and for k_{CS} and $\{ Sign_{K_S^{-1}}\{k_{CS} :: N_C\} \}_{K_C}$ in S

$$\bar{\psi}(C) = \mathsf{knows}(N_C :: K_C :: Sign_{K_C^{-1}}\{C :: K_C\})$$
$$\wedge (\mathsf{knows}(\{ Sign_{x^{-1}}\{y :: N_C\} \}_{K_C}) \wedge \mathsf{knows}(Sign_{K_{CA}^{-1}}\{S :: x\}) \Rightarrow \mathsf{knows}(\{m\}_y))$$

where $x \in \{K_C, K_S, K_A\}$ (the public keys) and $y \in \{k_A, k_{CS}\}$ (the symmetric keys). We do not explicit the whole dependency tree for C but we note that the secrecy of m is preserved because: if $y = k_{CS}$ the adversary does not have knowledge of k_{CS}; if $y = k_A$ the adversary would need knowledge of $Sign_{x^{-1}}\{k_A :: N_C\}$ and $Sign_{K_{CA}^{-1}}\{S :: x\}$ for some x. Since he only knows $Sign_{K_{CA}^{-1}}\{S :: K_S\}$ then $x = K_S$. In that case to gain knowledge of $Sign_{K_S{}^{-1}}\{k_A :: N_C\}$ he needs to posses ${K_S}^{-1}$ which he does not. In Fig. 3 we depict partially this dependency tree for the case $y = k_{CS}$, $x = K_S$. Now, the instantiated formulas for S are:

$$\bar{\psi}(S) = \mathsf{knows}(c_1) \wedge \mathsf{knows}(c_2) \wedge \mathsf{knows}(Sign_{c_2^{-1}}\{C :: c_2\})$$
$$\Rightarrow \mathsf{knows}(N_S :: \{ Sign_{K_S^{-1}}\{k_{CS} :: c_1\} \}_{c_2})$$

with $c_1 \in \{N_S, N_C, N_A\}$, $c_2 \in \{K_C, K_S, K_A\}$. The secrecy of m is preserved in S simply because m is not a subterm of any formula in S.

To see why the composition fails to preserve secrecy, we illustrate (partially) the dependency trees of k_{CS} and $\{ Sign_{K_S^{-1}}\{k_{CS} :: c_1\} \}_{c_2}$ in case $c_1 = N_C$ and $c_2 = K_A$ in Fig. 3. In fact, since C leaks N_C and K_C, k_{CS} turns to be not secret after composition in the tree of S. This also modifies the secrecy status of $\{ Sign_{K_S^{-1}}\{k_{CS} :: N_C\} \}_{K_C}$ which results in a secrecy violation for m after updating the tree of C. We have performed a similar analysis for a fix to this protocol proposed in [10] where the composition preserves secrecy but for space reasons we do not explicit the details here.

5 Validation and Efficiency

We have implemented our approach as an extension to the UMLsec tool support [1]. That is, we can extract the protocol specification from a sequence diagram using the DSL described in Sect. 2 and translate it to First Order Logic. Since by construction each guard accepts only finitely many messages (depending on

[1] `http://www-jj.cs.tu-dortmund.de/jj/umlsec/`

Table 1. Execution times of our experiment

# Messages	# Compositions	Generation trees (ms)	Composition (ms)
11	5	3660	47
21	10	6214	88
31	15	9323	114
51	25	15406	198
101	50	31730	401
501	250	182771	1948
1001	500	375474	3963

the set of keys and nonces), we can build finite dependency trees for all relevant symbols by means of a properly generated prolog program.

Reasoning about composition amounts then to join the trees from two processes. Therefore, we can at least avoid to recompute the constraint solving for the single processes. We have conducted experiments to measure the time of the composition, and compare it to the overall process of constraint-solving and prolog generation as depicted in Table 1. The first column contains the number of messages for a single session of the composition and the second column corresponds to the number of composed processes. The third column is the time in ms. needed to extract the FOL formulas from the UML diagram and generate the derivation trees. The last column is the time needed for deciding the composition given the single derivation trees. In other words, if we would have a repository of 500 processes that by themselves are secrecy preserving, and we would like to check whether the composition of any 5 of them is also secrecy preserving, it would be highly desirable if we could use the existing results as opposed to re-verify from scratch every time.

6 Related Work

Overviews of applications of formal methods to security protocols can be found for example in [1,12], some examples in [11,13]. The question of protocol composition has been studied by different authors. More prominently, Datta, Mitchell et al. [6] have defined the PCL (Protocol Composition Logic), aimed at the verification of security protocol by re-using proofs of sub-protocols using a Hoare-like logic, focusing on authenticity. Guttmann [7] gives results about protocol composition at a lower abstraction level, considering unstructured 'blank slots' and compound keys that result from hashes of other messages. Jürjens [9] has explored the question of composability aiming at given sufficient conditions under which composition holds. Stoller [14] has computed bounds of parallel executions that could compromise the authenticity of protocols. These approaches aim at giving at a collection of theorems that if satisfied by two protocols in a composition, ensure a given property. One must show (by using a theorem prover, or by hand) that some properties are satisfied by both protocols like *disjointness* in [8]. Our approach differs from this assume/guarantee reasoning in that we efficiently check whether the composition harms secrecy given pre-computed 'proof

artifacts': the dependency trees. In other words, we give accurate results about compositions (that are equivalent to re-verification), by amortizing the cost of verification at an initial phase.

7 Conclusions

The problem of compositionality is of particular importance for software development when the security of reusable components has been established, since guarantees about the composition are needed. The decision procedure should also scale efficiently to be of practical use, and most of all, sound. We have shown that our procedure is sound and complete with respect to previous work on First Order Logic protocol verification. This comes at the price of an initial verification of the single components that considers all the possible acceptable messages. Nevertheless, this is compensated when it comes to decide compositionality with an arbitrary process for which the same process has also taken place, since this can be done very efficiently, as we have empirically tested. There are different ways in which this work could be further extended. On the one hand, one can further explore the efficiency of the approach, for example by formally deriving its complexity . On the other hand, one could extend the approach to cope with the preservation of other security properties like authenticity.

References

1. Abadi, M.: Security protocols and their properties. In: Bauer, F., Steinbrüggen, R. (eds.) 20th International Summer School on Foundations of Secure Computation, Marktoberdorf, Germany, pp. 39–60. IOS Press, Amsterdam (2000)
2. Apostolopoulos, G., Peris, V., Saha, D.: Transport layer security: How much does it really cost? In: Proceedings of the IEEE Infocom, pp. 717–725 (1999)
3. Armando, A., Carbone, R., Compagna, L., Cuéllar, J., Tobarra, M.L.: Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps. In: Shmatikov, V. (ed.) FMSE, pp. 1–10. ACM (2008)
4. Broy, M.: A logical basis for component-based systems engineering. In: Calculational System Design. IOS Press (1999)
5. Clarke, E.M., Long, D.E., Mcmillan, K.L.: Compositional model checking. In: Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS 1989). IEEE Computer Society (1989)
6. Datta, A., Derek, A., Mitchell, J.C., Roy, A.: Protocol composition logic (pcl). Electronic Notes in Theoretical Computer Science 172(0), 311–358 (2007); Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin
7. Guttman, J.D.: Cryptographic Protocol Composition via the Authentication Tests. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 303–317. Springer, Heidelberg (2009)
8. Guttman, J.D., Javier, F., Fábrega, F.J.T.: Protocol independence through disjoint encryption. In: Proceedings 13th Computer Security Foundations Workshop, pp. 24–34. IEEE Computer Society Press (2000)
9. Jürjens, J.: Composability of Secrecy. In: Gorodetski, V.I., Skormin, V.A., Popyack, L.J. (eds.) MMM-ACNS 2001. LNCS, vol. 2052, pp. 28–38. Springer, Heidelberg (2001)

10. Jürjens, J.: A domain-specific language for cryptographic protocols based on streams. J. Log. Algebr. Program. 78(2), 54–73 (2009)
11. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. Software Concepts and Tools 17(3), 93–102 (1996)
12. Meadows, C.: Open issues in formal methods for cryptographic protocol analysis. In: DARPA Information Survivability Conference and Exposition (DISCEX 2000), pp. 237–250. IEEE Computer Society (2000)
13. Paulson, L.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
14. Stoller, S.D.: A bound on attacks on authentication protocols. In: Proc. of the 2nd IFIP International Conference on Theoretical Computer Science: Foundations of Information Technology in the Era of Network and Mobile Computing (2001)

Plagiarizing Smartphone Applications: Attack Strategies and Defense Techniques

Rahul Potharaju, Andrew Newell, Cristina Nita-Rotaru, and Xiangyu Zhang

Department of Computer Science, Purdue University, IN, USA
{rpothara,newella,crisn,xyzhang}@cs.purdue.edu

Abstract. In this paper, we show how an attacker can launch malware onto a large number of smartphone users by plagiarizing Android applications and by using elements of social engineering to increase infection rate. Our analysis of a dataset of 158,000 smartphone applications *meta-information* indicates that 29.4% of the applications are more likely to be plagiarized. We propose three detection schemes that rely on syntactic fingerprinting to detect plagiarized applications under different levels of obfuscation used by the attacker. Our analysis of 7,600 smartphone application *binaries* shows that our schemes detect all instances of plagiarism from a set of real-world malware incidents with 0.5% false positives and scale to millions of applications using only commodity servers.

1 Introduction

Smartphone applications repositories have been growing at a high rate with support from hundreds of thousands of developers. AppStore [1] contains more than half-a-million applications, and Android Market [2] has just crossed the two hundred thousand mark. The two repositories use different procedures for accepting an application. Apple's AppStore accepts only applications that have been tested for potential vulnerabilities by Apple's test engineers. Android Market accepts applications without subjecting them to any code review or inspection.

The approach of being open allows the Android Market to make applications immediately available to users. However, this also makes it an easy target for attacks where plagiarized applications are used by an attacker as means to launch malware or gain personal profits. First, an attacker can easily reverse engineer applications using existing tools. Second, an attacker can easily manipulate any arbitrary application from the market and re-pack it under his name. Third, the attacker can leverage the centralized nature of the market, dashboard features that make applications immediately available to users, and social engineering (*e.g.*, using catchy titles) to push malicious applications to a large number of victims. Thus, an attacker can easily download a popular application, insert malicious code into the application, and resubmit the malicious version back into the market without being detected. We refer to this class of actions as *plagiarism* and to the modified application as a *plagiarized* application.

While several plagiarizing incidents [3, 4] targeting applications from the Android Market have been reported, there are currently no fool-proof preventative

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 106–120, 2012.

mechanisms in place to detect plagiarism in open markets. Signature-based malware detection tools such as Lookout Security [5], Norton Mobile Security [6] and BitDefender Mobile Security [7] detect applications that contain malware. However, they do not detect plagiarized applications that use legitimate permissions, users will still be infected before an attack signature is learned, and the number of infected users can be large due to the centralized nature of markets. Information leakage detection techniques based on taint analysis [8], access control policies [9], and kernel modifications [10] protect against stealing critical user information. However, such schemes require significant user input in order to achieve high accuracy. In addition, most of these techniques work on the client-side and often demand heavy resources that lead to battery drain.

Contributions: In this paper we propose a solution that detects plagiarized applications and prevents their acceptance into the market. As a result, our approach raises the bar for the attackers, forcing them to create original applications to host their malware. Our solution is designed to be applied on the market side and is complementary to client-side defense techniques such as malware detection and information leakage prevention. Our contributions are:

– We analyze the *meta-information* of 158,000 applications from the Android Market and find that 29.4% of the applications are more likely to be plagiarized because of the permission rights they provide to an attacker. We also found that an attacker can use category, total number of downloads, and published weekday to increase the first-day number of downloads for the plagiarized application.
– We propose three schemes *Symbol-Coverage*, *AST-Distance*, and *AST-Coverage* that rely on symbol tables and method-level Abstract Syntactic Tree (AST) fingerprints to detect plagiarized applications under different levels of obfuscation used by the attacker. Our analysis of 7,600 smartphone application *binaries* shows that our schemes can detect all instances of plagiarism from a set of real-world malware incidents while having only a 0.5% false positive rate.
– We show that our detection schemes scale to millions of applications using commodity servers, i.e. it takes 2-8 seconds to reverse-engineer and fingerprint an application and 0.8-1.4 seconds to retrieve plagiarized versions of an application.

The rest of the paper is organized as follows. Section 2 describes our system and threat model. Section 3 presents more details about how an attacker can plagiarize applications. Section 4 gives an overview of our defense solution, and Section 5 presents its evaluation. Section 6 discusses related work and Section 7 concludes our paper.

2 System and Threat Model

In this section we first give more details about the application submission process in the Android Market. Then, we describe the resources and mechanisms available to an attacker to plagiarize applications.

2.1 Android Development Process

Android [11] is an open source software stack for mobile devices that includes an operating system, an application framework, and core applications. The operating system relies on a kernel derived from the Linux kernel. The application framework consists of the Dalvik Virtual Machine [12] that runs .dex files. Applications are written in Java using the Android SDK [13], compiled into .dex (Dalvik Executable), and packaged into .apk (Android package) archives for installation.

To submit an application, a developer must have a publisher account obtained by paying a nominal one-time fee of $25.00 USD and by having a valid Gmail account. He can then upload the application, optionally setting the price for a paid application. Each binary is accompanied by *meta-information* which includes: name, rating, date updated, version, category, number of installs, size, price, etc. The Android Market requires that all applications are digitally signed, with the public key made available as a *digital certificate*. As an optional step, developers can obfuscate their binaries through a tool called ProGuard [14] which removes any debugging information and renames the identifiers (*e.g.*, class, method, variable names) while maintaining the same functionality.

2.2 Threat Model

The attacker collects sensitive information stored on smartphone devices or obtains monetary profit by exploiting users. Examples of sensitive information include usage information, IMEI numbers, and GPS location. Ways to obtain monetary profit include redirecting ad-revenue or forcing smartphones to call a toll number that is owned by the attacker. We assume that the attacker can obtain a developer account for the Android Market without being traced by the market administrators.

We consider attacks that exploit the popularity and permissions already available in existing applications. Thus, an attacker chooses an existing application that already has permissions that can be exploited, modifies it according to his needs, and uploads the modified version to an open market. The modified version not only has the same functionality as the original application but also includes malicious code to collect sensitive information or obtain monetary profit.

Android Market requires developers to use digital certificates to attest their identity. However, it does not require a trusted *certificate authority* (CA) to sign the certificates. Thus, digital certificates will not prevent an attacker from plagiarizing an application. Establishing trust of developers through CAs would hinder the openness of the markets. If CAs were to enforce high requirements to deter malicious developers then many legitimate developers will also be excluded from being trusted by the CA.

2.3 Obfuscation Model

We assume that an attacker can apply the following obfuscation techniques:

– **Level-1:** Symbol table is obfuscated such that methods, classes, variables, and other identifiers are all changed. The tool ProGuard provided by Google allows only Level-1 obfuscation. To the best of our knowledge, this is the only kind of obfuscation that is being applied in the real-world for mobile applications.
– **Level-2:** α random methods with no functionality are added. This level of obfuscation has not been seen yet in real smartphone application repositories, we nonetheless consider it as attackers can leverage it without substantial efforts.

While more advanced obfuscations have been proposed in the research community [15, 16], their applicability to mobile applications remains unknown due to the specific byte-code format and the tight resource and energy constraints.

3 Plagiarizing Applications

The goal of an attacker plagiarizing an application is to take advantage of its popularity and collect sensitive information or obtain monetary profit. We first describe the attack payloads that the attacker can embed inside the plagiarized version of the application and then describe strategies that an attacker can leverage to increase the overall infection count.

3.1 Plagiarism Mechanisms and Payload

An attacker resorting to plagiarism first downloads an application and obtains the .dex files of the application. The attacker then uses one of the two approaches: (1) *direct byte-code insertion*, (2) *assisted byte-code insertion*. In *direct byte-code insertion*, the attacker writes his own bytecode into the application byte-code and re-packages it into an APK package. This approach usually requires heavy expertise in writing Dalvik specific byte-code but has the advantage that it can evade detection of certain static analysis tools that rely on application-level source code heuristics. In *assisted byte-code insertion*, the attacker first writes his malicious code as part of a stub application and compiles it using the Android SDK. The attacker reverse engineers his own APK to obtain the .dex files and extracts relevant portions of the byte-code for insertion into the original application and then re-packages it into a separate APK package. Possible payloads for the plagiarized application include:

– **Privacy Exploitation:** If the original application requests for permissions to obtain the GPS coordinates of the user (ACCESS_FINE_LOCATION) or to read the user's contact data (READ_CONTACTS), the attacker can insert a code snippet that obtains this information and sends it to back to the attacker.
– **Monetary Exploitation:** If the original application requests for permissions to send SMS messages (SEND_SMS) or to allow the application to initiate a phone call without going through the *Dialer* user interface that forces users to confirm the call being placed (CALL_PHONE).

Table 1. Attack payload collected from a dataset of 158,000 applications. A 29.4% of the applications are susceptible to at least one payload listed in the table.

Permission	Permission	# of App.	Possible Attack Payload
INTERNET	ACCESS_COARSE_LOCATION	28,759	retrieve location through WiFi
INTERNET	ACCESS_FINE_LOCATION	27,258	retrieve location through GPS
INTERNET	READ_CONTACTS	11,870	retrieve user's contacts
INTERNET	CAMERA	6,936	record/retrieve images from camera
ANY	SEND_SMS	7,652	send SMS messages
ANY	CALL_PHONE	8,074	place phone calls
ANY	BRICK	11	permanently disable the device
ANY	INSTALL_PACKAGES	430	install arbitrary packages

To gain insights into how many applications are vulnerable to attacks we examined the *meta-information* (descriptive information about an application) of 158,000 applications that we collected from the Android Market. We count the number of applications from our dataset that request permissions that can be exploited for various types of attacks. The results presented in Table 1 show that many applications require permissions that can be leveraged by attackers. For instance, an attacker interested in sending SMS from legitimate phones can choose applications to plagiarize out of a set of 7,652. We estimate that 29.4% of the applications from our dataset are susceptible to at least one attack payload listed in Table 1. Our findings are consistent with results in [17] which showed that many Android applications violate the principle of least privilege and request more permissions than needed.

3.2 Improving Infection Count

An attacker can increase the infection count of a plagiarized application by carefully choosing what applications to plagiarize and what day of the week to perform the attack. A good strategy for an attacker is to target applications that can rapidly become popular the first day. The optimal strategy we found from our dataset of applications is to plagiarize an *Arcade & Action* game that has more than 250,000 downloads and release this plagiarized application on Sunday.

We extract from our dataset: *(i) the category of the application*, *(ii) the number of downloads the first day the application was submitted*, *(iii) the current download count*, and *(iv) the day of the week published*. From our dataset of 158,000 applications, a subset of 36,000 applications have sufficient information to extract these four pieces of information. We cannot obtain exact download counts each day due to the way Android Market reports download counts in ranges, so we simply assume the average of the upper and lower bound to be the number of downloads (100-500 downloads is interpreted as 300 downloads).

We divide the 36 categories on the Android Market into 4 groups: {*Personal, Games, Media, Leisure*}. Figure 1(a) shows a TreeMap [18] where each category is represented by a rectangle with a size corresponding to number of applications and color corresponding to number of first day downloads. Clearly, some categories have a much higher number of initial first day downloads compared to other categories, e.g., the category *Arcade & Actions* achieves nearly double

the downloads of the next highest category, *Casual Gaming*. Also, *Games* has high first day download counts compared to the other groups. An attacker can choose specific categories to achieve higher initial downloads since the category choice affects initial download count.

We found that applications which have a high download count went viral on their first day. Figure 1(b) plots the average download count on the first day for each download range listed currently in the market. The applications that have reached greater than 250,000 downloads are the most popular applications, receiving four times the initial download count as opposed to the next highest bracket of 50,000-250,000 downloads.

An attacker can also try to choose the day of the week to load the application to increase the probability that the plagiarized application will become viral on its first day. Figure 1(c) shows the average download count on the first day of an application given the day of the week it was uploaded to the market. We have between 3,996 and 5,750 samples for each day of the week, so we can confidently conclude that an attacker can expect higher first day download counts by roughly 20% by selecting an appropriate day to execute the attack (e.g., weekends).

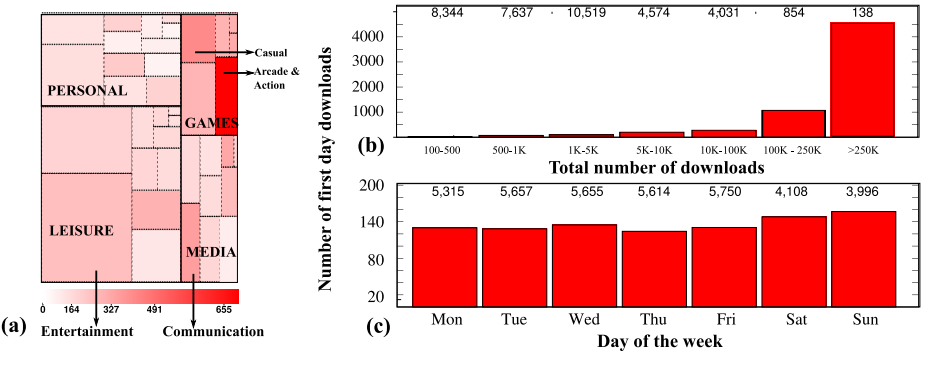


Fig. 1. Choosing an application based on: (a) **category:** the area of each rectangle maps to the number of applications and the color corresponds to first day download counts, (b) **total downloads:** number of applications is on top of each bar, (c) **application publish weekday:** number of applications is on top of each bar

4 Detecting Plagiarized Applications

We first describe the extraction of symbol tables and application fingerprints from binary code. We then describe how we use the extracted information to detect potentially obfuscated plagiarized applications with three schemes *Symbol-Coverage*, *AST-Distance*, and *AST-Coverage*. See Appendix for more detailed descriptions of the algorithms presented in this section.

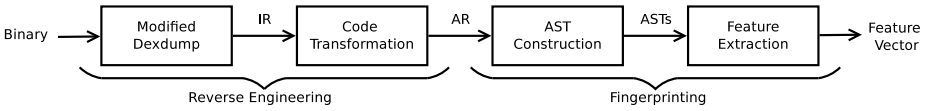


Fig. 2. Reverse Engineering and Fingerprinting Procedure

4.1 Reverse Engineering and Fingerprinting Android Applications

Figure 2 shows how we obtain application fingerprints by starting with an application binary. We first reverse engineer it to obtain an Abstract Representation (AR) of the corresponding source code. Then we use the AR to construct method-level ASTs and extract feature vectors that represent the fingerprints.

Reverse Engineering. Android has a built-in open source disassembler called *dexdump*. Given a .dex file, *dexdump* creates a dump file (IR code) of all the classes and methods. We modified five functions: *dumpClass*, *dumpMethod*, *dumpCode*, *dumpSField* and *dumpIField*[1] to capture relevant method-related information. *Code Transformation* is then used to obtain an AR based on pre-defined rules for each statement type. A rule is a regular expression that captures the various parts of the statement (such as method name, variable names, number of arguments etc.). Once this information is captured, all variables are named alike (e.g. x,y,z are all replaced with ARG). The advantage of using AR is that even if the code is obfuscated through means such as variable re-naming, the syntactic structure is preserved and any fingerprint generated out of this transformed code will be the same for both obfuscated and non-obfuscated code.

Consider an excerpt from the IR representation of a larger method shown in Figure 3(a). The example shows a method invocation that starts by defining the type of invocation (*virtual*, *static*, *direct*, *super* or *interface*), then the registers that are to be checked for the arguments, and finally the signature of the method that is being invoked. We fingerprint the two most common types of invocations: *invoke-direct*, which is used to invoke an instance method that is by nature non-overridable (either a *private* method or a *constructor*) and *invoke-virtual*, which is used to invoke a normal virtual method (a method that is not *static* or *final*, and is not a *constructor*). For instance, the first invocation is made to the *init* method of the super class *android/app/Activity* and the second to the *findViewById* method of the Fuzzer class from our sample application. The *Code Transformation* module rewrites this set of statements as seen in Figure 3(b).

Extracting Fingerprints. Once we have an abstract representation of the IR code, we perform *AST Construction* which generates a special type of Abstract Syntax Tree called the *method-level Abstract Syntax Tree*, for each method in the byte-code. A method-level AST captures the following information: (i) Number of arguments that the current method accepts, (ii) Other methods invoked by

[1] While our tool is home-brewed and not ready for a production usage yet, we do acknowledge the presence of several other promising tools [19, 20] that came out recently.

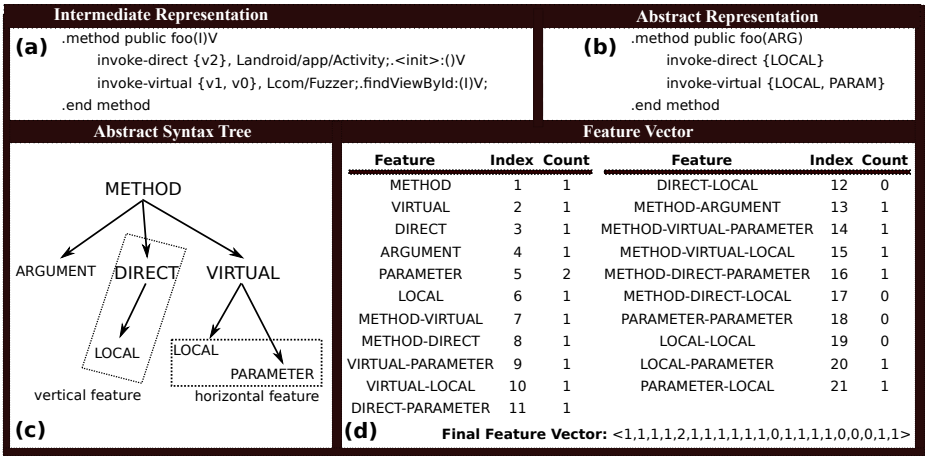


Fig. 3. Example of Fingerprinting an Applications

the current method with the invocation type, direct or virtual. Other syntactic artifacts inside a method, such as assignment and conditional statements, are precluded. For instance, consider the AST given in Figure 3(c). The root node is always the METHOD label and subsequent nodes ARGUMENT, DIRECT, and VIRTUAL denote the method has an argument and two function invocations, one with the direct type and the other the virtual type. The children of the second-level nodes then, are the registers that these methods are utilizing.

We then use the ASTs to extract a feature vector that represents the application fingerprint. We adapt the structural feature extraction method in [21] to work with the method-level ASTs that we constructed. For a given AST, we record two types of patterns of structural information: *(l,m)-leaf* and *n-path*. A *(l,m)-leaf* is a pair of leaf nodes having a common parent and is used to capture the *horizontal paths* in an AST. In Figure 3(d), {*LOCAL-PARAMETER*} is one such *horizontal path* in the AST. Note that an AST does not need to have any *(l,m)-leaf* pairs. This can happen when the method does not have any arguments or does not contain other methods that have arguments. An *n-path* is a directed path of n nodes, *i.e.*, a sequence of n nodes in which any two consecutive nodes are connected by a directed edge in the tree. *DIRECT-LOCAL* is one such *vertical path* in the AST. Other *vertical paths* are: {*METHOD, METHOD-ARGUMENT, ...* }. A special case is *1-path* which contains only one node.

The feature vector in our case is the occurrence count of all the *horizontal* and *vertical* paths extracted from the AST. To derive this for a given AST, we first allocate a vector filled with 0's and compute all *(l,m)-leaf paths* and *n-paths*. For each individual path, we get the path's identifier from a global lookup table that holds all possible paths. This identifier is used to determine which of the dimensions in the vector needs to be incremented. A feature vector for an entire application can be generated in the same way by calculating the feature vector over a graph that is a forest where each tree represents a method of the application. Figure 3 shows an example of this procedure.

In terms of performance, feature extraction is at most quadratic with respect to the total number of nodes, n. For vertical feature extraction, the paths of an AST are limited to a constant length, three, so the number of total paths to traverse is $O(n)$. For horizontal feature extraction, the number of pairs to iterate over and count in the worst case, a single DIRECT or VIRTUAL node has $O(n)$ children (this case is highly unlikely), corresponds to $O(n^2)$ pairs.

4.2 Detection Techniques

We design three defense schemes: Symbol-Coverage for non-obfuscated applications, AST-Distance for applications with Level-1 obfuscation(observed in the real-world), and AST-Coverage for applications with Level-2 obfuscation (not observed in real-world to the best of our knowledge but possible).

Symbol-Coverage. If an attacker does not obfuscate, the symbol table information is available from the application byte-code. We consider the coverage of symbol table information for every application $A_1, A_2, ..., A_n$ by an uploaded application A. If some application A_i is covered highly by A, then we consider A a plagiarized version of A_i. The coverage of an application A_i by A is computed as the number of classes and methods in A_i that also exist in A divided by the total number of classes and methods in A_i. We only consider methods as matching if they belong to classes that match. The application A_i with the highest coverage is reported if the coverage exceeds some threshold.

AST-Distance. If an attacker obfuscates symbol table information (Level-1), we use a defense based on feature vectors derived from method-level ASTs (see Section 4.1). We use Euclidean distance due to its high accuracy in preliminary results where we tested various distance metrics. Let A_i be the application with a feature vector that has the smallest distance to the feature vector of the application A, then, A_i is reported if this distance is smaller than some threshold.

AST-Coverage. In our final algorithm, we aim to accurately detect plagiarism where applications were obfuscated with Level-2 obfuscation. We combine the AST based feature vectors with the coverage approach. Specifically, once the ASTs of each method of the applications in the market $A_1, A_2, ..., A_n$ and the uploaded application A are transformed into feature vectors, feature vectors of each method of $A_1, A_2, ..., A_n$ that are close to a feature vector of A are marked as covered. The maximally covered application A_i is reported if the coverage is greater than some threshold value.

5 Defense Evaluation

We evaluate the detection accuracy of our schemes using a dataset of 7,600 application binaries that we refer to as the Pseudo-Market.

5.1 Real-World Plagiarism Detection

We analyzed 13 instances[2] of the HongTouTou [22] malware which plagiarized highly-popular legitimate applications and relied on social engineering. The

[2] We thank Tim Strazzere of Lookout Security for sharing the malware samples with us.

Table 2. Real-World Plagiarisms: List of plagiarized instances of legitimate applications that have occurred in the Android Market. We show the coverage of AST-Coverage with (+) and without (-) the legitimate application in the market.

Application		Characteristics		AST-Coverage	
Legitimate Title	**Malware Title**	**Price**	**Downloads**	+	-
yxPlayer	Flash Player	Free	≥250,000	1.000	0.100
Steamy Window	Screen Mist	Free	≥250,000	1.000	0.118
Hello Kitty LWP Lite	HelloKitty Livewallpaper	Free	≥250,000	1.000	0.053
Wave Live Wallpaper	Wave Livewallpaper	Free	50,000-250,000	1.000	0.077
AndroMax	Multi-Keyboard Shortcuts	Free	50,000-250,000	1.000	0.100
Shamrock Live Wallpaper	Clover Wallpapers	Free	50,000-250,000	1.000	0.053
City at Night	NightCity	$0.99	50,000-250,000	1.000	0.077
Hi-Hiker Pro	Hiker	Free	50,000-250,000	1.000	0.100
Dandelion Livewallpaper	TAT-LWP-Mod-Dandelion	Free	10,000-50,000	1.000	0.006
Robo Defense	Robo_Defense	$1.88	1,000-5,000	1.000	0.105
Sense Live Wallpaper Pro	Beautiful Live Wallpaper	$1.88	1,000-5,000	1.000	0.333
Yo Handcar: Off the Rails	yohandcar	Free	1,000-5,000	0.992	0.182
Roller Rev 99	Crazy Roller Coaster	$2.99	100-500	1.000	0.182
Stickers Off	Miniv	Free	100-500	1.000	0.100
Snow Flurry Live Wallpaper	LiveWinter	$0.99	100-500	1.000	0.043

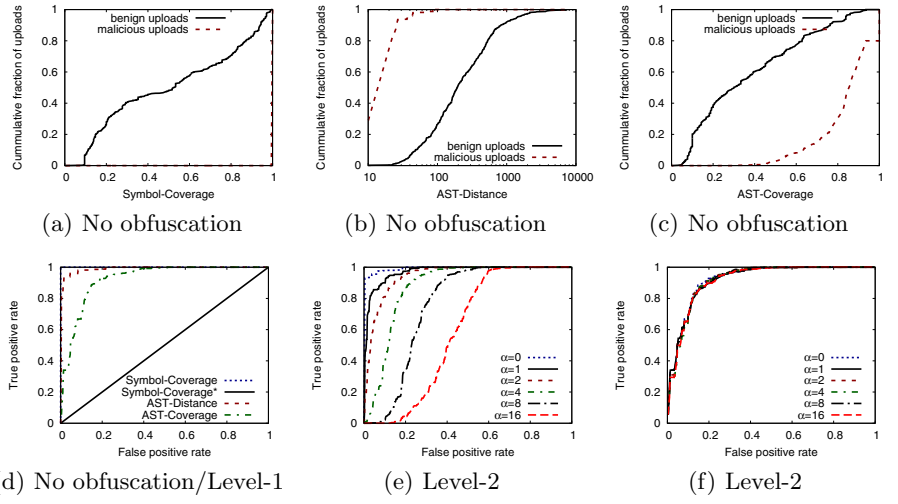
malware sends the device's IMEI and IMSI numbers to a remote host [22] and receives a set of search engine URLs and keywords that are then used to emulate keyword searches and clicks committing various types of click-fraud.

Table 2 presents results for the AST-Coverage algorithm for HongTouTou instances of plagiarism when compared with applications from our Pseudo-Market. We found that all but one application resulted in full AST-Coverage. A full coverage means that all the methods in an original application are covered by the given plagiarized version. The only exception was "*yohandcar*" (malware) which covered 99.2% of the methods of "*Yo Handcar: Off the Rails*" (goodware). We manually verified the reverse-engineered code of both applications to confirm that the malware was indeed mimicking the functionality of the legitimate application. We also found that this malicious instance of HongTouTou not only adds new methods to the original version, but also changes some existing methods in the original version, leading to the slight mismatch.

To show that our detection does not wrongly accuse applications of plagiarism, we removed the legitimate versions of the malware samples from our Pseudo Market and re-ran the AST-Coverage detection. As seen in Table 2, it correctly reported a low coverage of the malware samples with the other applications.

5.2 Accuracy

We perform 500 benign and 500 plagiarized uploads. For a benign upload, we select a random application, remove it from the Pseudo-Market, and then upload it back. For a plagiarized upload, we select a random application, insert malicious code into it, and upload the application back to the Pseudo-Market. Both the original and the plagiarized versions are in the Pseudo-Market. We model insertion of malicious code by selecting a random method from all the methods of all applications, and inserting it into the application. By not inserting specific



(a) No obfuscation (b) No obfuscation (c) No obfuscation

(d) No obfuscation/Level-1 (e) Level-2 (f) Level-2

Fig. 4. (a) CDF of largest coverage of Symbol-Coverage; (b) CDF of smallest distance of AST-Distance; (c) CDF of largest coverage of AST-Coverage; (d) ROC of the accuracy for all schemes; Symbol-Coverage changes from no obfuscation (case w/o asterisk) to Level-1 obfuscation (case with asterisk); (e) ROC of accuracy AST-Distance; (f) ROC of accuracy AST-Coverage

malicious code, but using random code, the results approximate the performance in the presence of arbitrary types of malicious code.

We show CDFs of the coverage value for Symbol-Coverage in Figure 4(a), the plagiarized uploads are not obfuscated. Symbol-Coverage distinguishes all plagiarized applications from correct applications since all malicious uploads have 1.0 coverage and no benign uploads had 1.0 coverage. Thus, there are no false positives in this case.

Figures 4(b) and 4(c) show the CDF distance and coverage for the AST-Distance and AST-Coverage, when the plagiarized uploads are not obfuscated. The two schemes cannot perfectly distinguish benign uploads from plagiarized uploads based on a single threshold value. In both cases, there is some fraction of benign uploads that overlaps with plagiarized uploads. For AST-Distance, the plagiarized upload distance to the original application is quite small (note the log-scale of the x-axis) in relation to the closest distance of some benign uploads, but there is a portion of benign applications that are close to some other benign application in terms of AST-Distance. For AST-Coverage, the plagiarized uploads have methods where the AST fingerprint is identical to many methods of various applications due to shared libraries, and the methods are not always matched to the methods of the original application that was plagiarized.

Each of our detection techniques can distinguish, with high accuracy, a malicious versus benign upload given the coverage or distance of the uploaded application to the next closest application in the market. The exact accuracy for each is shown in the ROC curve of Figure 4(d) which is created by plotting TPR and

FPR for each possible threshold value of Figures 4(a), 4(b), and 4(c). Although AST-Distance and AST-Coverage have lower accuracy when no obfuscation is used, these two schemes are more resilient to obfuscated uploads.

5.3 Obfuscation Resilience

We now evaluate the robustness of our schemes to Level-1 (replace method and class names with random names) and Level-2 (add α random methods) obfuscation. While more advanced obfuscations have been proposed in the research community [15, 16], their applicability to mobile applications remains unknown due to the specific byte-code format and the very tight resource constraints.

Figure 4(d) shows results for no obfuscation and Level-1 obfuscation. Symbol-Coverage's performance degrades substantially when Level-1 obfuscation is used, with an accuracy equivalent to guessing instead of a perfect accuracy. This is because the algorithm relies completely on values inside the symbol table which are obfuscated under Level-1 obfuscation. The accuracy of AST-Distance and AST-Coverage remain the same since they do not use any symbol table information. AST-Distance is the most effective under Level-1 obfuscation.

Figures 4(e) and 4(f) show the results of AST-Distance and AST-Coverage under Level-2 obfuscation. We increase α to show that the accuracy of AST-Distance degrades significantly while the accuracy of AST-Coverage does not change significantly. No threshold exists for AST-Distance that can distinguish between the distance of random methods added and the distance between a legitimate application and all other applications in the Pseudo-Market. AST-Coverage is robust to this problem because it relies on coverage of each application in the Pseudo-Market instead of distance to each application, so a larger size does not affect coverage of other applications as much.

5.4 Computational Feasibility

We evaluate the execution time of our schemes on a Quad-Core AMD Opteron(tm) Processor 2380 machine of 2.50 GHz with 16 GB of RAM that represents a typical commodity server used in data centers such as Amazon EC2. We use synthetic datasets with sizes of 250K, 500K, 750K, 1000K, 1250K and 1500K applications, created by randomly selected applications from our dataset of the real 7,600 binaries. For the Symbol-Coverage algorithm we store symbols in a B+ tree of a MySQL database. For AST-Distance and AST-Coverage we utilize BDD-trees [23] which is a data structure for performing k-nearest neighbor searches on high-dimensional vectors. Figure 5 shows that our schemes scales to millions of applications. The AST-Distance and AST-Coverage differ in performance because the AST-Coverage must search over many more vectors.

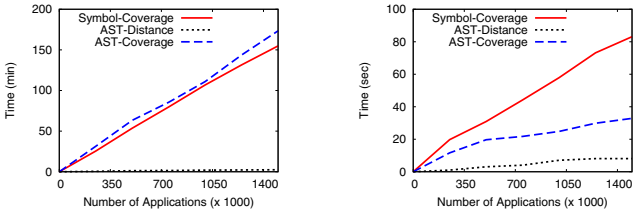


Fig. 5. (a) Preparation time for inserting and indexing the symbols inside the MySQL database, (b) Query time to fetch all symbols from the database. Notice that after indexing, the query time is less than 0.3 seconds per symbol

6 Related Work

Our work builds upon work on clone detection [24, 25] that determines the existence of duplicated code fragments in large enterprise source code bases. Deckard [24], a state-of-the-art tree-based approach, extracts characteristics vectors from parse trees by counting *q-level* binary subtree patterns. Nguyen et al. [21] improve upon this approach by efficiently capturing more structural characteristics. Compared to these techniques, ours handles Dalvik byte-code. Our feature extraction method is also different.

Our work is also related to algorithms comparing program versions, such as a program and its obfuscated version [26–30]. These algorithms perform program differencing at various levels: control flow graph level [26, 28], procedure level [27], and statement level [29, 30]. These approaches require source-code, whereas our approach works on byte-code. They are far more computationally demanding than our AST based algorithms that are effectively accurate.

From a client-side defense perspective, there has been significant work in the area of program analysis and access control to protect users against malicious applications. Enck *et al.* [8] describe a framework to detect potentially malicious applications based on permissions requested by Android applications. Nauman *et al.* [9] propose Apex, a policy enforcement framework for Android that allows a user to selectively grant permissions to applications as well as impose constraints on the usage of resources. These solutions must run on the resource-constrained mobile devices, and they rely on appropriate configuration by the user.

7 Conclusion

In this paper we focused on attacks that plagiarize popular smartphone applications to collect sensitive information or obtain monetary profit. We analyze the *meta-information* of 158,000 applications from the Android Market and find that 29.4% of the applications are more likely to be plagiarized. We proposed three schemes that rely on method-level AST fingerprints to detect plagiarized applications under different levels of obfuscation used by the attacker. Our analysis of 7,600 smartphone application *binaries* shows that our schemes detect all

instances of plagiarism from a set of real-world malware incidents with 0.5% false positives and scale to millions of applications using only commodity servers.

References

1. Kerris, N., Neumayr, T.: Apple App Store Downloads Top Two Billion (2009)
2. Chu, E.: Android Market: A User-driven Content Distribution System (2008)
3. Animal Rights Protesters use Mobile Means for their Message, http://goo.gl/An7Rp
4. Warning on Possible Android Mobile Trojans, http://goo.gl/A8Ow9
5. Lookout Anti-Virus, https://www.mylookout.com/
6. Norton Mobile Security, http://us.norton.com/mobile-security/
7. Bitdefender Mobile Security, http://m.bitdefender.com/
8. Enck, W., Gilbert, P., Chun, B., Cox, L., Jung, J., McDaniel, P., Sheth, A.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: OSDI (2010)
9. Nauman, M., Khan, S., Zhang, X.: Apex: Extending Android Permission Model with user-defined runtime constraints. In: ICCS (2010)
10. Jakobsson, M., Johansson, K.: Retroactive detection of malware with applications to mobile platforms. In: HotSec (2010)
11. Google Android, http://code.google.com/android
12. Dalvik Virtual Machine, http://www.dalvikvm.com
13. Google Android SDK, http://developer.android.com/sdk/
14. Lafortune, E., et al.: ProGuard (2004), http://proguard.sourceforge.net
15. Linn, C., Debray, S.K.: Obfuscation of executable code to improve resistance to static disassembly. In: CCS (2003)
16. Collberg, C.S., Thomborson, C.D.: Watermarking, Tamper-Proofing, and Obfuscation-Tools for Software Protection. In: IEEE TSE (2002)
17. Felt, A., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. Technical Report UCB/EECS-2011-48, University of California, Berkeley, Tech. Rep. (2011)
18. Shneiderman, B.: Treemaps for space-constrained visualization of hierarchies. In: ACM TOG (1998)
19. de-Dexer, http://dedexer.sourceforge.net
20. dex2jar, http://code.google.com/p/dex2jar/
21. Nguyen, H., Nguyen, T., Pham, N., Al-Kofahi, J., Nguyen, T.: Accurate and Efficient Structural Characteristic Feature Extraction for Clone Detection. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 440–455. Springer, Heidelberg (2009)
22. Lookout Security Blog, http://goo.gl/q9sI8
23. Arya, S., Mount, D., Netanyahu, N., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor search in fixed dimensions. JACM (1998)
24. Jiang, L., Misherghi, G., Su, Z., Glondu, S.: Deckard: Scalable and accurate tree-based detection of code clones. In: ICSE. IEEE Computer Society (2007)
25. Li, Z., Lu, S., Myagmar, S., Zhou, Y.: CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. In: IEEE TSE (2006)
26. Apiwattanapong, T., Orso, A., Harrold, M.: A Differencing Algorithm for Object-Oriented Programs. In: ASE (2004)
27. Jackson, D., Ladd, D.: Semantic Diff: A Tool for Summarizing the Effects of Modifications. In: ICSM (1994)
28. Laski, J., Szermer, W.: Identification of Program Modifications and its Applications to Software Maintenance. In: ICSM (1992)

29. Aiken, A., et al.: Moss: System for detecting software plagiarism, http://www.cs.berkeley.edu/aiken/moss.html
30. Komondoor, R., Horwitz, S.: Semantics-Preserving Procedure Extraction. In: POPL (2000)

Appendix: Feature Extraction and Detection Techniques

Algorithm 1 shows our feature extraction alogorithm which is performed on a forest of abstract syntax trees (as shown in Figure 3c). The pairs of children of VIRTUAL and DIRECT nodes are used to update the horizontal features, and a depth first traversal is used to find all paths to update vertical features.

Algorithms 2, 3, and 4 show the details of our detection techniques. Symbol Coverage, Algorithm 2, compares the coverage by the submitted application with each application existing in the repository. AST Distance, Algorithm 3, compares the feature vector distance of a submitted application with every application in the repository. AST Coverage, Algorithm 4, compares the coverage of methods by the submitted application with each application in the repository, and methods are compared by the distance between their feature vectors. In each algorithm, the closest matched algorithm in the repository is compared with a threshold to determine whether the new application is a plagiarized version.

Algorithm 1. Feature Extraction

let G be a forest of the method ASTs
for each each node v in G **do**
  traverse($v, \{v\}$)
for each VIRTUAL or DIRECT node v in G **do**
  for each each child pairs (u, w) of node v **do**
    update_horizontal_feature($\{u, w\}$)

procedure traverse(v, p) **do**
  update_vertical_feature(p)
  for each child u of node v **do**
    traverse($u, p + \{u\}$)
end procedure

Algorithm 2. Symbol-Coverage:

Initialize numbers $c_1, c_2, ..., c_n, t_1, t_2, ..., t_n$ to zero
for all $i \in \{1, 2, ..., n\}$ **do**
  $shared\_classes = Classes[A] \cap Classes[A_i]$
  $t_i := len(Classes[A_i])$
  $c_i := len(shared\_classes)$
  if $len(shared\_classes) > 0$ **then**
    for all $x \in shared\_classes$ **do**
      $shared\_methods = Methods[A][x] \cap Methods[A_i][x]$
      $t_i := t_i + len(Methods[A_i][x])$
      $c_i := c_i + len(shared\_methods)$
$j :=$ argmax$(\frac{c_i}{t_i})$
$p := \frac{c_j}{t_j}$
if $p >$ Threshold **then**
  Alarm(A_j)

Algorithm 3. AST-Distance

$\mathbf{x} :=$ Extract_AST_Feature_Vector(A)
for all $i \in \{1, 2, ..., n\}$ **do**
  $\mathbf{y} :=$ Extract_AST_Feature_Vector(A_i)
  $d_i := ||\mathbf{x} - \mathbf{y}||$
$j :=$ argmin(d_i)
$d := d_j$
if $d <$ Threshold **then**
  Alarm(A_j)

Algorithm 4. AST-Coverage

Initialize set Z to be empty
for all $i \in 1, 2, ..., n$ **do**
  for all $y \in$ Extract_Methods(A_i) **do**
    $\mathbf{y} :=$ Extract_AST_Feature_Vector(y)
    $Z := Z \cup \mathbf{y}$
for all $x \in$ Extract_Methods(A) **do**
  $\mathbf{x} :=$ Extract_AST_Feature_Vector(x)
  $Y :=$ Nearest_Neighbors($k, Z, \mathbf{x}$)
  for all $\mathbf{y} \in Y$ **do**
    $A_i :=$ Get_Application($\mathbf{y}$)
    Update_Coverage_Information($A_i, \mathbf{y}$)
for all $i \in 1, 2, ..., n$ **do**
  $c_i := Count\_Covered\_Methods(A_i)$
  $t_i := Number\_Of\_Methods(A_i)$
$j :=$ argmax$(\frac{c_i}{t_i})$
$p := \frac{c_j}{t_j}$
if $p >$ Threshold **then**
  Alarm(A_j)

Design of Adaptive Security Mechanisms for Real-Time Embedded Systems

Mehrdad Saadatmand, Antonio Cicchetti, and Mikael Sjödin

Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Västerås, Sweden
{mehrdad.saadatmand,antonio.cicchetti,mikael.sjodin}@mdh.se

Abstract. Introducing security features in a system is not free and brings along its costs and impacts. Considering this fact is essential in the design of real-time embedded systems which have limited resources. To ensure correct design of these systems, it is important to also take into account impacts of security features on other non-functional requirements, such as performance and energy consumption. Therefore, it is necessary to perform trade-off analysis among non-functional requirements to establish balance among them. In this paper, we target the timing requirements of real-time embedded systems, and introduce an approach for choosing appropriate encryption algorithms at runtime, to achieve satisfaction of timing requirements in an adaptive way, by monitoring and keeping a log of their behaviors. The approach enables the system to adopt a less or more time consuming (but presumably stronger) encryption algorithm, based on the feedback on previous executions of encryption processes. This is particularly important for systems with high degree of complexity which are hard to analyze statistically.

Keywords: Security, real-time embedded systems, runtime adaptation, trade-off.

1 Introduction

Security is gaining more and more attention in the design of embedded systems. Embedded systems are nowadays everywhere. They are used in controlling systems of power plants, vehicular systems and medical devices, as well as, mobile phones and music players. The operational environment, physical accessibility, and mobility of embedded systems make them prone to certain types of attacks which might be less relevant for ordinary computer systems, such as side channel attacks, time and power analysis to determine security keys and algorithms [1]. Also, the increasing use of embedded devices as parts of networked and interconnected devices makes them prone to new types of security issues [2].

On the other hand, introducing security in embedded systems requires careful considerations, trade-off analysis, and balance with other aspects such as performance, power consumption, and so on. This is mainly due to the resource constraints and limitations that these systems have. For example, choosing an

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 121–134, 2012.

encryption algorithm that performs heavy computations, requires lots of memory, and, as a result, consumes more energy, may impair the correct functionality of the system and violate its specified requirements. This is basically because of the fact that non-functional requirements, such as security, are not independent and cannot be considered in isolation [3]. Therefore, it is important to understand the impacts and consequences of designed security mechanisms on other aspects of the systems.

In real-time embedded systems, where timing requirements are critical, choice of security mechanisms is important in terms of satisfaction of timing requirements. One way to achieve this, is to find a security mechanism that fits and matches the timing requirements of the system (e.g., by performing timing analysis), and then implement it [4]. This method leads to a static design in the sense that a specific security mechanism, which is analyzed, and thus, known to execute within its allowed time budget, is always used in each execution. However, this method may not be practical for systems with high complexity, which are hardly analyzable or systems with unknown timing behaviors of their components. Instead, for such systems, an adaptive approach to select appropriate security mechanisms, based on the state of the system, can be used to adapt its behavior at runtime and stay within the timing constraints. In this paper, we introduce this approach, and describe its implementation for selecting appropriate encryption algorithms at runtime (in terms of their timing behaviors) in an adaptive way, using OSE real-time operating systems [5]. To this end, the timing behavior of each execution of the encryption procedures is logged, and used as feedback for selecting a more suitable encryption algorithm in the next execution.

The rest of the paper is structured as follows. Section 2 describes the motivation and background of this work. In Section 3, we discuss the approach, describe how the adaptation mechanism in the proposed approach works. Implementation and experimental results are also explained in this section. Section 4, discusses the context where the proposed approach can be more applicable and suit well. In Section 5, related work is discussed, and finally in Section 6, conclusions are drawn, and pointers to future directions of this work are provided.

2 Background and Motivation

Designing security for real-time embedded systems is a challenging task. This is due to the fact that security features have impacts on other aspects of the system, such as timing, and if these impacts are not identified, analyzed, or managed properly, they can lead to violations of other non-functional requirements, and thus failure of the system. While in small and simple systems timing and schedulability analysis, covering security algorithms, can be done to ensure satisfaction of timing requirements, when it comes to very complex systems, such static and offline analyses might not be practical and feasible [6]. Even in cases where they are feasible, the results of such analyses may be invalidated at run-time, due to several factors such as transient loads, difference between the ideal execution

environment (taken into account for analysis) and the actual one, which leads to violation of the assumptions that are used to perform analysis [7].

Telecommunication systems are examples of systems with high complexity that require massive execution capacity and have security requirements. Due to the complexity of these systems, the main focus in their design is to be able to handle massive connection requests and data loads that arrive in a bursty and unpredictable fashion, than just trying to merely perform analysis of all possible conditions in the system [8]. Therefore, it is important that such systems are designed in an adaptive way, so that they can reconfigure themselves at runtime to continue providing their services, although under different Quality-of-Service (QoS) levels. Also, most of the real-time tasks in these systems have soft deadlines. This means it is acceptable, in general, for the functionality of the system, if some tasks complete their jobs within a reasonable margin after their deadlines, and the result of a single deadline miss is not catastrophic.

OSE is a Real-Time Operating System (RTOS) developed by Enea [9], which is used heavily in telecommunication systems, especially by Ericsson, from Radio Network Controllers, and Radio Based Stations (RBS) to mobile devices. In this paper, focusing on the needs of such systems that require runtime adaptation, an approach is suggested to select encryption algorithms at runtime based on how they behave in terms of their time constraints. To implement and evaluate the approach, OSE is used as the base platform. It is an example of a RTOS which is designed from scratch to provide the necessary determinism level required for fault-tolerant real-time embedded systems with high availability, particularly in telecommunication domain.

3 Approach

To design a system that can adapt itself and adjust the balance between its time constraints and security level, the approach depicted in Figure 1 is suggested.

When an application needs to encrypt some data, it sends the data along with the allowed execution time for the encryption procedure to the main encryption process. Based on the received time constraint, this process will then try to encrypt the data with an appropriate encryption algorithm, by invoking the corresponding process implementing that algorithm. This is done by first consulting the log information generated by the monitor process plus a pre-defined table for ranking of preferred encryption algorithms. An example of such a table is shown in Table 1. The table is used to capture the preferences for different encryption algorithms, but it has to be in descending order in terms of execution times. As we will see, this is important for the correct behavior of the system. Comparison of execution times for different encryption algorithms can be obtained from the result of studies such as [10], which has performed performance measurements of different encryption algorithms. This table is, therefore, filled by the user, using the result of such studies. For each algorithm in the table, there is a process that implements it (named in Figure 1 as 'process for encryption algorithm 1'... 'process for encryption algorithm n').

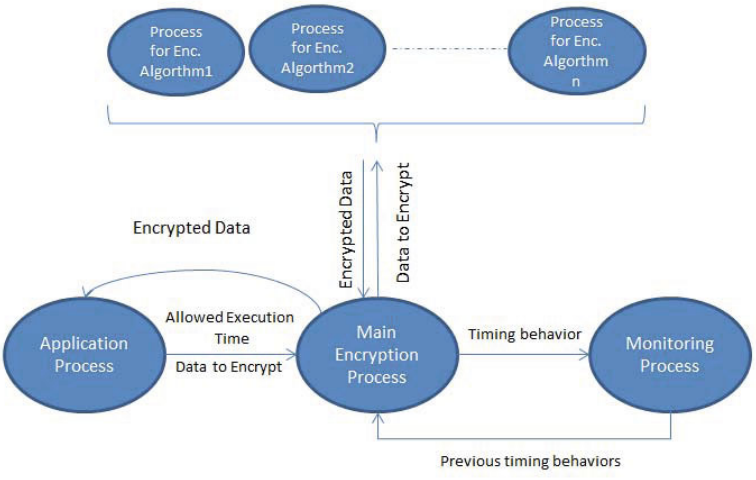


Fig. 1. Adaptive design of encryption algorithms

Table 1. Preferred encryption algorithms in descending order in terms of execution times

Rank	Encryption Algorithm
1	AES
2	3DES
3	DES
...	

Then, when an appropriate encryption algorithm is decided (the following section describes in detail how this is done), the main encryption process passes the data to the process implementing that encryption algorithm to perform the actual encryption of the data. The time taken from the point that the main encryption process sends the plain data it has received from the application process to one of the processes implementing an encryption algorithm, until encrypted data is returned to the main encryption process by it, is sent as part of the log information to the monitoring process. This log information will be used as feedback in the next invocation of the main encryption process.

The assumption that is implicit in this design is that, when it is detected that an executing encryption algorithm is exceeding its allowed time budget, it is basically more costly to terminate it in the middle of the encryption procedure, and restart encryption of the data with another encryption algorithm, than just letting it finish its job, and instead use one with a lower execution time in the next invocation of encryption procedures. This is why the encryption algorithms in Table 1 need to be sorted according to their execution times. In this way, the system first tries to encrypt all data with the algorithm at the top of the

table. If it is observed that it cannot fulfill the specified time constraint, the second ranked algorithm (which has shorter execution time) will be chosen for encryption in the next invocation.

On the other hand, if an encryption algorithm completes its job sooner than its specified time limit, the unused portion of its time budget is used to calculate and determine whether it is feasible to go back to the previous higher ranked algorithm for the next encryption job or not. Using this approach, the system tries to adapt itself based on the feedback it receives regarding its timing behavior. Therefore, when a burst of processing loads arrive, the system adapts itself to this higher load, and when the processing load decreases, it can gradually go back to using more time-consuming (and presumably more secure) encryption algorithms. This is, for example, very useful in telecommunication systems, where lots of other services (than the one(s) using encryption) are active and need to be responsive at the same time, where task pre-emption and context switches increase dramatically and interfere with the encryption job.

3.1 Log Information and Adaptation Mechanism

The information that is logged by the monitor process has the following format:

Timestamp, Encryption algorithm, Time constraint, Actual execution time

An Example of the generated log information is shown in Table 2.

Table 2. Sample log information

10360, AES, 50, 90
11800, 3DES, 80, 70
14500, 3DES, 60, 70
21353, DES, 60, 10
22464, 3DES, 90, 40
23112, AES, 50, 50
28374, AES, 60, 58

The table shows that the system has first performed encryption using AES algorithm, with a time constraint of 50 time units, but the actual execution time has been 90; in other words, it has violated its time constraint. Therefore, in the next execution, 3DES is automatically chosen for encryption (as it is the first lower ranked algorithm under AES in Table 1), which has finished its job in 70 time units while having 80 as its time constraint. Hence, no change in the encryption algorithm is observed and the same algorithm (3DES) is used for the third execution as well. The fifth and sixth rows in Table 2 (which represent the fifth and sixth executions) show that the system has chosen to apply a higher ranked encryption algorithm (fourth to fifth: DES->3DES, and fifth to sixth: 3DES->AES).

In the log that is kept in memory from the log information generated above by the monitoring process, if an encryption algorithm is again selected for the next invocation, its latest log record will replace the previous one. In other words, for each two consequent log records for a certain encryption algorithm, only

the most recent one is kept. This is done to only keep the information that is necessary, which also leads to having only log information indicating changes of encryption algorithms being stored. Table 3, shows what is actually necessary from the generated log information shown in Table 2, for making adaptation decisions.

Table 3. Necessary portion of log information for making adaptation decisions

10360, AES, 50, 90
14500, 3DES, 60, 70
21353, DES, 60, 10
22464, 3DES, 90, 40
28374, AES, 60, 58

Considering the last row from the log as:

ts, alg, t, e
(ts: timestamp, alg: encryption algorithm, t: time constraint, e: actual execution time)

the decision that the system should adopt a lower ranked algorithm is made using the following formula:

(i) $e > t \Rightarrow$ *move down in the encryption algorithms table and select the next algorithm with a lower rank.*

Also, considering the two log records described as follows:

ts(l), alg(l), t(l), e(l) : representing the last log record
ts(h), alg(h), t(h), e(h) : representing the log record for the first encryption algorithm with a higher rank that was used before the last log record

the decision to adopt a higher ranked algorithm is made using the following formula:

(ii) $e(l) < t(l) \wedge t(l) - e(l) > abs(e(h) - t(h)) \Rightarrow$ *move up in the encryption algorithms table and select the previous higher ranked algorithm.*

For example, in Table 3, applying formula (i) on row 1 (i.e., 90>50) shows that AES has taken more time than it was allowed to; therefore, a lower ranked algorithm (3DES) is used for the next invocation. However, for row 4, and the higher ranked algorithm just before it, which is AES at row 1, formula (ii) holds (i.e., 90-40>abs(90-50)); therefore, at the next invocation (corresponding to row 5), AES algorithm was used again.

3.2 Implementation Details

The implementation of the approach is done on OSE real-time operating systems [5]. The OSE edition that is used is OSE Soft Kernel (OSE SFK), which is a simulation of the actual environment to be downloaded to the target embedded hardware, and is possible to run on a host machine (e.g., on Windows or Linux).

The execution unit in OSE corresponding to a real-time task is called process. For all the processes depicted in Figure 1, an OSE process is implemented. OSE offers two types of processes: static and dynamic. Static processes are created upon system startup, cannot be terminated, and last as long as the system is up and running. On the other hand, dynamic processes can be created and killed on the fly by another process using OSE APIs. Main application, main encryption process, and monitoring process, are created as static processes. Each encryption algorithm is implemented as dynamic processes, which are, in turn, created by the main encryption process at runtime as needed. This is just a design choice to reduce the number of active processes in the system, as encryption algorithms can also be well created as static processes, in which case, the overhead of creating them for each invocation will be reduced, while increasing the total scheduling overhead and memory usage of the system.

An interesting feature of OSE is offering the concept of *load module*. Load modules are relocatable program units that can be loaded into a running system and dynamically bound to that system. A loadable module can be uploaded, rebuilt, and quickly downloaded while the remainder of the system continues to run [5]. A load module can be considered similar to Windows .exe or .dll files. Once installed, the program (consisting of one or more processes) that they contain, can be created and started. In the case of our system, this means that security designers can add new encryption algorithms to the system dynamically or update them on the fly, while the system is active and running. Such a feature is very important in high-availability systems, where updates and upgrades (e.g., patching security issues) should affect the up-time of the system to the least degree possible.

Also, the direct and asynchronous message passing mechanism in OSE, makes this real-time operating system a great choice for use in distributed systems. This further facilitates the scalability of systems that are built on OSE. Data between processes are passed as signals using three basic OSE APIs for signal passing: *send*, *receive*, and *alloc* (to create signals containing data). The Inter-Process Communication protocol (IPC) used in OSE, makes the location of processes transparent to the user; meaning that no matter whether two processes are located on the same board or on different ones, the communication between them is done using the same set of APIs and code. Using this feature, the proposed approach is implemented without the need to use shared memory among processes. The communication between processes is implemented using the three above-mentioned APIs. This brings along the possibility to deploy the processes shown in Figure 1 on different processors and boards without affecting the generality of the approach, which is important for highly-distributed real-time embedded systems such as telecommunication systems.

Using the aforementioned features, several signals have been defined for synchronization, and to pass data between processes. For example, signal HISTORY_INFO_REQUEST is defined which is sent from the main encryption process to the monitoring process to request log information. In case of receiving this signal, the monitoring process sends last log record, and the log record for

the first encryption algorithm with a higher rank, used before the last record, to the main encryption process using HISTORY_INFO_REPLY signal. Using this signaling mechanism also allows to deploy processes on different nodes if needed. This is possible since the necessary information to make adaptation decisions such as the time it takes to encrypt is passed between processes as part of the signals, making the actual location of processes in different nodes unimportant and transparent for the approach to work. The rank table for encryption algorithms is actually implemented using enumerations in C/C++ in the main encryption process.

3.3 Evaluation

To test the behavior of the system, a tool called CPU Killer [11] was used to create desired percentage of CPU loads at desired times. Moreover, Optima, which is a debugging, profiling, and monitoring tool developed by Enea for OSE, was also used to monitor and observe, in the form of graphs and tables, CPU usage levels at different system ticks from the startup of OSE. The system was run two times: once without having adaptation and the second time using adaptation. At each time, CPU loads of 10%, 50%, 70%, and then back to 50%, and 10% were applied. The results are shown in Figure 2.

The columns for each log record identify: system time (ticks), encryption algorithm, time constraint (ticks), and actual execution time (ticks). As mentioned in the previous section, an enumeration in the form of *"enum algorithms { AES=1, THREEDES=2, DES=3 };"* was used to represent the information of Table 1 in the code. The logs are decorated here with additional marks to facilitate explanation and understanding of the results.

In case I, where no adaptation was applied, AES (as number 1 in the second column) algorithm is constantly used for encryption. This is because the goal is to provide the maximum level of security, and therefore, the system is designed to prefer and choose the topmost encryption algorithm (whenever possible) from the table, which represents the strongest one. The system was started while applying 10% CPU load, and as can be seen from the figure, encryption is done within its time constraint of 300 and no violation is observed. However, when CPU load is increased to 50%, encryption starts violating its time constraint. Violations are marked with * mark in the figure. In case of the first violation, it can be seen that encryption was completed in 305 ticks while the time constraint is 300 ticks. Violations get worse (with more time margins) at 70% CPU load. It is only after going back to 10% CPU load, that encryption can meet its constraint again.

In the second case (II), where adaptation was used, it can be easily understood at the first sight that the number of violations have decreased. The first violation occurs when the CPU load is set to 50%, however, the system adapts itself to this new load and uses 3DES (as number 2 in the second column), which helps the system to perform within its time constraint again. When CPU load is set to 70%, violations are again observed. Therefore, the system adapts itself by using DES (as number 3 in the second column) instead of 3DES, to stay within the

(I) No Adaptation		(II) With Adaptation	
10%:	580,1,300,258	10:%	584,1,300,276
	859,1,300,269		868,1,300,273
	1145,1,300,276		1155,1,300,277
	1429,1,300,274		1441,1,300,276
	1711,1,300,272		1719,1,300,268
50%:	2027,1,300,305*		2111,1,300,382A
	2474,1,300,429*		2331,2,300,203
	2918,1,300,430*	50%:	2770,1,300,428*
	3364,1,300,432*		2996,2,300,211
	3813,1,300,435*		3229,2,300,214
70%:	4482,1,300,658*		3452,2,300,203
	5399,1,300,900*		3706,2,300,243
	6301,1,300,881*	70%:	4105,2,300,381*
	7199,1,300,880*		4500,3,300,380*
	8115,1,300,898*		4880,3,300,361*
50%:	8640,1,300,505*		5257,3,300,360*
	9086,1,300,429*		5639,3,300,362*
	9529,1,300,428*		5950,3,300,294A
	9974,1,300,430*	50%:	6162,3,300,194
10%:	10285,1,300,296		6380,2,300,197
	10556,1,300,261		6594,2,300,199
	10819,1,300,251		6810,2,300,200
	11083,1,300,255		7023,2,300,200
	11349,1,300,256		7236,2,300,199
			7450,2,300,199
			7641,2,300,177
		10%:	7754,2,300,103
			8026,1,300,262
			8307,1,300,270
			8572,1,300,255
			8846,1,300,264

Fig. 2. Results of running the system with and without adaptation

time constraint. Even using DES, the system still fails to meet its constraint, however, within a smaller time margin. In spite of violations, since no lower rank algorithm than DES was defined in this experiment, the system keeps using it as the last possible choice. When the CPU load is reduced back to 50%, using the two formulas described at the beginning of the section, the system realizes that it can go back to using a higher ranked algorithm (3DES in this case; hence number 2 is again observed in the second column as the used algorithm) without causing any violations. Finally, by reducing the CPU load further to 10%, the systems goes back to using AES again.

Two rows are marked with A (at time 2111 and 5950) to show anomaly in the results. The first one which shows a violation for AES algorithm under 10% load, while previous rows show that it can meet its constraint under this load. This can be due to some background services doing some work in the system at that point, which has affected AES to perform encryption as before, and thus, resulted in a violation. Or, it can be because of a relatively slow movement of the slider in the CPU Killer application, which is used to raise the CPU load to 50% manually, and thus resulting in the system working in a CPU load between 10% and 50% for a short while at that moment. This can also be the reason for the anomaly observed in the next row marked with A (at time 5950). In this case,

this row shows that DES has managed to complete within its time constraint under 70% CPU load. While, it could not perform so in the previous rows related to this algorithm (having number 3 in the second column). This can again be because of the relatively gradual decrease of the CPU load from 70% to 50%, causing the system to work at some CPU load in between.

Also another interesting observation from this result is that, as the consequence of using adaption, more encryption jobs have been performed in the second case (II), under a shorter period of time.

4 Discussion

Our suggested approach and the way we implemented it, gives this flexibility to have different time constraints for each invocation of encryption procedures. This may not, however, be needed in all systems, and only a fixed value (e.g., originating from a system level requirement) might seem to be enough for many situations, but other systems can well benefit from this flexibility.

The whole adaptation mechanism can also be used as an option; in the sense that, if a system detects certain patterns in CPU load variations and violation of timing constraints in the applied encryption algorithm, it can *turn on* adaptation mechanism and let the system decide which encryption algorithm to use. Moreover, use of the suggested adaptation mechanism may be most beneficial when there are many requests for encryption and frequency of CPU load changes are such that they make the overhead of adaptation mechanism acceptable. On the other hand, if there are only a few encryption requests or there are not big changes in CPU load (or the range of changes is very small and known beforehand), using a fixed encryption algorithm may be more desirable (to remove the overhead cost of adaptation).

The security level of the system, originating from the choice of encryption algorithms, is actually determined by the list of encryption algorithms that designers choose to include in the described encryption algorithms table. So, for example, if for a system only AES and 3DES are acceptable, the table can be constructed using only these two algorithms. This also defines what is the range of strongest and weakest encryption algorithms that the system may be using at any moment. Moreover, while we only focused on the algorithm itself, and did not discuss key length or block length explicitly, these factors (even the number of rounds), where applicable, can easily be taken into account using the table. For example, instead of just having AES, we can put AES256 and AES128 as items in the table, to bring into picture the role of key length, and the system will choose each when decided appropriate.

Providing adaptations on encryption algorithms, also automatically leads to some sort of security through obscurity (note the famous quote of "security through obscurity is not security") [12,13]. One interesting topic that we leave as a future direction of this work to be investigated, is that whether adaptive mechanisms, as the one described here, can lead to weaknesses in the system and facilitate the job of attackers. For instance, issues such as this can be analyzed

more throughly that if attackers get to know the details of adaptation mechanisms, they might force the system into adopting the lowest ranked (weakest) encryption algorithm by creating CPU loads, and making it easier for themselves to break into the system.

5 Related Work

Designing security features for embedded systems has its unique challenges and requires specific engineering methods and considerations. These issues have been the subject of many studies such as [14,1,2]. [14] and [1], focus on these unique challenges of security in embedded systems in general, and discuss them under the *processing gap*, *battery gap*, *flexibility*, *tamper resistance*, *assurance gap*, and *cost* titles. [14] also provides workload analysis of SSL protocol, and examples for energy consumption of different ciphers, to discuss and illustrate the impacts of security features in embedded systems. The vision for security engineering of embedded systems in the scope of a project is described in [2].

[15] and [10] are examples of works that study the impacts of security mechanisms on specific aspects of a system. Measurement and comparison of memory usage and energy consumption of different encryption algorithms on two different sensor nodes (MicaZ and TelosB motes) are performed and discussed in [15], and [10] offers performance and timing comparisons of encryption algorithms on two Pentium machines. The approach we proposed in this paper, relies on the results from the performance and timing comparisons of encryption algorithms as provided in the aforementioned study.

In this paper, an adaptive way to deal with the timing costs and requirements on security mechanisms was introduced. It should, however, be mentioned that there are other ways for taking into account these timing costs, which might suit very well other types of systems than discussed here. In systems with less complexity which are analyzable, a static and non-adaptive structure can be designed (i.e., a fixed set of security features will be used all the time e.g., to encrypt data). The idea we proposed in [4] is basically an example of such approach and systems.

The use of adaptive approaches and feedback mechanism for better CPU utilization and task scheduling in dynamic systems, where execution times of tasks can change a lot at runtime, has also been the topic of many studies in the real-time systems domain, such as [16]. One of the interesting works in the area of security for real-time embedded systems which uses an adaptive method is the study done in [17]. One difference between our work and [17] is that, there, the focus is on a set of periodic tasks with known real-time parameters, whereas, our main target is complex systems consisting of periodic, sporadic and aperiodic tasks. Therefore, the analysis and formulas they offered in that work may not be applicable or need to be extended to support the type of systems we discussed here. Also, they consider the security level of the system as a QoS value explicitly, while in this paper, it is considered implicitly and left to the user through the use of a sorted table for encryption algorithms. Moreover, the

main adaptation component of the system in that work is key length, while in our work it is the encryption algorithms that are adaptively replaced, and can easily include key length or any other relevant parameters as well. One of the interesting and close studies to our work is [18]. They basically use a similar type of adaptation mechanism as ours. However, the main focus in this work is on client-server scenarios using a database, and to manage performance of transactions. Also, security level adjustment in this work is done periodically using a security manager component, while in our method, adaptation mechanism executes per request and is not active when application has no request for encryption. Moreover, in that work, while security level switch is occurring, it can lead to use of an inappropriate encryption method by a client, which is solved by rejecting it, through passing several acknowledgment messages and repeating the process. Therefore, synchronization and message loss due of out of order arrival of messages are problematic for the security manager, which is handled by re-sending of data and applying other means.

As another approach for managing security in real-time systems, Tao Xie and Xiao Qin, has basically incorporated timing management of security mechanisms as part of the scheduling policy and developed a security-aware scheduler in [19].

6 Conclusion and Future Work

In this paper, we discussed security, as a non-functional requirement, in the design of real-time embedded systems, and particularly, how the choice of encryption algorithms, can affect timing requirements in these systems. An adaptive approach for selection of encryption algorithms was suggested for systems which need to balance their services at runtime in order to achieve their time constraints. It was shown how the approach can help the system to react to different processing loads and perform its encryption procedures within the defined time constraints. While, OSE RTOS was used as the base platform for implementation of the approach, there is nothing that stops it from being implemented on other platforms.

In the suggested approach here, a gradual increase or decrease of the rank of encryption algorithms was used. In other words, in each adaptation step, the system chooses either the next higher or lower ranked algorithm. As a future work, it can be evaluated how the approach would perform, if in each adaptation, the lowest or highest ranked algorithm was selected instead. For example, if it is observed that the system completes its encryption job earlier than its time constraint, it jumps to the top of the rank table and chooses the highest ranked algorithm for the next invocation. It would be interesting to study for which systems/situations, each of these methods work better and categorize systems accordingly. Calculating the overhead of adaptation mechanisms and optimizing them is also left as a future study.

Moreover, in this work we focused on complex systems with not much information about timing properties of each individual task in the system to perform analysis. This situation was observed in the design of a telecommunication subsystem during our work in the CHESS project [20]. Accordingly, the approach

that is suggested in this paper tries to improve satisfaction of timing constraints of the system by keeping a history of the timing behavior of the system. There is room to improve the suggested adaptation mechanism by taking into account more information about the system than was used here, and also more knowledge about the task that requires encryption when available.

Another direction of this work is to introduce other factors besides time for performing adaptations. These factors may include energy consumption, memory usage, and even situations where a system is under attack. Moreover, it was discussed whether and how adaptation mechanisms might actually help attackers to break more easily into a system. This issue can serve as an interesting topic for more careful investigations.

References

1. Kocher, P., Lee, R., McGraw, G., Raghunathan, A.: Security as a new dimension in embedded system design. In: Proceedings of the 41st Annual Design Automation Conference, DAC 2004, Moderator-Ravi, Srivaths, pp. 753–760 (2004)
2. Gürgens, S., Rudolph, C., Maña, A., Nadjm-Tehrani, S.: Security engineering for embedded systems: the secfutur vision. In: Proceedings of the International Workshop on Security and Dependability for Resource Constrained Embedded Systems. S&D4RCES 2010, pp. 7:1–7:6. ACM, New York (2010)
3. Cysneiros, L.M., do Prado Leite, J.C.S.: Non-functional requirements: From elicitation to conceptual models. IEEE Transactions on Software Engineering 30, 328–350 (2004)
4. Saadatmand, M., Cicchetti, A., Sjödin, M.: On generating security implementations from models of embedded systems. In: The Sixth International Conference on Software Engineering Advances, ICSEA 2011 (2011)
5. Enea: The architectural advantages of enea ose in telecom applications, `http://www.enea.com/Templates/Landing.aspx?id=27011` (last accessed September 2011)
6. Wall, A., Andersson, J., Neander, J., Norström, C., Lembke, M.: Introducing Temporal Analyzability Late in the Lifecycle of Complex Real-Time Systems. In: Chen, J., Hong, S. (eds.) RTCSA 2003. LNCS, vol. 2968, pp. 513–528. Springer, Heidelberg (2004)
7. Chodrow, S., Jahanian, F., Donner, M.: Run-time monitoring of real-time systems. In: Proceedings of the Twelfth Real-Time Systems Symposium, pp. 74–83 (1991)
8. Saadatmand, M., Cicchetti, A., Sjödin, M.: Uml-based modeling of non-functional requirements in telecommunication systems. In: The Sixth International Conference on Software Engineering Advances, ICSEA 2011 (2011)
9. Enea, `http://www.enea.com` (last accessed September 2011)
10. Nadeem, A., Javed, M.: A performance comparison of data encryption algorithms. In: First International Conference on Information and Communication Technologies, ICICT 2005, pp. 84–89 (2005)
11. CPU Killer, `http://www.cpukiller.com/` (last accessed September 2011)
12. Mercuri, R.T., Neumann, P.G.: Security by obscurity. Commun. ACM 46 (2003)
13. Hissam, S., Weinstock, C., Plakosh, D., Jayatirtha, A.: Perspectives on open source. Software Engineering Institute, Carnegie Mellong, `http://www.sei.cmu.edu/library/abstracts/reports/01tr019.cfm` (Published November 2001, last accessed September 2011)

14. Ravi, S., Raghunathan, A., Kocher, P., Hattangady, S.: Security in embedded systems: Design challenges. ACM Transactions on Embedded Computing Systems (TECS) 3, 461–491 (2004)
15. Lee, J., Kapitanova, K., Son, S.H.: The price of security in wireless sensor networks. Journal of Computer Networks 54, 2967–2978 (2010)
16. Khalilzad, N.M., Nolte, T., Behnam, M., Åsberg, M.: Towards adaptive hierarchical scheduling of real-time systems. In: 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011) (2011)
17. Kang, K.D., Son, S.H.: Towards security and qos optimization in real-time embedded systems. SIGBED Rev. 3, 29–34 (2006)
18. Son, S.H., Zimmerman, R., Hansson, J.: An adaptable security manager for real-time transactions. In: Euromicro Conference on Real-Time Systems, pp. 63–70 (2000)
19. Xie, T., Qin, X.: Scheduling security-critical real-time applications on clusters. IEEE Transactions on Computers 55, 864–879 (2006)
20. CHESS Project: Composition with Guarantees for High-integrity Embedded Software Components Assembly, `http://chess-project.ning.com/` (last accessed August 2011)

Hunting Application-Level Logical Errors

George Stergiopoulos, Bill Tsoumas, and Dimitris Gritzalis

Information Security and Critical Infrastructure Protection Research Laboratory
Dept. of Informatics, Athens University of Economics and Business (AUEB)
76 Patission Ave., Athens GR-10434, Greece
{geostergiop,bts,dgrit}@aueb.gr

Abstract. Business applications are complex artefacts implementing custom business logic. While much research effort has been put in the identification of technical vulnerabilities (such as buffer overflows and SQL injections), application-level logic vulnerabilities have drawn relatively limited attention, thus putting the application's mission at risk. In this paper, we design, implement, and evaluate a novel heuristic application-independent framework, which combines static and dynamic analysis, input vector, and information extraction analysis, along with a fuzzy logic system, so as to detect and assert the criticality of application-level logic vulnerabilities in Java stand-alone GUI applications.

Keywords: Error Detection, Vulnerability, Application Logic, GUI.

1 Introduction

Vulnerability management in information systems mainly concerns the individual standard components and, usually, does not refer to the business applications per se. The security of the application logic itself (which effectively realizes the business logic programmed inside) is vastly unexplored due to the diverse and customized nature of the business logic and developer intentions. We define as "Application Logic Vulnerability" (hereafter "Logic Vulnerability", "LV"), *the flaw present in the faulty implementation of the business logic within the application code*. An example from [9]: An online store web application allows users to use coupons to obtain a one-time-discount-per-coupon on certain items; a faulty implementation can lead to use the same coupon multiple times, eventually zeroing the price.

In this paper we introduce APP_LogGIC, a novel framework for the identification and criticality assessment of application LVs in *standalone Java applications with a GUI*. Our prototype-level implementation analyzes application code and introduces a new way of vulnerability assessment. It is based on extensive code analysis from multiple viewpoints, which feed a mathematical fuzzy logic system calculating the Vulnerability and Severity of possible LVs, close to a security auditor analysis.

In section 2 we briefly review existing work. In section 3 we present our idea. Section 4 describes the fuzzy LV-reporting module, whereas section 5 details our experiments. Finally, we conclude and discuss future work in section 6.

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 135–142, 2012.

At a glance, the paper contributes the following:

(a) A new method (based on [9]), capable of detecting LVs in GUI applications, through the use of dynamic and static analysis, along with the use of information gathered through a suggested Information Extraction Method (IEM), with almost zero false positives.
(b) A new IEM, which makes use of an existing effective Input Vector Analysis method, an Abstract Syntax Tree analysis method, and a code-branching analysis method. It provides extensive structural information for any Application Under Test (AUT).
(c) In collaboration with the Java PathFinder [11] group, we extended the functionality of its JPF-AWT component used and fixed a number of bugs of this tool.
(d) Finally, we propose and implement *a method* for asserting and evaluating possible LVs ("APP_LogGIC"), by using a Fuzzy Logic system, which computes and graphically represents the criticality of each possible LV. [13].

2 Related Work

In this section, we briefly summarize a few pieces of related work. Researchers in [1], [2], [4] and [5] focus on detecting code vulnerabilities but fail to include any method for detecting LVs.

Researchers in [9] use the MIT's Daikon dynamic analysis tool to infer a simple set of behavioral specifications for a web application. Then, they filter the learned specifications to reduce false positives and use NASA's JavaPathFinder (JPF) tool [11] for model checking over symbolic input, in order to identify program paths that, under specific conditions, may indicate the presence of a certain type of web application logic flaws.

The authors in [2] propose an approach for penetration testing using: (a) static analysis for identifying possible input vectors to an application, and (b) dynamic analysis for automatic analysis the output of the AUT.

3 The APP_LogGIC Framework

APP_LogGIC framework analyzes applications with no finite-state execution (i.e., with potentially indefinite scenarios of actions). It flags possible LVs by using generic characteristics of their behavior and location in code. The more suspicious a point of source code is, the higher it scores in the Fuzzy Logic system [13].

The overall APP_LogGIC architecture is depicted in Fig. 1. More specifically:

(a) The *Invariant-Based Method* (IBM) is based on [9] and uses Daikon dynamic analysis to extract possible source code rules that describe the business logic of an AUT. It, also, uses JPF to statically analyze the Daikon results.
(b) The *Information Extraction Method* is a set of IEMs, i.e., a Decision Analysis Method that analyzes branches in source code, a Javac Abstract Syntax Tree (AST) Parser that focuses on the relationships and structure of the AUT's code, and an Input Vector method tracking AUT data entry points.

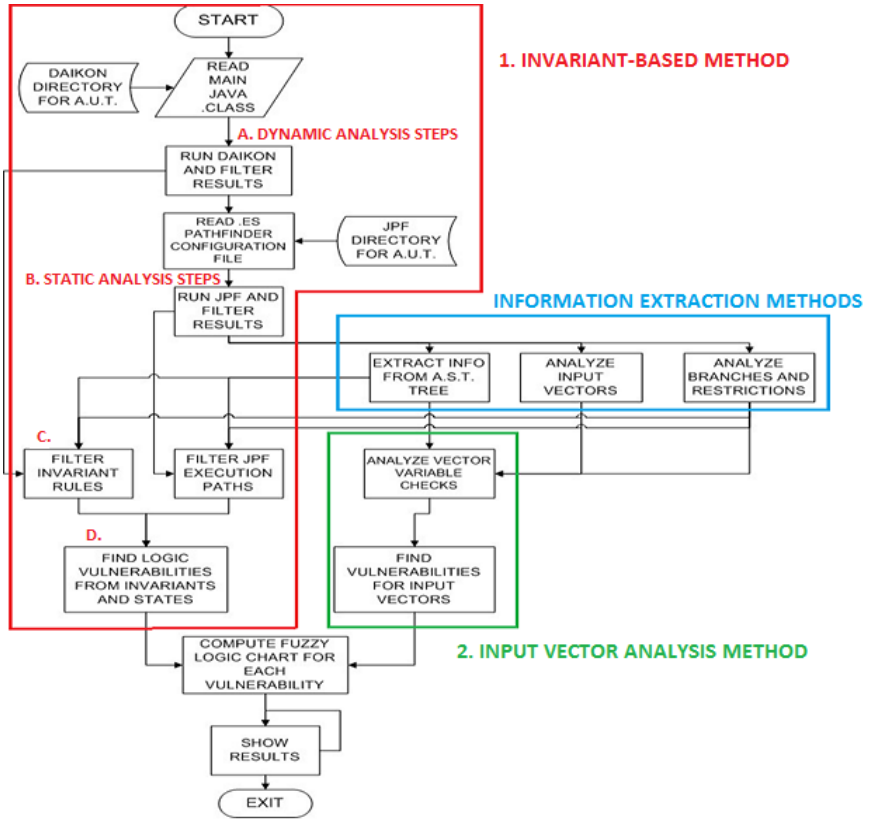


Fig. 1. The APP_LogGIC architecture

(c) The *Input Vector Analysis Method* extends work in [2] and provides information for the detection of possible LVs due to incomplete checks on data.
(d) APP_LogGIC combines the information selected from the previous methods, and assesses the criticality of all suspicious sources using Fuzzy Logic.

The analysis of GUI applications is done with the use of the referred heuristic mechanisms and a fuzzy logic system. It aims at determining the criticality and impact of the detected key-points in the source code. This is a novel approach, as it has not been applied so far to stand-alone applications, let alone inter-platform Java applications with a GUI.

The functional parts of the APP_LogGIC architecture are presented in the sequel.

3.1 The Invariant-Based Analysis Method (IBM)

The IBM combines Dynamic and Static Analysis; however, our Static Analysis approach does not use symbolic execution [9], but scripted, looped sequences of actions feeding the application, as well as a listener recording the state of data variables.

Dynamic Analysis

This step monitors the execution of the AUT with input data. By using the Daikon tool [8], we extract a set of traces representing expected application behavior based on various sets of inputs, named *logic invariants* (i.e., logic rules that hold true during all possible execution states and describe the relationships and restrictions enforced on the application variables). Since these invariants hold throughout the execution of the method they refer to, they eventually provide a window towards detecting the logic.

```
---LoginFrame.initComponents():::EXIT
identity == "-1"
```

Fig. 2. Invariant produced by Daikon Dynamic Analysis

In Fig. 2, the line *initComponents():::EXIT* shows that the invariant *<identity>* must be equal to "-1", holding as a rule whenever the LoginFrame.*initComponents()* method finishes its execution. APP_LogGIC, uses information extracted with IEM to filter all such invariants and focus on those that hold LVs (Fig. 1, Blue Method).

APP_LogGIC adopts the approach in [9], by focusing on the invariants for the "IF" branches checks, lying inside the AUT code; the IBM uses the results from the Decision Analysis and categorizes the produced invariants by their restriction in branch decisions.

Static Analysis

Each application function is a sum of sequential instruction executions (*execution path*). This step uses scripted GUI program executions, so as to produce the execution paths that are necessary to cover the AUT's functionality. The analyst should ensure that Daikon fully covers the AUT possible execution paths, otherwise the AUT logic rules may not be correctly extracted.

These paths, together with the recording of all paths' variables, serve as clues for the detection of LVs. In order to statically analyze AUTs, we use the JPF tool [14] and a VarRecorder listener. To make JPF work with GUI applications, we extended its JPF-AWT component by fixing bugs in its (a) Swing part and (b) native code execution; changes were incorporated in the official JPF-AWT package [11]. Fig. 3 depicts logging variable states and execution paths.

```
VARIABLE: user -> "admin"
        LoginFrame.java:122             : if (user.equals("") && pass.equals(""))
        LoginFrame.java:125             : if (identity.equals("-1")) {
VARIABLE: identity -> "1"
        LoginFrame.java:125             : if (identity.equals("-1")) {
        LoginFrame.java:132             : System.out.println("Login");
VARIABLE: out -> gov.nasa.jpf.ConsoleOutputStream@e5
```

Fig. 3. Execution Path with variable states and variable parsing steps

Detecting Potential Logic Vulnerabilities

We detect the possible LVs with the IBM as follows: APP_LogGIC detects points in static analysis execution paths, where a variable described in an invariant rule is used.

Then, it records whether it enforces or violates the invariant produced by dynamic analysis. Next, we check the rest of the execution paths produced by the application, in search of paths that violate the same rule. For example; back to the Identity invariant case (Fig. 3), the current execution path violates the rule *identity == "-1"*, as the path finishes with Identity having the value "1". If another version of the same execution path exists, which finishes with a "-1" state, then the *Identity* might be a LV.

Information Extraction Method (IEM)
The IEM uses information from the IBM, the Javac compiler Abstract Syntax Tree (AST), and the entry points of input vectors, in order to refine data related to possible LVs. The outputs are fed back to the IBM for enhancing the filtering of the invariant rules and JPF execution paths (point "C", Fig. 1), thus enhancing the overall quality of the data related to the possible LVs. The three steps of it are: 1) *Branches and Restrictions Analysis*: Decisions in an execution path are, generally, implemented by two means: a *branch* (selection point) and a *set of restrictions* (a variable or constant, with a specific value within the branch). These provide sufficient information for the detection of LVs, as the vulnerability is expressed as an erroneous branch decision during execution. Thus, we developed a method that extracts the AUT logic from source code branch restrictions; 2) *Information Extraction from Javac AST*: the Javac compiler is used to provide structural info for the AUT, which is later fed into the Fuzzy Logic. The code is represented as a tree, with variables or values as leafs and instruction as nodes. As all the work is done by the Javac, the APP_LogGIC's execution is kept fast, no matter the size of the AUT; 3) *Analysis of Input Vectors (Entry Points)*: The Input Vector Analysis component monitors the checks enforced on source code variables that hold data from input vectors. It performs structural (REGEX) checks and analyzes: a) the tainted variable that holds the initial value passed from a vector; b) the structure of the data inside a tainted variable (but not their actual content); and c) occasions where user input is never checked or sanitized in any way. Finally, the results are combined with information from the IEM component and a certain subset of LVs is detected: the *LVs that exist due to insufficient checks on input vector data.*

4 Fuzzy Logic System

We use a scalable Fuzzy Logic system [13] in order to rank possible LVs according to their severity and place in code. This ranking system, combined with our improved code analysis, provides a robust way of identifying yet-unknown LVs. We use two 5-grade Likert scales (1 for low, 5 for max, Table 1), for ranking the Vulnerability and the Severity of a specific code point of interest (Severity: "*How severe the impact will be if a LV does exist at the specific point in source code*", Vulnerability: "*How high is the possibility for a specific, critical point inside the source code to contain a LV*").

According to the position of each restriction in the code, a severity value is assigned in each variable involved in the restrictions found - i.e., the Basic Severity takes into account whether a variable acts as a restriction or a sink [12] (i.e. data store) inside the AUT code. Finally, Extended Severity and Final Severity are calculated.

Table 1. Criteria for calculating Basic, Extended, and Final Severity

STEP 1: CALCULATE BASIC SEVERITY	STEP 2: CALCULATE EXTENDED SEVERITY
Criterion 1: Random variable	Variable used as a restriction on an "IF" branch twice or more
Criterion 2: Variable used as a sink for data originated from input vectors	Variable used as a sink AND as a restriction on an "IF" or "switch" branch
Criterion 3: Variable used as a restriction on an "IF" branch	**STEP 3: REGISTER FINAL SEVERITY**
Criterion 4: Variable used as a restriction on an "switch" branch	

The other factor under consideration is the *Vulnerability, i.e.,* if a branch variable is never checked or sanitized, its Vulnerability level is set to 5 (max). All information gathered along with the newly assigned Vulnerability and Severity ranks are fed to the Fuzzy Logic, so as to qualitatively estimate the criticality of the detected possible LVs. We configured the system to use linguistic variables and rules of the IF-THEN type (Fig. 4). Based on the Severity and Vulnerability scoring of every possible source code point that might be a LV, the Criticality of each detection is plotted separately, thus *producing a numerical and a fuzzy result*. The former result is calculated by the *Center of Gravity* technique[1] via its Severity and Criticality values (Fig. 5).

RULE 1:
IF Severity IS small AND Vulnerability IS trifle THEN Criticality IS

Fig. 4. Example of a Fuzzy Logic rule

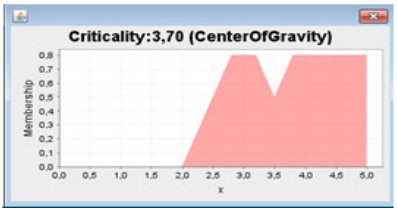


Fig. 5. APP_LogGIC output sample: Criticality of a possible logic vulnerability

5 Implementation

We developed a sample Java GUI application testbed ("*LogicBomb*"), so as to avoid the technical limitations and incompatibilities of external tools (e.g. JPF's limited static analysis in Swing GUIs) and also to demonstrate our method capabilities.

[1] For further information on the rule ranking system please refer to [7].

LogicBomb contains five LVs found in real world applications. Their types are described in the first three of the CCWAPSS scale criteria [13]. Three of the LVs introduced are insufficient checks on Input Vector sinks and one is an invariant violation that manifests due to erroneous transfer of data between variables in source code (a common error, using global variables and failing to set and clear their contents correctly). Notably, the fifth vulnerability - an LV generated by an erroneous sink usage - was not coded deliberately, but was also successfully detected.

APP_LogGIC framework was tested on an Intel Core 2 Duo E6550 PC (2.33 Ghz, 2GB RAM). Table 2 shows the execution times for the tools used (including the APP_LogGIC analysis). The Dynamic Analysis step includes the time needed for the user to manually execute AU (actual Daikon processing took ~4 sec).

Back to the Identity example, the IBM found the LV due to the violation of the invariant Identity == "-1" and correctly assigned with a *Vulnerability value of 5* and a *Severity value of 4* for the Fuzzy Logic (**Fig. 5**). APP_LogGIC dynamically detected all vulnerabilities and assigned fuzzy logic values accordingly.

Table 2. Execution Times

Type of execution	**Time** (sec)
Dynamic Analysis	26.519
Static Analysis (JPF)	6.266
APP_LogGIC analysis	1.172

5.1 Limitations

The current version of the framework, though able to detect numerous vulnerabilities, suffers by a number of limitations. First, the types of vulnerabilities that can be detected are limited to those that appear in the control flow restrictions found inside an execution path. Also, due to inherent incompatibilities, the current version of APP_LogGIC does not support the Java language "switch" branch type. The Daikon tool does not support the creation of invariants for loops ("While" and "For" constructs). Besides that, AUT execution should cover (ideally) all possible paths. JPF cannot analyze complex GUIs, due to the lack of support for the latest Swing classes. The use of scripted configuration files limits the amount of execution paths generated and, concurrently, the amount of information for checking the validity of invariants. Finally, the Input Vector Analysis method cannot validate the sink data context.

6 Conclusions and Further Research

We implemented a novel framework that detects LVs and assesses their criticality in Java GUI applications with visual support. Although our test results stem from real-world examples, third party applications testing is limited due to JPF.

We plan to extend our work to “While” and “For” constructs and to support more input methods. Finally, our approach cannot detect LVs based on the application variables context (we plan to use semantic constructs such as XBRL [14] or OWL [15]).

Acknowledgments. This work was performed in the framework of the SPHINX (09SYN-72-419) project (http://sphinx.vtrip.net), which is partly funded by the Cooperation ("SYNERGASIA") Programme of the Hellenic General Secretariat for Research and Technology.

Our thanks to P. Mehlitz and the NASA JPF team for their input. Credit goes, also, to S. Dritsas (AUEB) for his comments and suggestions.

References

[1] Cova, M., Felmetsger, V., Banks, G., Vigna, G.: Static Detection of Vulnerabilities in x86 Executables. In: Proc. of the 22nd Annual Computer Security Applications Conference (ACSAC 2006), USA, pp. 269–278 (December 2006)

[2] Halfond, W., Choudhary, S., Orso, A.: Penetration Testing with Improved Input Vector Identification. In: Proc. of the 2009 International Conference on Software Testing Verification and Validation (ICST 2009), pp. 346–355. IEEE Computer Society, USA (2009)

[3] Zhou, J., Vigna, G.: Detecting Attacks That Exploit Application-Logic Errors Through Application-Level Auditing. In: Proc. of the 20th Annual Computer Security Applications Conference (ACSAC 2004), USA, pp. 168-178 (December 2004)

[4] Newsome, J., Song, D.: Dynamic Taint Analysis for Automatic Detection, Analysis and Signature Generation of Exploits on Commodity Software. In: Proc. of the Network and Distributed System Security Conference (NDSS 2005), USA (February 2005)

[5] Tevis, J., Hamilton, A.: Static Analysis of Anomalies and Security Vulnerabilities in Executable Files. In: Proc. of the 44th Annual Southeast Regional Conference, USA (2006)

[6] Jovanovic, N., Kruegel, C., Kirda, E.: Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities. In: Proc. of IEEE Symposium on Security and Privacy (S&P 2006), USA, pp. 258–263 (2006)

[7] Stergiopoulos G.: Development of a methodology for identifying logical vulnerabilities in application penetration testing, M.Sc. Thesis, Athens University of Economics & Business (AUEB), Greece (2011) (in Greek)

[8] Ernst, M., Perkins, J., Guo, P., McCamant, S., Pacheco, C., Tschantz, M., Xiao, C.: The Daikon Invariant Detector User Manual. MIT, USA (2007)

[9] Felmetsger, V., Cavedon, L., Kruegel, C., Vigna, J.: Toward Automated Detection of Logic Vulnerabilities in Web Applications. In: Proc. of the 19th USENIX Security Symposium, USA (2010)

[10] Charpentier, F.: Common Criteria Web Application Security Scoring (CCWAPSS) (November 2007), `http://www.xmco.fr/whitepapers/ccwapss_1.1.pdf`

[11] Mehlitz, P., et al.: Java PathFinder. Ames Research Center, NASA, USA, `http://babelfish.arc.nasa.gov/trac/jpf/wiki`

[12] Livshits, S., Lam, F.: Finding Security Vulnerabilities in Java Applications with Static Analysis. In: Proc. of the 14th USENIX Security Symposium, USA (2005)

[13] Cingolani, P.: Open Source Fuzzy Logic library and FCL language implementation, `http://jfuzzylogic.sourceforge.net/html/index.html`

[14] Fuger, S., et al.: ebXML Registry Information Model, ver. 3.0 (2005)

[15] OWL 2 Web Ontology Language Document Overview, W3C Recommendation (2009)

Optimal Trust Mining and Computing on Keyed MapReduce

Huafei Zhu[1] and Hong Xiao[2]

[1] I[2]R, A*STAR, Singapore
[2] TEI, AFEU, Xi'An, China

Abstract. This paper studies trust mining in the framework of keyed MapReduce and trust computing in the context of the Bayesian inferences and makes the following two-fold contributions:

In the first fold, a general method for trust mining is introduced and formalized in the context of keyed MapReduce functions. A keyed MapReduce function is a classic MapReduce function associated with a common reference keyword set so that a document is projected on the specified common reference set rather the whole dictionary as that defined in the classic MapReduce function. As a result, keyed MapReduce functions allow one to define flexible trust mining procedures: a look-up table which records the comments of neighbors can be constructed from the inverted index of the keyed MapReduce function;

In the second fold, a new method for trust computing is introduced and formalized in the context of maximum likelihood distribution. A look-up table generated in the trust mining stage is now viewed as the current state of the target server and then the maximum likelihood distribution over the look-up table is deduced. We show that the proposed trust computing mechanism is optimal (an upper bound of trust values).

Keywords: Bayesian inference, Maximum likelihood distribution, keyed-MapReduce, trust computing, trust mining.

1 Introduction

Trust is considering a pre-requisite for establishing relationship in peer networks. Beth, Borcherding and Klein [4] and Yahalom, Klein and Beth [15] formally studied the trust computing in the context of direct and indirect trust values based on transitive graphs. Since then tons of definitions on trust and trust computing have been presented and analyzed, often in the association with reliability, expectancy, willingness, belief and predicate. The definition of trust is not unique and may vary depending on the context and the purpose where it is used. A commonly cited definition of trust is due to [9]: *trust of a user in a server for a service is a measurable belief of the user in the server behaving dependably for a specified period within a specified context in the relation to the service.*

G. Barthe, B. Livshits, and R. Scandariato (Eds.): ESSoS 2012, LNCS 7159, pp. 143–150, 2012.

1.1 The Motivation Problem

The state-of-the-art trust computing procedures assume a trust mining procedure is simply a query to a trust mining oracle $\mathcal{O}$ and a trust value is computed from the response of that query [13,16,17,18,19]. The trust mining oracle $\mathcal{O}$ is a probabilistic polynomial time Turing machine which assumes to be with the following properties:

- extracting a user U's history information aux on the service X provided by the sever S;
- $b \leftarrow U^{\mathcal{O}(S,X)}(aux)$, where $b \in [0,1]$ by applying the Bayesian inference technique.

The computation of a prior distribution of transactions provided by the server is an easy task in the continuous Bayesian system since it assumes that the distribution of successful transactions is uniform in the interval $[0,\infty)$, i.e., the prior distribution of E is the beta function $\int_0^\infty p^m(1-p)^{n-m}dp$, where E is an event that among n services m are successfully provided by a server S. While such a trust mining oracle assumption is not a problem within the continuous Bayesian system framework, it is a difficult task in the discrete Bayesian system to calculate the probability $\Pr(E)$. This is because we do not know the discrete distribution of successful transactions provided by the server S and no known work addresses the trust mining problem in the context of the discrete Bayesian system (to the best of our knowledge, trust mining problem is not addressed in the references mentioned above), it is certainly welcome to investigate such an interesting research problem from the point view of real-world application scenarios.

1.2 This Work

This paper studies trust mining in the context of MapReduce and trust computing in the context of maximum likelihood estimation within the framework of discrete Bayesian systems. The computation of trust consists of the following two steps

- a trust mining step: the task in the trust mining step is to collect information (such as opinions, comments and recommendations distributed in the networks) regarding a target server as much as possible and then tries to retrieve useful information from the collected information. That is, in the trust mining step, we apply keyed map functions to collect information regarding the target S who provides a service to users in the context of bulletin-board model, where users are allowed to post anonymous comments on the server. We then retrieve useful information from the collected information that will be used to calculate trust values in the terms of keyed reduced functions;
- a trust computing step: the task in the trust computing phase is to calculate $\Pr(E)$, where E denotes an event such that m out of the n ($m \leq n$) transactions behaving dependably for a specified period within a specified context

in the relation to the successful services (or services with positive comments, opinions and recommendations, etc) based on the collected information in the trust mining step. We then predict its dependable service (a measurable brief that the server provides a dependable service with probability p) in the future transactions.

In the trust mining phase, auditors of service providers are introduced. An auditor in our model is an insurance company who collects related information on the insured service providers. The introduction of service auditors gives both customers and service providers protections and a clear signal about which providers are trustworthy enough to be insured since the auditor has expertise and capabilities that the customers do not have. Since issuance premiums will reflect both the known risks and the uncertainty in risks, service providers will have an incentive to improve their risk management practices and to increase their transparency to the auditors [12]. Also since auditors understand the threats posed, know best practices, service quality and perform these checks through well-defined interfaces to the services, we assume that the auditors maintain readable and writable bulletin-board.

Individual comments posted in the bulletin-board is viewed as a document or a record stored in the database. Note that in our model, the id of an user who posts the comments on the service providers is anonymous but it is traceable by an auditor. To retrieve the useful information from the database, a tool called keyed MapReduce for trust mining introduced in [20] is applied. A keyed MapReduce is a classic MapReduce algorithm associated with a public keyword set (i.e., a common reference keyword set). A document/record is parsed as a sequence of distinct keywords in the specified common reference keyword set. A keyed MapReduce function is general in the sense that if the underlying keyword set K selected by the MapReduce function is the whole dictionary D, then a keyed MapReduce function is equivalent to the classic MapReduce function. The keyed MapReduce function benefits us to select a common reference word set K independent with a dictionary D and to avoid mining every word $k \in D$ in the stored documents d. As a result, the trust mining procedure defined over the keyed MapReduce is much more efficient and flexible than that defined over the classic MapReduce program. We retrieve the related information from which a look-up table is established and predicate a posteriori trust score by means of the maximum likelihood estimations based on the generated look-up table. The posteriori probability p is computed from the following equation $\max_p \Pr(P = p \wedge \mathrm{E}) = \max_p \Pr(P = p|\mathrm{E})\Pr(E)$. We show that if the trust value is defined by the maximum likelihood distribution of the generated look-up table, then our trust computing mechanism defined is optimal.

Road-Map: The rest of this paper is organized as follows: in Section 2, tools for trust mining is sketched; The details of trust mining on MapReduce is proposed in Section 3. We describe the method for trust computing by means of maximum likelihood distribution in Section 4 and conclude our work in Section 5.

2 A Keyed MapReduce

MapReduce introduced by Dean and Ghemawat [6,7,8] is a programming model automatically parallelized and executed on a large cluster of the commodity machines. A MapReduce function consists of two parts: a map function and a reduce function. A map function Map parses each document as words, and emits a sequence of `<documentID, word>` pairs. The reduce function Reduce accepts all pairs of a given word, sorts the corresponding document IDs and emits a `<word, list(documentID)>` pair.

Let W be a universe of words , i.e., $W = \{0,1\}^*$ and $D \subseteq W$ be a dictionary. A document d is viewed as a set of distinct words in D in the standard MapReduce program, i.e., d is viewed as a matrix M of form (document$_j$: word$_{j,1}$, ..., word$_{j,\alpha_j}$, $j = 1, \ldots, l$) below

$$\begin{pmatrix} d_1 : w_{1,1}, \ldots, w_{1,\alpha_1} \\ d_2 : w_{2,1}, \ldots, w_{2,\alpha_2} \\ \ldots \ \ \ldots, \ \ldots, \ \ \ldots \\ d_l : w_{l,1}, \ \ldots, \ w_{l,\alpha_l} \end{pmatrix}$$

where $(d_1, \ldots, d_l)$ are records/comments/recommedations stored in the bulletin-board of an auditor and $w_{j,k_j} \in D$ for $j = 1, \ldots, l$ and $k_j = 1, \ldots, \alpha_j$.

2.1 A Keyed MapReduce Function

Let $K = (k_1, \ldots, k_\lambda)$ be a set of keywords associated with a MapReduce function (*we call K a common reference keyword set*). We also assume that $K \subseteq D$. A MapReduce function is called a keyed MapReduce if it projects a document d on K. That is, a stored document d is viewed as a matrix M of form (document$_j$: word$_{j,1}$, ..., word$_{j,\alpha_j}$) for $j = 1, \ldots, l$ below

$$\begin{pmatrix} d_1 : k_{1,1}, \ldots, k_{1,\alpha_1} \\ d_2 : k_{2,1}, \ldots, k_{2,\alpha_2} \\ \ldots \ \ \ldots, \ \ldots, \ \ \ldots \\ d_l : k_{l,1}, \ \ldots, \ k_{l,\alpha_l} \end{pmatrix}$$

where $k_{i,j} \in K$.

Now a keyed reduce function Reduce with input matrix M is invoked to generate inverted index $\hat{M}$

$$\begin{pmatrix} k_1 : d_{1,1}, \ldots, d_{1,\beta_1} \\ k_2 : d_{2,1}, \ldots, d_{2,\beta_2} \\ \ldots \ \ \ldots, \ \ldots, \\ k_\lambda : d_{n,1}, \ldots, d_{\lambda,\beta_\lambda} \end{pmatrix}$$

3 Trust Mining on MapReduce

We study trust mining on keyed MapReduce functions. To avoid leaking ids of users who attempt to post comments on the public bulletin-board, we allow users to access the public bulletin-board using shared random strings. We setup peer database by the standard incentive-based solution for sharing useful comments on the service provider.

3.1 Pseudonymous of Users

We assume that an auditor $\mathcal{O}$ holds a pair of public/secret keys (pk_R, sk_R) a CCA secure public key encryption scheme (say, the Cramer and Shoup's public-key encryption scheme [5] while a user is assumed to hold a shared τ-bit key k_U with $\mathcal{O}$. To access the bulletin-board, we assume that U and $\mathcal{O}$ runs the following authentication procedure

- $\mathcal{O}$ sends an authentication request together with a random string $a \in \{0,1\}^\lambda$ to a user U;
- Upon receiving the request and a challenging string a, the user U computes $C = \mathcal{E}_{pk_R}(k_U, a)$ and sends C to $\mathcal{O}$;
- $\mathcal{O}$ decrypts C to obtain (a, k_U) and checks that a is correct and that k_U is a valid shared key stored in $\mathcal{O}$'s database.

Vaudenay [14] has shown that the authentication protocol sketched above is secure in the presence of malicious adversaries assuming that the underlying public-key encryption is CCA secure.

3.2 Trust Mining

A user in our model is encouraged to provide comments, opinions and recommendations on a server S (in essence, it is an incentive-based mechanism. the research on how to reward the user is out of the scope of this paper and thus leaves this interesting problem to the research community). Such an opinion, comment and recommendation is called a document or a record.

When U tries to deduce the trust relationship with the server S, it invokes the auditor $\mathcal{O}$ who maintains the database of documents (in essence, the trust computing model is viewed as a trust computing-as a service model. Consequently, our trust computing model can be applied to the recently well developed Cloud Computing model as well). The trust mining procedure consists of the following three steps:

1. In the first step, a user U invokes the auditor $\mathcal{O}$ to collect all documents posted in the public bulletin-board

$$\text{NSet} = \begin{pmatrix} d_1^{(1)} & \dots & d_{s_1}^{(1)} \\ \dots & \dots & \dots \\ d_1^{(l)} & \dots & d_{s_l}^{(l)} \end{pmatrix}$$

where $d^{(i)} = [d_1^{(i)}, \ldots, d_{s_i}^{(i)}]$, a collection of records such that each record posted in the bulletin board posted by user U_i;

2. Let $K = (k_1, \ldots, k_\lambda)$ be a set of publicly known keywords selected that is maintained by keyed Map and keyed Reduce functions.
   Let $\cup_{i=1}^{l} d^{(i)} = \{d_1, \ldots, d_u\}$. The auditor $\mathcal{O}$ invokes a map function Map with input $\{d_1, \ldots, d_u\}$ to generate a matrix of form (record, key) below

$$\mathrm{M} = \begin{pmatrix} d_1 : w_{1,1}, \ldots, w_{1,t_1} \\ d_2 : w_{2,1}, \ldots, w_{2,t_2} \\ \cdots \ \ \cdots, \ \cdots, \ \ \cdots \\ d_u : w_{u,1}, \ldots, w_{u,t_u} \end{pmatrix}$$

   where $w_{i,j} \in K$.
3. The auditor $\mathcal{O}$ now invokes a keyed reduce function Reduce with input matrix M to generate inverted index $\hat{\mathrm{M}}$

k_1	$d_{1,1}, d_{1,2}, \ldots, d_{1,v_1}$
k_2	$d_{2,1}, d_{2,2}, \ldots, d_{2,v_2}$
$\ldots, \ldots,$	$\ldots, \ldots, \ \ \cdots$
k_λ	$d_{\lambda,1}, d_{\lambda,2}, \ldots, d_{\lambda,v_\lambda}$

   where $\{d_{i,1}, \ldots, d_{i,v_i}\} \subseteq \{d_1, \ldots, d_{u_k}\}$.

4 Trust Computing

In this section, a model for extracting auxiliary strings and computing trust scores in peer networks is proposed. Our extracting model is based on the Bayesian inference structure which in turn is instantiated by a look-up table generated by keyed MapReduce programs.

Let E be an event that among n comments on the target server S, m comments are positive (here we further assume that the keyword set divided into two sets: a positive keyword set and a negative keyword set). Let k be random variable defined over the set $k_{\hat{M}}$ ($=\{k_1, \ldots, k_v\}$). We now define a mapping ϕ: $k \in k_{\hat{M}} \rightarrow$ (positive, negative). Based on the mapping ϕ, one can compute the success probability p_M of an transaction from which $\Pr[E]$ is computed. More precisely, for each $k_i \in k_{\hat{M}}$, $\phi(k_i) =$positive if and only if $\{d_{i,1}, \ldots, d_{i,v_i}\} \subseteq k^+_{\hat{M}}$, where $k^+_{\hat{M}}$ defines set of documents such that $d \in k^+_{\hat{M}}$ if and only if $\phi(d) = 1$ (here we abuse the notation of ϕ defined over $k_{\hat{M}}$ and $\phi(d) = 1$ means that d is a pre-image of positive). The total number of $k^+_{\hat{M}} = m$. Let $n = v_1 + \cdots + v_\lambda$. Let $p_{\hat{M}} = m/n$. Given n, m and $p_{\hat{M}}$, one compute $\Pr[E] = p^m_{\hat{M}}(1 - P_{\hat{M}})^{n-m}$.

We now view the look-up table generated by the auditor $\mathcal{O}$ as an imaginary transaction generated by the target server S itself with some probability p (this probability is defined as the trust value of on the server S). Let P be an $(n+1)$th transaction. We now consider the probability that $(n+1)$th transaction is positive. That is, given Pr(E), we want to compute the maximum likelihood distribution $\Pr(P = p|\mathrm{E})$.

Notice that $\Pr(P = p \wedge \mathrm{E}) = \Pr(\mathrm{E}) \times \Pr(P = p|\mathrm{E})$ and $\Pr(P = p \wedge \mathrm{E}) = p \times p^m(1-p)^{n-m}$. Since $\Pr(\mathrm{E})$ is fixed, it follows that $\max_p \Pr(P = p \wedge \mathrm{E}) = \max_p \Pr(P = p|\mathrm{E})$. One can verify that when $p = \frac{m}{n}$, $\Pr(P = p \wedge \mathrm{E})$ reaches the maximum probability. We call $p = \frac{m}{n}$, the trust score of the target server S. Clearly, our mechanism used to compute trust score p is optimal for the given look-up table (an upper bound of trust values).

5 Conclusion

In this paper, a general trust mining method has been introduced and formalized in the context of keyed MapReduce and then a novel method for trust computing has been proposed in the context of the maximum likelihood estimations within the framework of the discrete Bayesian system. We have shown that our computing of trust score is optimal for a given look-up table.

References

1. Salehi-Abari, A., White, T.: Witness-Based Collusion and Trust-Aware Societies. CSE (4), 1008–1014 (2009)
2. Salehi-Abari, A., White, T.: The relationship of trust, demand, and utility: Be more trustworthy, then i will buy more. In: PST 2010, pp. 72–79 (2010)
3. Salehi-Abari, A., White, T.: Trust Models and Con-Man Agents: From Mathematical to Empirical Analysis. In: AAAI 2010, pp. 842–847 (2010)
4. Beth, T., Borcherding, M., Klein, B.: Valuation of Trust in Open Networks. In: Gollmann, D. (ed.) ESORICS 1994. LNCS, vol. 875, pp. 3–18. Springer, Heidelberg (1994)
5. Cramer, R., Shoup, V.: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
6. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI 2004, pp. 137–150 (2004)
7. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
8. Dean, J., Ghemawat, S.: MapReduce: a flexible data processing tool. Commun. CACM 53(1), 72–77 (2010)
9. Golbeck, J.: Computing and applying trust in web-based social networks, Ph.d dissertation, University of Maryland, College Park (2005)
10. Gudes, E., Gal-Oz, N., Grubshtein, A.: Methods for Computing Trust and Reputation While Preserving Privacy. In: DBSec 2009, pp. 291–298 (2009)
11. Gal-Oz, N., Yahalom, R., Gudes, E.: Identifying Knots of Trust in Virtual Communities. In: Wakeman, I., Gudes, E., Jensen, C.D., Crampton, J. (eds.) Trust Management V. IFIP AICT, vol. 358, pp. 67–81. Springer, Heidelberg (2011)
12. Shah, M.A., Baker, M., Mogul, J.C., Swaminathan, R.: Auditing to keep online storage services honest. In: Proceedings of the 11th USENIX Workshop on Hot topics in Operating Systems, HOTOS 2007. USENIX Association, Berkeley (2007)
13. Sabater, J., Sierra, C.: REGRET: reputation in gregarious societies. In: Agents 2001, pp. 194–195 (2001)

14. Vaudenay, S.: On Privacy Models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (2007)
15. Yahalom, R., Klein, B., Beth, T.: Trust-Based Navigation in Distribution Systems. Computing Systems 7(1), 45–73 (1994)
16. Yu, B., Singh, M.P.: A Social Mechanism of Reputation Management in Electronic Communities. In: CIA 2000, pp. 154–165 (2000)
17. Zhu, H., Bao, F., Liu, J.: Computing of Trust in ad-hoc networks. In: Leitold, H., Markatos, E.P. (eds.) CMS 2006. LNCS, vol. 4237, pp. 1–11. Springer, Heidelberg (2006)
18. Zhu, H., Bao, F.: Quantifying Trust Metrics of Recommendation Systems in Ad-Hoc Networks. In: Wireless Communications and Networking Conference (WCNC 2007). IEEE (March 2007)
19. Zhu, H., Bao, F.: Computing of Trust in Complex Environments. In: 18th IEEE Annual International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC 2007), Greece, September 3-7 (2007)
20. Zhu, H., Bao, F.: Private Searching on MapReduce. In: Katsikas, S., Lopez, J., Soriano, M. (eds.) TrustBus 2010. LNCS, vol. 6264, pp. 93–101. Springer, Heidelberg (2010)

Author Index