CESSA

Compositional Evolution of Secure Services using Aspects

ANR project no. 09-SEGI-002-01

State of the art and requirement analysis of security functionalities for SOAs

Abstract. This deliverable offers motivation and basic security properties and requirements for SOA applications. It defines the most important security properties and security requirements for the service and aspect models that are used as part of the project. These identified security requirements will help select the mechanisms for securing service composition considering both horizontal and vertical composition, i.e., the evolution of large scale SOAs involving many stakeholders, and the evolution of the local services and their orchestration. Those security requirements constitute a first abstract specification of the expected evolution features that will have to be satisfied at subsequent stages of the project.

Deliverable No.	D2.1
Task No.	2
Type	Deliverable
Dissemination	Public
Status	Final
Version	1.2
Date	23 Jul. 2010
Authors	Sabir Idrees, Yves Roudier, Davide Balzerotti (EURECOM); Gabriel Serme, Jean-Christophe Pazzaglia (SAP); Hervé Grall, Jean-Claude Royer, Rémi Douence, Mario Südholt (EM Nantes); Fred Rivard (IS2T)

Contents

Chapter 1

Introduction

1.1 Background

Service-oriented architectures (SOAs) are considered as advanced component based architectures for the construction of distributed systems. A service is a software application that can be located over a network, and whose interfaces and bindings can be defined, described, and discovered by using standardized interfaces and formats. Services support direct interactions with other software agents using message exchanges over the network via well defined protocols.

Such applications frequently are subject to stringent security requirements, for example, in order to protect company internal data, to avoid breaches of the right to privacy of clients, and to provide some traceability to official institutions for auditing purposes. Security properties generally pervade software systems; in more technical terms, security properties crosscut service based systems: security-relevant policies and implementations depend on and affect large parts of the underlying system.

SOA-related security is a rather recent field of research. Security in SOA has previously been addressed mainly from an application only point of view, in terms of enforcement of properties like authenticity or confidentiality of messages, in particular in early SOA middlewares (COSS and Jini), then with more acuity and by handling additional complexity in the case of Web Services. It is becoming apparent that the enforcement of security properties is the fundamental problem that has to be addressed and that it is an inherently crosscutting concern that we specifically plan to study within CESSA.

1.2 Purpose and Scope

This deliverable will provide a comprehensive treatment of security functionalities for SOAs in the presence of horizontal and vertical composition as well as evolution using aspects. This deliverable provides a comprehensive review of the security functionalities required in SOAs such as threats concerning service messaging, service coordination, the composition of security

functionalities, or the analysis and verification of security properties. Given a specification of security properties and requirements for a service oriented architecture, this deliverable helps to investigate the following problems:

- How to synthesize secure services to satisfy the security requirements?
- How to certify that the services synthesized effectively satisfy the security requirements?
- How to ensure that properties satisfied by a service-oriented architecture are preserved after evolutions using aspects?

1.3 Outline of the Document

This document offers motivation and basic security properties and requirements for the models and applications targeted within the project. Concretely, we present the following information. The first chapter provides background information, the purpose and scope of this document and ends with a summary of the key contributions of this deliverable. Chapter 2 provides an overview of generic security properties that are relevant for distributed SOA applications. Within Chapter 3, we introduce basic concepts relating to security policies and their static (security policies, static analysis), runtime (for instance, reference monitors) or post-mortem (verification of their conformance) enforcement. In Chapter 4, we summarize work related to the protection of the integrity of computation, in particular based on the software or hardware protection of the execution environment, or the protection of messages exchanged between entities. Chapter 5 focuses on key security mechanisms in SOA security models and infrastructures. The specific security vulnerabilities of Web Services are discussed within Chapter 6. Best practices to developing and auditing applications are presented in Chapter 7 together with a state of the art of approaches for extracting security requirements. Chapters 8 provides an analysis of the security requirements, considering different aspects for identifying security requirements such as, requirements originating from the attacker model, requirements originating from the deployment model (horizontal compositions), security requirements originating from the infrastructure (Vertical compositions), requirements originating from the use case, and requirements originating from the service model. Finally, Chapter 9 concludes this document.

Chapter 2

Security Properties

Before detailing the security engineering process, we introduce classes of security properties. The reference model for security objectives was defined in ISO 7498-2 and is still largely used. We follow a revised definition of security properties specified in the EVITA project [138], that is relevant for heterogeneous networks and embedded systems. The informal explanations below reflect how these concepts are generally understood.

2.1 Data origin authenticity

A data origin authenticity property applies to a quantum of information and a claimed author. The property is satisfied when the quantum of information truly originates from the author. The property can be made more specific by providing an observation of the quantum of information (defined, e.g., by a time and a location in the system). The author can also be constrained by adding a time and/or a place of creation of the quantum of information. Note that in most security oriented frameworks, data origin authenticity implies integrity.

2.2 Integrity

An integrity property applies to a quantum of information between two observations (defined, e.g., by a time and a location in the system). The property is satisfied when the quantum of information has not been modified between the two observations. It guarantees for instance that the content of a storage facility has not been modified between two given read operations, or that a message sent on a communication channel has not been altered during its journey.

2.3 Authorization

A controlled access property or requirement applies to a set of actions and/or information and a set of authorized entities. The property is guaranteed if the specified entities are the only entities that can perform the actions or access the information. The property can be further detailed

with time constraints on the period of authorization. Controlled access is needed to ensure that stakeholders only have access to information and functions that they are authorized to access as appropriate to their expected activities.

2.4 Freshness

A freshness property applies to a quantum of information, a receiving entity, and a given time. The property is satisfied if the quantum of information received by the entity at the given time is not a copy of the same information received by the same or another entity in the past. Ensuring freshness can be used to prevent replay attacks.

2.5 Non-repudiation

A non-repudiation property or requirement applies to an action and an entity performing the action. The non-repudiation of the action is guaranteed if it is impossible for the entity that performed the action to claim that it did not perform it. This property can be further detailed with a set of entities for which the action needs to be undeniable, with a time limit, etc. There may be specific legal requirements for non-repudiation. However, non-repudiation may also be introduced for convenience, for example, as an aid in providing evidence or proving liability.

2.6 Privacy

A privacy property or requirement applies to an entity and a set of information. Privacy is guaranteed if the relation between the entity and the set of information is confidential. This relation can however significantly vary. For instance, one generally distinguishes different types of privacy, typically anonymity, unlinkability, and pseudonymity.

- **Anonymity :** This is the property that the relation between an entity and its identity is strictly confidential. Privacy property must be made consistent with potentially conflicting requirements for identification, auditing, non-repudiation and jurisdictional access, which may require users to be identified and information about their interactions to be stored.
- **Unlinkability :** The unlinkability of two or more Item of Interest (abbreviated IOIs, e.g., subject, messages, actions, ... sent, received, or performed by the principal) from an attacker′s perspective means that within the system (comprising these and possibly other items), the attacker cannot sufficiently distinguish whether these IOIs are related or not.
- **Pseudonymity :** Pseudonymity refers to the capability of recognizing the same subject without being able to relate him to his identity [1].

[1] A pseudonym is an identifier, that is, a name or another bit string, generated in a fully independent manner from the subject and related attributes values, do not contain side information on the subject they are attached to.

2.7 Confidentiality

A confidentiality property applies to a quantum of information and a set of authorized entities. The property is satisfied when the authorized entities are the only ones that can know the quantum of information. Privacy relies on confidentiality and can be considered as a special case of confidentiality.

2.8 Availability

An availability property applies to a service or a physical device providing a service. The property is satisfied when some service is operational. Denial of service attacks aim at compromising the availability of their target. The property can be further detailed with the specification of a period during which the availability is required and of a set of client entities requesting the availability.

Chapter 3

Security Policies: from Specification to Enforcement

This chapter defines how security policies can be defined to describe security objectives as presented before. We in particular present the concept of a *reference* monitor that enables the runtime enforcement of a security policy. Since its introduction in the beginning of the seventies, it has been considered as an efficient principle to defend a computing system against a malicious user, as Irvine recalls [82]. We first describe this concept in a general and abstract manner. We then study how to implement reference monitors, and more generally, how to enforce a security policy using other approaches, namely the post-mortem conformance verification, which is based on traces of the execution, and the pre-execution static verification of the policy enforcement, at different levels of abstraction.

3.1 Security Policies

Security policies define what must be secure in system. This generally describes the control or information flow allowed in an application, and if distributed, the resulting constraints on communication protocols. Security policies also describe relevant parameters for security like typically keys, their length, algorithms, etc. Access control policies represent the most influential and studied type of security policy to date, and which we describe in the following.

Basic design principles : Access control systems are based on design principles, as first introduced by Salzer and Schroeder in [139], and then exposed plainly in [18]. Those principles derive from the general need for simplicity and restriction.

- **Principle of least privilege:** *The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task* [18]. In practice, most systems do not have the granularity of privileges and permissions required to apply this principle stringently, and designers only take it as a guideline.
- **Principle of fail-safe defaults:** *The principle of fail-safe defaults states that, unless a*

subject is given explicit access to an object, it should be denied to that object [18]. This principle responds to the issue of default permission. By default, a subject has no access to an object. Moreover, if a subject is unable to complete a task, it should undo its changes to the security state for the safety of the system.

- **Principle of economy of mechanism:** *The principle of economy of mechanism states that security mechanisms should be as simple as possible* [18]. When the design and implementation of a security mechanism are simple, there are fewer possibilities of errors.

- **Principle of complete mediation:** *The principle of complete mediation requires that all accesses to objects be checked to ensure that they are allowed* [18]. This principle restricts the caching of information. A software system that checks rights to an object each time a subject requests access mitigates the risk of granting elevated permissions by mistake.

- **Principle of open design:** *The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation* [18]. This principle is analogue to the absence of *security through obscurity*.

- **Principle of separation of privileges:** *The principle of separation of privileges states that a system should not grant permission based on a single condition* [18]. Systems and programs should be granted access to resources only when more than one condition is met. Checking access on only one condition may not be enough for strong security. If an attacker is able to obtain one privilege but not a second, he may be prevented from launching a successful attack.

- **Principle of least common mechanism:** *The principle of least common mechanism states that mechanisms used to access resources should not be shared* [18]. This principle aims to limit the sharing of mechanisms used to grant access to a resource. Sharing resources may also provide a channel along which sensitive information may be transmitted, which should be avoided.

- **Principle of psychological acceptability:** *The principle of psychological acceptability states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present* [18]. Configuring and executing a program should be as easy as possible, output should be clear, useful, and imparting no unnecessary information.

Access control techniques : Access control policies can be grouped into two main classes as either discretionary or mandatory. Alternative approaches are role-based access control, rule-based access control, or domain type enforcement.

- **Discretionary access control :**Discretionary access control (DAC) allows the owner of the information to decide who can read, write, and execute a particular object. DAC is based on the idea that the owner of the data should determine who has access to it. DAC implementations include identity-based access control and access control lists (ACLs).

Identity-based access control makes object access decisions based on the user ID or the user group. ACLs allow groups of objects or subjects to be controlled together, which makes administration easier. Identity is the key in DAC: the owner states the constraint in terms of the subject identity (or the owner of the subject). The primary benefit associated with the use of DAC is enabling fine-grained control over protected objects. DAC implementation is intuitive and mostly invisible to users; it is also flexible for environments that do not require stringent object security. However allowing users to be in charge of object access control may expose the system to more threats, like Trojan horses for instance. DAC allows data to be freely copied from object to object, so even if access to the original data is denied, access to a copy can be obtained. The lack of constraints on the information copying makes system maintenance and verification of security principles difficult in DAC.

- **Mandatory access control :** Mandatory access control (MAC) is a technique where access is determined by the system, not the owner. Administrators and overseeing authorities predetermine who can access an object. Typically, they will check information associated with both the subject and the object to determine whether the subject should access the object. Rules describe the conditions under which access is allowed, MAC is occasionally called rule-based access control. MAC assigns security labels (level of sensitivity) to objects and subjects, limiting access across labels. Examples of security labels are unclassified, sensitive but unclassified (SBU), confidential, secret, and top secret; more examples of classification with their description can be found in [147]. Security labeling confers MAC a focus on information confidentiality. MAC is often associated with the Bell-LaPadula confidentiality model. The Biba model developed a similar method, but rather aims at providing information integrity. MAC is relatively straightforward and relatively more secure than DAC. However, to prevent dynamic alteration of the underlying policies, it requires large parts of the operating systems and associated utilities to be trusted. So, MAC is expensive to implement due to the reliance on trusted components. Moreover, MAC has a tendency to over-classify objects, incurring difficult declassification issues for programmers.

- **Role-based access control :** Role-based access control (RBAC), or task-based access control, requires that access rights be assigned to roles rather than to individual subjects. Subjects obtain these access rights by virtue of being assigned membership in appropriate roles. This simple idea greatly eases the administration of authorizations. Although it does not provide fine-grained privileges, RBAC integrates support for the principle of least privilege, for the separation of duties, and for the central administration of role memberships and access control. A role may look like a group, but have several differences. A user is generally assigned to one role, but can be member of multiple groups. A group corresponds to a specific type of user association; however a role represents general tasks a user can perform.

- **Domain type enforcement :** Domain type enforcement (DTE) is an access control technology that restricts subject accesses according to a specific security policy. DTE is an

extension of Type Enforcement (TE) and is itself extended into Dynamic Typed Access Control (DTAC). Implementing TE gives priority to MAC over DAC. Access clearance is first given to a subject accessing objects based on rules defined in an attached security context. A security context in a domain is defined by the domain security policy. The principle of TE is to assign objects to types, columns in the access control matrix thus being replaced by types. The DTE extension assigns subjects to domains and completes the matrix transformation so the access control matrix is now a domain definition table (DDT) with rows of domains and columns of types. When a subject requests access to an object, the subject domain is compared with the type of the object. The attempt is denied if the requesting subject domain does not include a right to the requested access mode for that type. DTE is more flexible in terms of security policy definition, and gives a finer grained description of the policy.

3.2 Reference Monitors: Definition and Applications

The concept of a reference monitor can be easily expressed at an abstract level. First, there are *processes*. Second, there are shared *resources*, to which processes can make *references* to perform operations. Finally, a reference monitor validates every reference to resources made by a process against a security policy. Generally speaking, a security policy is a specification of the references allowed to processes at any given moment. For instance, consider a file as a shared resource. In a process, a reference to the file can be a call to a function performing an open, read, or write operation on this file. If the process is not authorized to read the file, the reference monitor does not validate an invocation of a read operation for the file each time it is performed by the process. A reference monitor like this can be implemented either at the level of the operating system, or at the level of an application programming interface.

In order to be secure, the validation mechanism of a reference monitor must satisfy the following requirements.

Tamper-proofness The validation mechanism must not be modified by processes. This condition prevents a malicious user from modifying the validation behavior, for instance to approve operations that are disallowed by the security policy. Very often, the tamper-proof requirement covers not only the validation mechanism but also the security policy. Indeed, otherwise, a malicious user could gain unauthorized access by modifying the security policy enforced by the reference validation mechanism.

Mediation completeness The validation mechanism must mediate all security-sensitive operations performed by processes. In other words, there is no covert channel allowing resources to be accessed without a validation by the reference monitor.

Verifiability The validation mechanism must be small enough to enable practical verification of correctness. The validation mechanism is correct if it correctly check the validity of the references performed by processes against the security policy and correctly implements the resultant decisions.

Thus, a reference monitor must be tamper-proof, always invoked and verifiable.

We now project this general model to the service model as described in Task 1 [42].

3.2.1 Reference Monitors in the Service Model

We first briefly recall the service model, as described in the first deliverable of Task 1 [42, Sect. 2.1]. Second, we extend the model in order to account for reference monitors.

The service model is based on three layers, corresponding to collaborations, processes and services. Each process is under the control of a peer. Processes can collaborate, by exchanging messages. A collaboration can involve different peers, when the messages exchanged cross peer boundaries. A message corresponds to the request of a service or to the subsequent reply. Services are provided by server processes and required by client processes. Precisely, a process can invoke a service provided by another process by sending a request: it therefore requires the presence of the service to execute.

We propose a unique addition to the service model to account for reference monitors: *resources*. A process can own *resources*, which are components of its state. The operations declared in a service can therefore act over resources of the associated process. In the resource-oriented model [42, Sect. 2.1], the operations correspond to four standard operations, called CRUD (Create, Request, Update, Destroy), directly used to manipulate resources. In the process-oriented model [42, Sect. 2.1], the operations correspond to any computation over resources. A security interpretation of these operations is therefore needed: it assigns to each operation its meaning related to the manipulation of security-sensitive resources. Thus, the validation mechanism of a reference monitor must integrate this interpretation in order to satisfy mediation completeness: when a service is invoked, the reference monitor must determine the references to sensitive resources that are caused by the service invocation.

In the context of service-oriented computing, at the abstract level that we consider, a *security policy* can be considered as a security protocol: it is a specification of the actions allowed over resources that are security-sensitive. A security policy can be not only local, at the level of a process, but also global, at the level of a collaboration. In the former case, with a local policy, the reference monitor is implemented in a centralized way. In the latter case, with a global policy, it is implemented in a distributed way. Distribution is especially needed when the collaboration involves different peers. The security policy then becomes a security contract taken by peers. The contract specifies the global protocol that the collaboration must follow and by projection, the local protocols that each peer must enforce.

3.2.2 Classification of Enforceable Security Policies

What kind of security policies can reference monitors enforce? A security policy can be formally defined as a protocol, *i.e.,* a set containing reference sequences that are allowed. Alternatively, it can be defined as a predicate over reference sequences, called a detector [67, Sect. 2.2]. A reference monitor enforces a security policy if it generates after control sequences in the security policy.

Since a reference monitor can make different actions following the result of the validation of a reference, the answer to the question depends on the action power. Traditionally, reference monitors have been used to enforce access control. A limited power is then sufficient: if the monitor validates the reference that it has intercepted, then it allows the reference to be executed and the process to continue; if it does not, then it interrupts the process. Formally, such an interruptive reference monitor enforces a *safety* property, as shown by Schneider [140]: this simply means in the finitary case that the property is prefix-closed. Since any sequence property is the intersection of a safety property and of a liveness property, as noticed by Alpern and Schneider [7], an interesting issue is to extend the action power of monitors to ensure properties beyond safety, in other words properties with a liveness component. For example, in the security field, processes must often satisfy some availability requirement: a paradigmatic instance is a property like "any user request must be responded", corresponding to a request-response protocol. Since a property like this is not prefix-closed, it contains a liveness component.

Recently, Bauer, Ligatti and Walker [97, 98, 99] have proposed a way to extend the action power of reference monitors. Whereas interruptive reference monitors can be described by standard automata, they have described reference monitors as special automata, called "edit automata". These automata mainly bring the ability to perform *insertions* and *suppressions* into the input sequence of references to be executed. Modeled as an edit automaton, a reference monitor defines a transformation of reference sequences: each sequence of references to be executed is transformed into a sequence of executed references by the reference monitor. Not all the transformations are meaningful. First, a transformation needs to be *monotone*, which expresses the continuous working of the reference monitor. Second, with respect to the property to enforce, a transformation must be *sound*, by ensuring that output sequences belong to the property, and *neutral* (or transparent), by preserving the input sequences that already belong to the property. Bauer et al.l [98, 99] have exactly characterized the enforceable properties: thus, they have shown that an edit automation can enforce not only a safety property, but also a property with a liveness component.

3.3 Approaches to Policy Enforcement

In the section, we describe different techniques that are used to enforce security policies.

As described above, reference monitors provide runtime monitoring, which implies that they control a unique execution. There are two main techniques for implementing monitors: runtime monitoring and program rewriting. In the following, we consider both strategies as instances of aspect-oriented programming. For some security policies, like information flow control, controlling a unique execution is not sufficient [67, Sect. 2.2]: the enforcement of these policies requires the analysis of all possible executions. For instance, suppose two resources, A and B, each one having two states, a_1 and a_2, and b_1 and b_2, respectively. An information flow policy could be: there is no information flow from A to B. Assume to simplify that a process use resource A as input and resource B as output. We can formally characterize the enforcement property associated to the policy as follows: the process enforces the security policy if for any pair of executions starting with resource A in state a_1 and a_2 respectively, both executions terminate with B either

in state b_1, or in state b_2. In other words, it is impossible to observe through the state of B a variation in the state of A. With a policy like the preceding one, it is impossible to limit to a unique execution. Thus, we also explore an alternative for enforcing security policies: static analysis.

This alternative could seem to question the universality of reference monitors. However, since static analyses can generally be formulated as abstract executions, the concept of reference monitors can be extended in order to encompass not only runtime monitoring but also static analysis. For a formal comparison between runtime monitoring, program rewriting and static analysis, the reader can refer to the work of Hamlen, Morrisett and Schneider [67], which determine the class of security policies enforceable by each technique.

3.3.1 Aspect-Oriented Programming and Security

Security concerns frequently crosscut the structure of the base (*i.e.,* unsecured) application. Indeed, a security policy must frequently be applied at different events during the execution of an application, *e.g.,* each time a resource is accessed. The implementation and evolution of applications that are subject to crosscutting security functionalities is therefore very cumbersome and highly error-prone. Aspect-Oriented Programming (AOP) methods and techniques [96, 4] are prime candidates to tackle these problems: AOP is now considered as the field of the systematic treatment of crosscutting functionalities.

In the following, we first study how AOP has been used to monitor programs, then consider AOP for security in a distributed context, specifically in service-oriented computing. Finally, we conclude by mentioning the main problem raised by the use of AOP: the preservation of properties.

Monitoring-Oriented Programming Runtime monitoring is akin to runtime verification, a relatively new formal analysis approach that aims at combining testing with formal methods[1]. With runtime verification, specifications, expressed formally and referring to temporal behaviors and histories of events or actions, are checked at runtime against the current execution of the program, rather than statically, against all hypothetical executions. Runtime verification has already been applied to web services [66].

Adding to verification recovery actions, Chen, Roşu and others have proposed Monitoring-Oriented Programming (MOP) [29, 30]. First, properties to be enforced are specified in a formal language, like a temporal logic. Then, monitoring code is automatically generated from properties and integrated into the original program. It aims not only at detecting the violations of the properties, but also at reacting to violations, by producing reports or recovery actions. MOP can be regarded as a specialized instance of AOP, in which aspects are formal specifications instead of modules of ordinary code. Thus, MOP can be implemented by using AOP tools: first, from the specification of properties and actions, aspects are generated by a MOP compiler, second the AOP tool is used to weave generated aspects in the base program. Note that for weaving, AOP uses a mix of program rewriting and runtime monitoring.

[1] See the series of workshops "Runtime Verification" from 2001 to 2009, becoming a conference in 2010.

AOP for Security in a Distributed Context In a distributed context, and specifically in service-oriented computing, there are currently still very few results on the enforcement and preservation of security properties: one major goal of the CESSA project is to leverage the existing body of results on the use of aspects for security to service-oriented computing.

In component-based systems, which are often used to implement services, this problem is often tackled using (object-oriented) framework that provides a restricted set of means to implement security policies. For instance, the Enterprise JavaBeans architecture [44] allows access control policies to be implemented based on an interceptor mechanism provided by the execution environment of components, the so-called container. The corresponding security models are, however, often rather simplistic and the means for addressing crosscutting security requirements very limited. For instance, in the case of Enterprise JavaBeans, the provided security model is a simple role-based access control one and the mechanisms to tackle crosscutting requirements are restricted to the use of annotations, the assignment of different stakeholders to common roles and the inheritance of access rights from a calling to a called method. However, these three mechanisms only enable simple crosscutting security requirements to be modularized.

Aspect-Oriented Software Development (AOSD) [96, 4] has been applied to service-oriented systems (mainly based on web services) and also to the modularization of crosscutting security policies for sequential and distributed systems (including few work on services however). In the remainder of this section, we briefly present a number of major approaches applying AOSD methods to services and to security issues in distributed systems.

Several proposals for mixing aspect-oriented programming and service-oriented architectures have already been made. For instance, Courbis and Finkelstein [37] propose to weave aspects into web service orchestrations. Service orchestrations, in particular using the language BPEL, have been extended with aspect support by Charfi and Mezini [28]. Aspects have been used to modularize and transparently add Quality of Service (QoS) properties into service-oriented systems, especially web services, notably by Baligand, Ledoux et al. [10, 11] and Ortiz and Bordbar [126]. Furthermore, aspects have been used to facilitate web service compositions, notably by Charfi and Mezini [27] and Benavides, Südholt et al.l [15]. Finally, several other references and additional information on state-of-the-art approaches for AOSD and services are presented in Section 4.3 of the CESSA deliverable D1.1 [42].

Since security is one of the most important crosscutting functionalities in large-scale distributed systems, especially business information systems, a number of aspect-oriented methods and techniques have been applied to the modularization and transparent implementation of security properties. Relevant approaches use aspects to derive and specify high-level security requirements, provide guarantees for the secure executions of large-scale distributed infrastructures in the presence of aspects, define aspect-based language support for the modularization and implementation of security mechanisms, and design calculi for a foundational purpose.

As to the use of aspects for the specification of security requirements and policies, Hayley et al. [64] present how to derive security requirements from crosscutting threat descriptions. Xu et al. [177] use aspects to specify security requirements from threat descriptions. Jones and Hemlen [87] have investigated how aspect-oriented security policies can be defined unambiguously.

The use of aspects for security purpose and their impact on security properties of large-scale

distributed systems have been studied, in particular, by Cannon and Wohlstadter [23] who show how to enforce security in client-server based web systems using authority aspects. Furthermore, De Borger et al. [39] present a permission system for secure AOP. Their system permits to shield existing applications from the effects of untrusted aspects and has been applied, *e.g.,* to the evolution of an FTP server.

Support for security mechanisms at the level of aspect languages has been provided, among others, in form of the so-called dataflow pointcut [6] that allows security properties to be defined in terms of dataflow dependencies. Fradet and Hong Tuan Ha [56], similarly, exploit aspects defined on the execution history of a program to enforce timed properties to prevent denial of service attacks.

Finally, basic calculi for security have been studied, *e.g.,* the lambda SAOP calculus [5].

Correctness of Aspects and Aspect Weaving In general aspects can arbitrarily modify the semantics of an existing application to which they are applied. Safety and liveness properties of the base system are therefore frequently not preserved in a woven system, see Djoko Djoko et al. [41]. This state-of-affairs, which is particularly problematic as far as security properties are concerned, has fostered a rich body of work on formal semantics and properties of aspect-based systems. In the remainder of this section, we focus on results for the preservation of formal properties in the context of aspects (more general references on the semantics of aspects, especially history-based ones and aspects that preserve whole classes of properties, are presented in Section 4.3 of the CESSA deliverable D1.1 [42]).

Several research groups have studied the semantic impact of aspects and devised manual or automatic proof methods for the preservation of properties. For instance, Barthe and Kunz [13] rely on code certificates in order to manually prove that aspects preserve specifications. Finally, Katz et al. [95, 94] have shown how to automatically prove certain properties expressed using temporal logics, notably LTL, using model checking techniques.

3.3.2 Analysis of Access Control Policies

Policy verification may be required in order to check the conformance of the policy with respect to what the designer thinks, as a form of enforcement at a high level of abstraction.

The purpose of access control systems is to make the operating system secure while allowing a flexible definition of security policy. In this regard, a tool for checking the access control policy is useful as it enables the verification of complex security policies and is thus close to real-world concerns. Simple models like RBAC, which may already require some verification based on the concepts of role and delegation, are generally combined with other models like DTE in real-life systems. This is typically the case for systems like SELinux (Security Enhanced Linux), or in systems that define a multi level security (MLS) policy, as can typically be expected in embedded systems/. The analysis of such policies is much more complex in that case, in particular because the verification of type conformance refers to a finer-grained control over low-level operations.

Usual goals of policy verification are to detect the unauthorized transitive flows of information including *overt channels* for example relating to communications mentioned in the policy.

The analysis of overt channels in particular aims at finding indirect flows between areas of different security or trust levels as defined in the policy. Tools like the Apol verifier [107] have for instance been defined to perform such verifications in SELinux, due to the complexity of the mandatory policies that have to be defined in that system. However, the analysis conducted on the security policy cannot lead to the discovery of *covert channels*, for instance based on communication between processes through memory paging, that would require the analysis of the code of the processes and of the underlying execution environment or operating system.

3.3.3 Static Analysis

In the past ten years, researcher have largely studied the use of static analysis techniques to automatically find software vulnerabilities [168, 167, 54, 114, 8]. A general overview of different static analysis approaches applied to computer security is presented in [31]. In the following, we summarize some of the more important contributions in three main categories.

Type-Based Analysis For typed programming languages, information about the taint status of variables can be propagated through the program by extending the type system of the language. For example, CQual [54] is a tool that allows one to extend the type system of the C language with user-defined qualifiers. After defining the new type system, the programmer manually introduces the additional qualifiers at a few key points in the application. CQual's qualifier inference then determines whether the program contains a type error under the extended system. This technique was used by Shankar et al. [144] for the detection of format string vulnerabilities, and by Johnson and Wagner [86] to identify user/kernel pointer bugs in the Linux kernel. Analogously, Zhang et al. [180] discovered security problems regarding the placement of authorization hooks in the Linux Security Modules framework.

JFlow [114] is an extension to the Java programming language that adds a type system for tracking information flow. In this system, the user is provided with annotations (labels) that define restrictions on the way in which the information may be used in the program, permitting the verification of information confidentiality and integrity. JFlow supports a wide range of language features (such as objects and exceptions), and is implemented in the Jif [84] tool.

Rule-Based Bug Finding In [45], Engler et al. present meta-level compilation, a technique for the translation of simple user-defined rules (such as "never use floating point in the kernel") into extensions for the C compiler. During the compilation of a program, these extensions are able to determine whether the program violates the specified rules. An automated extraction of such program rules from a given application is described in [46]. In [8], the authors use the system to detect potentially dangerous accesses to user-supplied, unchecked values in Linux and OpenBSD.

Web Application Analysis There exist several approaches that are focused on the detection of "taint-style" vulnerabilities (such as XSS or SQL injections), which frequently occur in web applications. Huang et al. [72] adapted parts of the techniques used in CQual to develop an

intraprocedural analysis for PHP programs. In [73], the same authors present an alternative approach that is based on bounded model checking. Whaley and Lam [172] described an interprocedural, flow-insensitive alias analysis for Java applications. Their analysis is based on binary decision diagrams, and was used by Livshits and Lam [102] for the detection of taint-style vulnerabilities. Finally, Pixy [88, 89] is an open source static PHP analyzer that uses taint analysis for detecting XSS vulnerabilities. The same tool was also extended to analyze the quality of the sanitization routines adopted in real world web applications [12].

In [65], the authors applied the Java String Analyzer by Christensen et al. [32] to extract models of a program's database queries, and used these models as the basis for a runtime monitoring and protection component for SQL injection attacks. Xie and Aiken [176] presented an interprocedural and flow-sensitive system for the discovery of SQL injection vulnerabilities through a bottom-up analysis of basic blocks, procedures, and the whole program. In their work, the authors take into account the effect of applying one of a number of regular expressions to an input value. In principle, the authors manually specify a list of regular expressions that simply extends the list of built-in sanitization routines (such as `htmlentities`).

Chapter 4

Protecting the Integrity of Computation

4.1 Software Based Execution Environment Protection

This section describes language based security approaches to protecting the environments where applications are executing against attacks in the software trying to subvert security mechanisms. These approaches rely on safe typing but may also combine such formal method based mechanisms with virtualization, as typically illustrated by the Java Virtual Machine.

4.1.1 Java Security Architecture

Java provides an extensive security architecture to allow the network environment (network computers) to share and distribute the processing in a secure way such as protecting the end user from hostile programs downloaded across the network from untrusted sources. This security model is based on the *Sandbox* mechanism, in which untrusted programs can be loaded, to restrict the behavior of the untrusted code. ***Java Sandbox Model***: The Sandbox creates a virtual environment where to run the program and restricts its access to only those resources which are allowed by the security policy. In other words, the sandbox provides a very restricted environment to run untrusted code in which it can do anything without consequences outside the sandbox boundary [158]. To restrict the untrusted code, the original sandbox model relies on code signing and authentication mechanisms. These mechanisms provide a way to restrict the resources allowed for some code based on its digital signature by trusted parties. The fundamental components responsible for the enforcement of the Java sandbox model are (i) the class file Verifier, (ii) the Class Loader Architecture, (iii) the Security Manager, and (iv) safety features built into the Java Virtual Machine (JVM) [164].

- **Java bytecode verifier :** The bytecode verifier is an internal part of the JVM class loader that ensures that the structure of Java class files is in conformance with the Java language with respect to various criteria like for instance array bounds checking. In terms of resources, the bytecode verifier helps enforcing memory protection for all Java programs [120] (see Figure 4.1). The bytecode verifier ensures that object casting is legal and no attempt at illegal casting is made. The bytecode verifier also proves that there are no

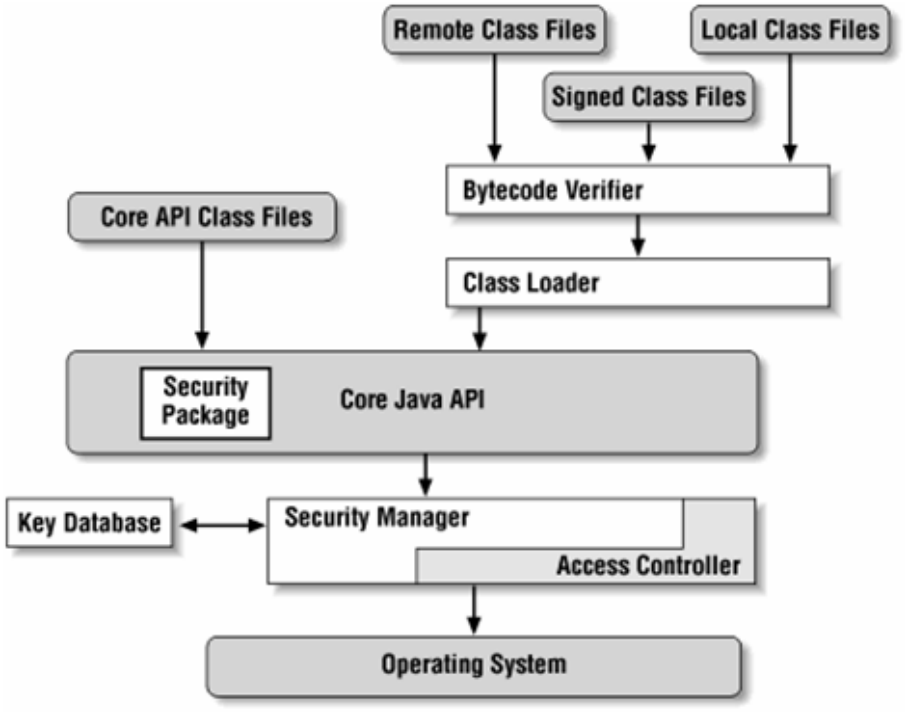

Figure 4.1: Anatomy of a Java application [120]

operand overflows or underflows. The bytecode verifier may delay some of the tests until the real program execute such as tests to ensure the validity of field and method access that are performed at execution time only. However, the bytecode verification is not used on all classes. All remaining rules and policies are enforced at run time by the JVM [120]. The class loader in turn *defines the behaviour of converting a named class into the bits responsible for implementing that class* [109].

- **Java Class Loader :** The class loader operates in three ways with respect to security: (i) it cooperates with the virtual machine to define class namespaces, (ii) it cooperates with the security manager to define appropriate permissions and (iii) it sets up the mapping permissions to class objects [120]. The class loader ensures that some malicious code is not interfering with honest code by defining with the virtual machine a class namespaces that would not overload an existing one. The class loader also categorizes the code in a particular protection domain according to its origin, which determines the policy applied to it. A class file verifier checks, mostly statically, the integrity of the bytecodes loaded into the virtual machine in order to prevent the JVM from being subverted [164]. The class file verifier operates in four distinct ways: it performs (i) structure checks on the class, (ii) semantic checks on the Type data, (iii) bytecode verification, and (iv) a verification of symbolic references.

- **Java Virtual Machine :** The third fundamental component used in the sandbox model is the Java Virtual Machine (JVM) itself. The JVM is a virtual environment for the execution

of other computer programs and scripts that can run multiple programs as separate threads [158]. The JVM provides several features to render a Java program robust and secure such as type safe reference casting, structured memory access, automatic garbage collection, array bounds checking, checking references for null [164]. The inner architecture of the JVM includes subsystems (class loader, execution engine), memory areas (Method area, Heap, Java Stacks, PC registers, Native method stacks, etc.), data types, and instructions [164]. The JVM execution engine provides the mechanism to execute the instructions contained in the methods of loaded classes. The JVM maintains several memory areas (runtime data areas) to store information such as bytecodes, and other information extracted from loaded class file. The JVM uses memory areas to store the bytecodes and garbage collections heaps are used to store the object created by the running programs. On creation of every new thread, The JVM decides where in the memory to put the object and where to put the corresponding data. The JVM provides structured error handling mechanisms, and whenever there is some security violation, it typically throws an exception or error to kill the offending thread.

4.1.1.1 Java Security Manager

The fourth and last component of the Java sandbox model is the security manager, which provides the mechanisms to (i) define security policies and (ii) enforce them [164]. The security manager serves as the central point for access control and defines the outer boundary of the sandbox as shown in figure 4.1. The Java sandbox provides the capability to customize both the class loader and security manager according to organizational needs. Customized security policies can be defined in separate files that grant permissions to specific internal code sources. Permissions are object-oriented and defined as subclasses of the java.security.Permission class [164]. Policies can be assigned to code from an external origin too by checking its digital signature and after verifying that the code has not been altered or corrupted since it was signed. The security manager is in essence a monitor verifying that an object can access to some resource at runtime, which explains how it enforces a given security policy. It is also responsible for handling privileged operations, which are necessary for accessing some protected resources like typically I/O resources.

4.1.1.2 Java Stack Inspection

As mentioned above, a specific case has to be handled: *In the Java security model, access control decisions are taken by examining the call stack at run-time. Permission is granted, if it belongs to all principals on the call stack. The so-called privileged operations are an exception. These are allowed to execute any code granted to their principal, regardless of the calling sequence. This access control mechanism is known as stack inspection* [14].
Stack inspection provides the controlled run-time mechanisms to all components based on their level of trust. For example, each time, when trust code access protected resources, stack inspection is invoked to determine the appropriate permission. There are four fundamental primitives necessary to use stack inspection: (i) enable- Privilege(), (ii) disablePrivilege(), (iii) checkPriv-

ilege(), and (iv) revertPrivilege() [169] . These primitives restrict the resources and define the specific privileges to access the system. Another concept that has been suggested for code protection but not implemented in Java is that of Proof Carrying Code (PCC). PCC ensures that the code/program supplied by untrusted source is safe to execute. As explained in [16] *the code supplier is required to provide an encoding of a proof that his or her code adheres to the security policy specified by the code consumer*. Such a proof would suppress the need for security manager and for its checks at runtime. However, while proofs of concept have been produced in assembler for instance, automatically generating a proof for a Java program still is an open issue.

4.1.2 Language Based Security

According to Schneider, Language Based Security (LBS) is *a set of techniques based on programming language theory and implementation, including semantics, types, optimization, and verification, brought to bear on the security question*. In turn, Kozen suggested to refer to LBS as *the idea of retaining extra information from a program written in a high-level language in the object code compiled from it. This extra information - call it a certificate - is created at compile time and packaged with the object code. When the code is downloaded, the certificate is downloaded along with it. The consumer can then run a verifier, which inspects the code and the certificate to verify compliance with a safety policy.*

This approach has given rise to a series of tools in the last ten years, of which we give an example underneath. These tools have often (but not always as we show below) been developed around languages with well-formed semantics or types.

4.1.2.1 Java Information Flow - JIF : analyzing information flows

Jif (which is short for Java + Information Flow or JFlow) is a security-typed programming language that extends the Java language by defining information flow control, exceptions, subclassing, and access control enforced both at compile time and at run time [115]. Its first aim is to make it possible to define legal information flows in a Java program. Jif treats the static checking of flow annotations as an extended form of type checking and prevents information leakage by storage channels. Several new features such as decentralized label model, label polymorphism, run-time label checking and automatic label inferences are introduced in this extension. In the decentralized label model, sets of security policies (labels) are attached with the data in order to restrict its exchanges. In Jif, variables are statically bound to static labels denoted by Label Expressions. Every label type has two parts: (i) an ordinary Java Type and (ii) a Label part. Ordinary java types include int, double, float, etc. whereas the label specify the ways a value can propagate. Type checking ensures the static type of each expression produced at run time, in particular by checking against its super type. In contrast, label checking ensures that the apparent label of each expression is at least as restrictive as the actual label of every value it might produce [115]. In order to prevent information leakage (implicit flows), Jif deals with both static entities and run-time labeling (first-class value (run-time label) of the new primitive type). Runtime labeling is required when the label of value cannot be determined statically. As explained

in [115] *The important power that run-time labels add is the ability to be examined at run-time, using the switch label statement*. Run-time labels can also be determined statically and treated as fixed labels. Jif further defines the authority (to act for some set of principals) and declassification (controls the ability to declassify data) properties, which can be checked both statically and dynamically. More complex access control mechanisms can then be incrementally defined. Jif extends the Java Class and interface declarations to include an optional set of parameters related to the information flow [115]. While Jif makes it possible to reason about memory based information leaks, it however does not eliminate all possible covert channels and information leaks such as Timing channels, Threats, Finalizer, Resource exhaustion, Wallclock timing channels, Unchecked exceptions, and Backward compatibility.

4.1.2.2 CCured

CCured represents another trend of LBS which consists in the a posteriori analysis of a program. According to [117], *CCured is a source-to-source translator for C. It analyzes the C program to determine the smallest number of run-time checks that must be, inserted in the program to prevent all memory safety violations*. The major feature provided by CCured is the analysis of pointers and arrays used in a C program. As defined in [34], the fundamental concept in CCured

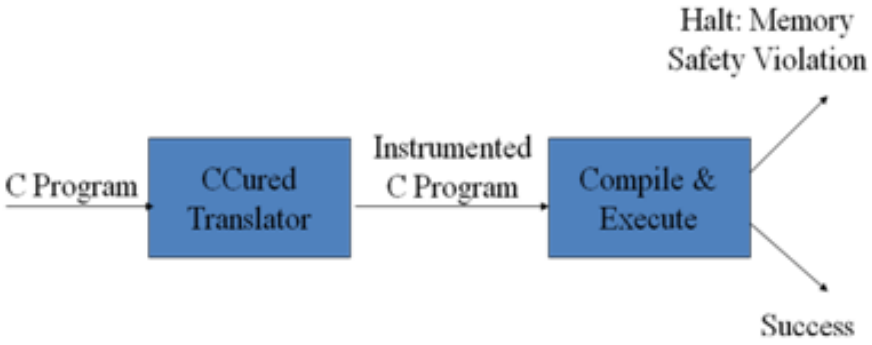

Figure 4.2: The CCured System [117]

is that of pointer qualifier. The CCured language distinguishes different kind of pointers: (i) SAFE (ii), SEQ, and (iii) WILD. SAFE pointers are considered as the easiest ones, which cannot be subject to arithmetic or (arbitrary) casts. SAFE pointers to type T can be defined as *either 0 or else it points to a valid area of memory containing an object of type T. Furthermore, all other pointers to the same area are also SAFE and agree on the type T of the stored object* [117]. A SEQ pointer can be used in pointer arithmetic but not in type casting. Whereas a WILD pointer can be dereferenced and can be subject to pointer arithmetic but can only be cast in very restrictive ways [117]. The overall architecture of the CCured system is shown in Figure 4.2. CCured is implemented on top of the C Intermediate Language (CIL) infrastructure, which means that C programs are first translated into a subset of the C language that has simple semantic rules. These C programs are passed from the CCured translator (cf. 0), where C expressions and statements are separated into expressions, instructions, and statements. The

CCured Translator moves all the type and structure declarations into the beginning of the file. The scopes of the variables are resolved, the local variables renamed, and all local variables moved to function-scope. The CCured Translator changes or converts all the implicit casts and conversions into explicit casts. This instrumented C program/CCured program can be further compiled and executed. The instrumented programs can either run successfully or generate memory halt safety violations. In case such a safety violation occurs, the program terminates immediately [117]. In [34], different extensions to the original CCured are proposed in order to apply it to real world security-critical networks. These extensions cope with incompatibility issues (upcasts, downcasts) between the CCured type representations and standard C representation used for precompiled libraries. These issues are solved by introducing the physical subtyping mechanism and by run-time type information mechanisms respectively. Furthermore other incompatibility issues are also solved in [34] by introducing a new representation for pointers.

4.2 Hardware Based Execution Environment Protection

In order to deploy the overall security architecture, several security mechanisms need to be used and coordinated. This section reviews a certain number of technologies available for protecting secrets or an execution platform based on the hardness of breaking hardware based security mechanisms.

4.2.1 Trusted Computing

The Trusted Computing technology as proposed by the Trusted Computing Group (TCG) [62] provides a set of basic security components and functionalities (e.g., isolated encryption) that form the basis for a larger set of high-level security functions (e.g., platform integrity attestation) that can be built upon. Together with a secure operating system, Trusted Computing can be used to improve security especially for distributed and embedded applications that are executed in hostile environments. The main TCG specifications are: (i) a component providing cryptographic functions called the *Trusted Platform Module* (TPM), (ii) a kind of (protected) pre-BIOS (Basic I/O System) called the Core Root of Trust for Measurement (CRTM), and (iii) the Trusted Software Stack (TSS), which is the software interface to provide TC functionalities to the operating system and the application layer. The TPM is primarily used as a root of trust for integrity measurement and reporting and to secure all critical cryptographic operations. It provides the following features:

- a hardware-based (i.e., physical) random number generator (RNG),
- a cryptographic engine for encryption and signing (RSA),
- a cryptographic hash function (SHA-1, HMAC),
- a read-only memory (ROM) for firmware and certificates,
- a volatile memory (RAM) for secure on-chip processing,

- a non-volatile memory (EEPROM) for durable internal storage of secret keys, monotonic counter values and authorization secrets, and optionally, sensors for tampering detection and corresponding tamper-response mechanisms.

The Core Root of Trust Measurement (CRTM) represents a small immutable code implemented into the boot ROM or similar that is executed at first during the booting process and hence initializes the root of trust of the corresponding device (cf. [9]). In doing so, the CRTM initializes a hierarchical measurement chain (e.g., a hash chain) starting with itself followed by a step by step measuring (e.g., hashing) of the program codes of the respective upper layers such as the remainders of the boot ROM, the boot routine, the boot loader, and all consecutive layers that are part of the Trusted Computing Base (TCB). Since the security of the measurement chain and hence the security of the TCB explicitly relies on the security of the CRTM, the CRTM has to be trusted a priori by all involved parties.
The TCG Software Stack (TSS) [163] comprises software modules and components that provide the functionality of the TPM to the operating system and to security applications. The TSS presents standardized cryptographic interfaces that require only minimal modifications to be used with existing (security) applications. The TSS handles internal keys management, the command synchronization and controls all critical (e.g., mutually depended) machine-level invocations. As the TSS handles security-critical data and operations, it has to be a trusted OS component that in turn can be protected by applying the TC integrity protection measures (e.g., Binding or Sealing).
Trusted Computing, as implemented by the TCG architecture, enables multiple security mechanisms, among others:

- secured cryptography through on-chip encryption and decryption
- an authenticated booting process which measures the code before it can be executed
- the binding of some data to a platform or their sealing for cryptographic storage together with their secure migration to another platform without decryption
- remote attestation for obtaining integrity and freshness information about a platform configuration
- privacy protection through anonymous attestation
- the establishment of trusted channels.

4.2.2 Cryptographic Hardware

Hardware has typically be used as a means to accelerate processor-intensive cryptographic computations like decrypting/encrypting while freeing the host CPU to perform other tasks. In general, cryptographic accelerator consist in co-processors. The complexity of cryptographic hardware essentially depends on the type of cryptography implemented: whereas symmetric algorithms require only several thousands of gates, the most efficient implementations of asymmetric cryptography require in contrast up to hundreds of thousands of gates. It can be noted

that the performance of symmetric algorithms is often better, except for more recent asymmetric algorithms as typically implemented based on elliptic curves.

4.2.2.1 Smartcards

Smart cards, chip cards, or integrated circuit cards (ICCs) are pocket-sized plastic cards that contain hardware logics, memories or even small microprocessors for on-chip data processing. Like microcontrollers, smartcards are increasingly penetrating many sectors of modern societies, be it in finance, commerce, identification, or medical usage as well as in all fields of computer security. These devices provide an interesting personal and portable yet secure way of carrying around secrets and offer a secure execution environment for plain or cryptographic computations. For instance Smartcards are used for ATM transactions, as SIM cards in mobile phones, decoder cards for Pay-TV, pass cards in public transportation, pay phone cards, electronic wallets and many more. In various countries they replace traditional national ID cards, passports, driving licenses and are used for physical access control to restricted areas. Another example is the usage as health insurance card. Smartcards also provide single sign-on to computer systems, hold disk encryption keys and store public/private key pairs and certificates for public key infrastructures (PKI).

4.2.2.2 Tamper Protection

In contrast to most IT attack scenarios, where attackers usually are only able to access the upper (security) application interface (e.g., online banking server, WLAN attacks), in most embedded IT attack scenarios and especially in many vehicular IT attack scenarios, attackers often also have full physical access to breach the security of a IT system. The physical attacker is able to modify also underlying hardware mechanisms that all upper layer (security) mechanisms inherently have to trust for their correctness. The physical attacker further has almost virtually unlimited time and unlimited trials and hence can undisturbed mount almost any feasible attack without having to fear to be detected, backtracked, or locked out. Physical attacks include amongst many others physically accessing (e.g., readouts, deleting, changing) chip memories with secret data such as cryptographic keys, physically manipulating (e.g., modification, deactivation) critical IC functionalities such as filters and sensors, forcing internal signals directly (e.g., via wire access) or indirectly (e.g., via special environmental parameters, fault injections), and sophisticated probing attacks such as SPA (simple power analysis), DPA (differential power analysis), or EMA (electromagnetic analysis).
Even though it is practically impossible to fend off a sufficiently motivated (and sufficiently funded) physical attacker completely once he has physical access to the attack target. However, if it is not possible to avoid any physical access, which is usually the case for most embedded and automotive security-relevant applications, where the attacker is often also the legitimate user of the corresponding IT system, appropriate protection is feasible only by the means of economic security. This means that the cost of compromising a hardware component should generally outweigh the corresponding potential rewards. This in turn can be reached by keeping the potential outcome of a successful compromise as small as possible (e.g., no global keys or passwords)

while keeping the costs for a successful compromise as high as possible or better, at least as high as the maximum potential outcome. While the first can be achieved by appropriate system design, the latter can be applying physical security measures also called tamper protection.
Tamper protection measures usually either aim to prevent any kind of disclosure and modification (tamper-resistance), or aim to at least enable a subsequent detection of potential disclosures or modifications by a regular and unpredictable examining control entity (tamperevidence). Physical security measures can be further distinguished into active (tamperresponsive) and passive (tamper-evident, tamper-resistant) protection measures. In many cases, physical security and hence tamper-protection can be increased by applying tamperevident, tamper-resistant, and tamper-responsive measures in parallel, or by applying them in a layered manner, where especially security-critical sub-components inside of a protected hardware component have additional individual tamper-protection mechanisms with yet a higher protection level. The final protection level, however, can be estimated by formal evaluations such as provided by the Common Criteria methodology [80].

4.2.2.3 Physically Unclonable Functions

A Physically Unclonable Function (PUF)[58] is a physical one-way function (or physical random function), as first introduced in [128]. Basically, random variations in the manufacturing process have been exploited in order to uniquely identify a device. Such variations make it nearly impossible to manufacture a clone. In other words, a PUF can be used to store a device identifier or secret key without the need for a non-volatile memory [19]. PUFs may be used for authentication purposes: for any challenge presented to a device, the device must answer with a certain unique response that matches an expected value initially recorded. PUFs may also be used for secret key storage. PUFs have the property of being tamper-resistant. Any probes or physical damage applied to a PUF will change its characteristics because a PUF embedded in the device in an inseparable way.

4.3 Network Based Security

Apart from the execution environment, the integrity of some computation also depends on the inputs to applications. In a distributed system, which is typically the case for the SOA applications we consider, messages have to be exchanged between separate execution environments and are thus susceptible of being attacked. It thus becomes necessary to ensure the integrity of those messages using cryptographic techniques but also structural properties of the messages. This is typically enforced through the use of filtering with firewalls.
Firewalls are used mostly in order to achieve access control based on user authentication and auditing or logging. They often constitute an essential element of a network security architecture. Broadly speaking, firewalls can be categorized into network layer firewalls and application layer firewalls. As defined in [136] *A network firewall is a system or group of systems used to control access between two networks a trusted network and an untrusted network using preconfigured rules or filters*. In a network layer firewall, a simple router can be considered as a network layer

firewall which, in the example of IP networks, filters packets based on their source address, destination address, and ports. In application layer firewalls instead, a host proxy server examines all the traffic passing through it and prevents direct traffic access among different networks. As explained by Chris Partsenidis *An application layer firewall can be used as a network address translator, since traffic goes in one side and out the other, after having passed through an application that effectively masks the origin of the initiating connection*. Not all of OSI layers require firewall capabilities. Modern firewalls operate on the Application, Transport, Network, and Data link layers to increase the configuration granularity present in firewalls. In other words, adding additional layers allows the firewall to accommodate advanced application and protocols [85]. Firewalls can act as Virtual Private Network (VPN) gateways. They can also be used for content filtering i.e. firewall-based content filtering. A firewall acting as a VPN gateway encrypts the content and forwards it to other networks such as remote networks, whereas in firewall-based content filtering, the firewall filters the content of the incoming information, e.g., the web traffic entering the network. Firewalls working at a higher layer (application, transport layer) such as application-proxy gateway firewalls usually impose a user authentication and logging events to specific users.

Chapter 5

SOA Security Models and Infrastructures

5.1 Primitive service stacks

The security model associated to the primitive stack is quite large with regards to existing standards (cf fig 5.1). In the following section, we are going to focus on the well-know WS-Security standards and will talk about some related standards and models (see section 5.1.2).

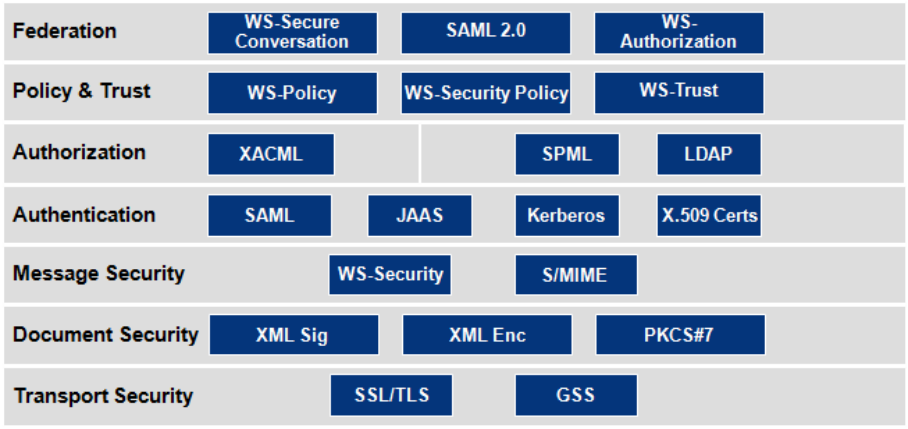

Figure 5.1: Security standards [2]

5.1.1 WS-Security

WS-Security [149] describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. In other word, it's an open format for signing and encrypting message parts (leveraging XML Digital Signature and XML Encryption protocols), for supplying credentials in the form of security tokens, and for securely passing those tokens in a message.WS-Security also provides a general-purpose mechanism for associating security tokens with messages, and for encoding binary security tokens.

The roadmap began back in 2002, but as of the middle of 2004 version 1.0 was ratified by OASIS, with an update (version 1.1) in February 2006. The core standards in this group comprise WS-Security Core (SOAP Message Security) and several token profiles including UserName Token Profile [152], X.509 Token Profile [153], Kerberos Token Profile [150], and SAML Token Profile [151]. The token profiles enable serializing credentials in a consistent manner across platforms, certainly one of the driving forces behind the adoption of WS-Security in the first place.

Given that WS-Security has evolved since 2002, it should be no surprise that most of the interoperability problems that plague early adopters in this space have been worked out. For example, early on even passing a simple UserName token was a challenge between platforms. Now, agreement on the format has aged long enough that the more recent versions of web service platforms are successfully agreeing on the implementation. The same types of challenges have existed with other aspects of the specification, only to eventually resolve for the most common scenarios. Among the most common scenarios for implementing WS-Security today are :

UserName token over transport security This is probably the most popular way to secure services today, because it is considered by many as ”good enough security” for their implementation needs.

UserName token over message security This is used in scenarios where it is desirable to secure messages that might pass through some form of intermediary such as a router. Message security can reduce liability in these scenarios - and in my opinion should be used wherever possible.

Mutual certificate authentication This is commonly used in business partner scenarios where the caller is identified by an X.509 certificate, in lieu of a username and password. In some cases an X.509 certificate is used to prove the originating partner, while a UserName token is also provided to identify the specific user making calls to the service, for authorization and audit trail purposes.

Using previsous security wasn't enough, as he industry had a set of existing and widely accepted transport-layer security mechanisms, such as SSL (Secure Sockets Layer), TLS (Transport Layer Security). Despite their popularity, SSL and TLS (which is a minor update of SSL) have some limitations when it comes to web services.

- SSL is designed to provide point-to-point security, which is not enough for web services because end-to-end security is required. In a typical web services environment where XML-based business documents are routed through multiple intermediaries, it becomes difficult for those intermediaries to participate in security operations in an integrated fashion.

- SSL-based communication provides security (confidentiality, integrity) at the transport level rather than at the message level. As a result, messages are protected only while in transit on the wire. For example, sensitive data received on SSL channel, once persisted, is not generally protected unless one applies some encryption technology.

- Finally, SSL does not provide element-wise signing and encryption. For example, if there is a large purchase order XML document, and there is a need to sign or encrypt only a credit card element, signing or encrypting only that element with SSL is not possible.

All of these limitations make SSL and TLS unsuitable for most of the security needs of web services. Considerable effort has been invested by industry to provide a new security architecture to address the needs of SOA.

All of these limitations make SSL and TLS unsuitable for most of the security needs of web services. In the last couple of years, this gap has been identified by the industry and considerable effort has been invested to provide a new security architecture to address the needs of SOA. It is important to realize that the new service oriented security architecture must adhere to the essential characteristics of SOA.

5.1.1.1 Message security

Message security is about providing message confidentiality, integrity, non-repudiation, and exchanging security credentials between web service client and web service.

WS-Security is an OASIS standard. WS-Security describes enhancements to SOAP messaging to provide message integrity, message confidentiality, and message authentication. WS-Security uses XML Signature [166] to provide message integrity and message authentication and uses XML Encryption [165] to provide confidentiality. WS-Security also provides a general-purpose mechanism for associating security tokens with messages. Examples of security tokens are X.509 certificate, SAML assertion.

XML Encryption is a W3C recommendation. It ensures confidentiality of XML information transfers. XML Encryption allows the parts of an XML document to be encrypted while leaving other parts open. WS-Security provides processing rules for using XML Signature for SOAP messages. XML Signature is a W3C recommendation. It ensures message integrity and authentication. WS-Security provides processing rules for using XML Signature for SOAP messages. Signature can be applied over parts of an XML document.

As mentioned above, in B2B applications, sometimes there is a need for message non-repudiation. Non-repudiation is required due to malicious senders who can later disavow having created and sent a particular message. Resolving non-repudiation issue requires message authentication and sender authentication simultaneously. This can be achieved by using XML Signature for message authentication and SSL-based sender authentication.

5.1.1.2 Trust

In a distributed environment like SOA, the two sides (client and service) need to establish trust before they can interact with each other.

WS-Trust [156] is an OASIS standard. WS-Trust defines extensions to WS-Security to provide mechanisms to get security tokens and to establish trust relationships. For example, a client can send only X.509 security tokens and the web service can accept only SAML security tokens. WS-Trust provides a protocol to get the SAML security token by presenting the X.509 security token. By doing so, WS-Trust resolves the token format mismatch; trust between client and web

service can be established. This provides a great benefit as different security infrastructures can interoperate with each other without significant changes.

5.1.1.3 Distributed Policies

In a typical SOA, where the client and the service may not be in the same security domain, policies enforce security rules on the outgoing (client side) and incoming (service) messages.

WS-SecurityPolicy [155] is an OASIS standard. It describes how senders and receivers can specify their security requirements and capabilities. For example, a service can specify that it requires SAML token and signed message in the incoming SOAP request. WS-Security Policy is based on WS-Policy (a W3C Recommendation). WS-Policy is fully extensible and does not place limits on the types of requirements and capabilities that may be described. It also defines a mechanism for attaching or associating service policies with SOAP messages.

5.1.1.4 Interoperability

In general, wide spread of adoption of security standards increases interoperability, but even different implementations of the same specification may not interoperate in some cases. Core specifications (WS-Security, SAML, etc.) are designed in a way to be extensible and provide number of options for doing the same thing.

WS-I BasicSecurity Profile [174] is a specification from WS-I [173] to promote interoperability in the context of Web Service security. WS-I isn't a standards development organization, but it works closely with a number of standards bodies - W3C, OASIS - to promote and utilize the right set of technologies in business scenarios. The Basic Security Profile provides guidance for the use of web services security standards and technologies in the development of interoperable web services. The Basic Security Profile is an interoperability profile that addresses transport security, SOAP messaging security, and other security considerations. The profile provides specific security requirements, which can be tested on sender as well as receiver side.

5.1.2 Intelligrid model

An interesting model of security comes from the Intelligrid project [79] : "The Integrated Energy and Communications Systems Architecture (IntelliGrid Architecture) project represents the initial steps on a journey toward a more capable, secure, and manageable energy provisioning and delivery system".

This project address security concerns for public electric systems, which are small resources constrained devices with limited computing resource spread over towns and countries. Security is an issue that several industries and most businesses are attempting to come to terms with. However, the implementation of a robust security infrastructure often appears to be a daunting and overwhelming task. According to this project, this can be attributed to several factors:

- There is no defined mechanism to decompose the security problem space and therefore it is perceived to be an impossible task.

- There may be a lack of understanding in regards to the importance of a security policy and a commitment to implement that policy.
- There has been no authoritative work in regards to defining abstract security services.
- There is typically a lack of understanding in regards to the impact of security on communication requirements. This is due in large part to the lack of communication/infrastructure requirement definition.

Thus this project proposes to deal with security by following processes shown in the figure 5.2.

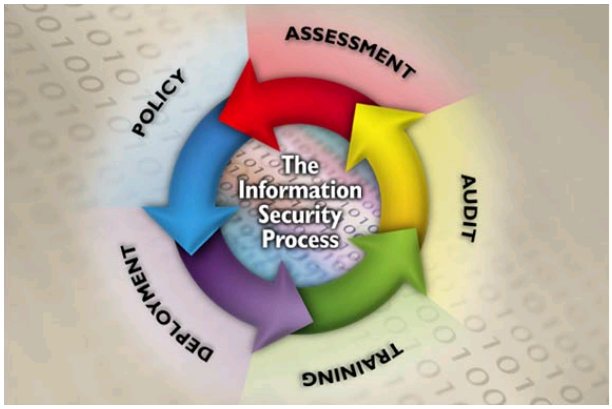

Figure 5.2: Intelligrid security processes [78]

Security Assessment - Security assessment is the process of assessing assets for their security requirements, based on probable risks of attack, liability related to successful attacks, and costs for ameliorating the risks and liabilities. The recommendations stemming from the security requirements analysis leads to the creation of security policies, the procurement of security-related products and services, and the implementation of security procedures.

The implication of the circular process is that a security re-assessment is required periodically. The re-evaluation period needs to be prescribed for periodic review via policy. However, the policy needs to continuously evaluate the technological and political changes that may require immediate re-assessment.

Security Policy - Security policy generation is the process of creating policies on managing, implementing, and deploying security within a Security Domain. The recommendations produced by security assessment are reviewed, and policies are developed to ensure that the security recommendations are implemented and maintained over time.

Security Deployment - Security deployment is a combination of purchasing and installing security products and services as well as the implementation of the security policies and

procedures developed during the security policy process. As part of the deployment aspect of the Security Policies, management procedures need to be implemented that allow intrusion detection and audit capabilities, to name a few.

Security Audit (Monitoring) - Security audit is the process responsible for the detection of security attacks, detection of security breaches, and the performance assessment of the installed security infrastructure.

However, the concept of an audit is typically applied to post-event/incursion. The Security Domain model, as with active security infrastructures, requires constant monitoring. Thus the audit process needs to be enhanced.

Security Training - Continuous training on security threats, security technologies, corporate and legal policies that impact security, Security measures analysis is a periodic, and best practices is needed. It is this training in the security process that will allow the security infrastructure to evolve.

This project doesn't stop to these steps, but dig into security domains, and security functions (access control, trust, authentication, confidentiality, etc.). It's a complete analysis of security with exhaustive analysis from design to technical recommendations.

5.1.3 Restful WS Security

We have already introduced RESTful services in deliverable 1.1, but didn't spend time on security for these services. The fact is nowadays, security for REST services is still an open question. It's mostly an aggregation of best practices rather than the use of standards like WS-Security. The main difference with standards web services (the WS-*) is the abstraction level isn't the same. Where WS-* services uses a reliable application layer (such as HTTP) as transport layer, REST services use the application layer as a mean. That means there is no additional features in top of the application protocol, and one should provide other mechanisms to ensure security.

As stated by Don Park [129] back in 2002, "If you do come up with some standard structure to do the same in REST, you are basically reinventing SOAP". So in the following sections, we'll present the current state of the art and best practices dedicated to RESTful services.

5.1.3.1 Amazon S3 model

For Amazon S3, Authentication [143] is the process of verifying the identity of a user or service trying to access an Amazon Web Services (AWS) product. Access Control defines who can access objects and buckets within Amazon S3 and the type of access (e.g., READ, WRITE, and so on). Authentication combined with access control prevents unauthorized users from accessing your data, modifying your data, deleting your data, or using your AWS account for services that cost you money.

Every interaction with Amazon S3 is authenticated or anonymous. When you sign up for an AWS account, you are provided with an *AWS Access Key ID* and a *Secret Access Key* . When

you perform a request with Amazon S3, you assemble the request, perform a hash on the request using your Secret Access Key, attach the Signature (hash) to the request, and forward it to Amazon S3. Amazon S3 verifies the Signature is a valid hash of the request and, if authenticated, processes the request. All requests the AWS must include a signature value. The signature value authenticates the request sender. This is important because some actions you take in Amazon S3 incur charges. Signing your request insures that you will only be charged for the actions you take. The signature value is, in part, generated from identifiers AWS assigns you when you create an AWS account.

More precisely, when the system receives an authenticated request, it fetches the AWS Secret Access Key that you claim to have, and uses it in the same way to compute a "signature" for the message it received. It then compares the signature it calculated against the signature presented by the requester. If the two signatures match, then the system concludes that the requester must have access to the AWS Secret Access Key, and therefore acts with the authority of the principal to whom the key was issued. If the two signatures do not match, the request is dropped and the system responds with an error message. The Amazon S3 REST API uses the standard HTTP *Authorization* [50], section 14.8, header to pass authentication information. (The name of the standard header is unfortunate because it carries authentication information, not authorization). Under the Amazon S3 authentication scheme, the Authorization header has the following form : "Authorization: AWS AWSAccessKeyId:Signature".

A valid time stamp (using either the HTTP Date header or an x-amz-date alternative) is mandatory for authenticated requests. Furthermore, the client time-stamp included with an authenticated request must be within 15 minutes of the Amazon S3 system time when the request is received. If not, the request will fail with the RequestTimeTooSkewed error status code. The intention of these restrictions is to limit the possibility that intercepted requests could be replayed by an adversary. For stronger protection against eavesdropping, use the HTTPS transport for authenticated requests.

To allow selected users to access objects or buckets in your Amazon S3 account, you can use access control lists (ACLs) or query string authentication. The access control policy in Amazon S3 works as follow. Because of restrictions in what can be sent via http headers, Amazon S3 supports the concept of canned access policies for REST, but also send the definition of ACL policy in one plain XML file. The canned access policy allows to define rules like (i) private (ii) public-read (iii) public-read-write (iv) authenticated-read (v) bucket-owner-read (vi) bucket-owner-full-control to allow an user to interact with a set of objects or buckets.

One last security property covered by Amazon S3 model is the Server Access Logging. An access log record contains details about the request such as the request type, the resource with which the request worked, and the time and date that the request was processed. Server access logs are useful for many applications, because they give bucket owners insight into the nature of requests made by clients not under their control.

5.1.3.2 OAuth model

OAuth is ”an open protocol to allow secure API authorization in a simple and standard method from desktop and web applications” [75]. It provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end- user). It also provides a process for end-users to authorize third- party access to their server resources without sharing their credentials (typically, a username and password pair), using user- agent redirections.

The security model of the first version is widely describe in [68] and represents one current work to provide security mechanisms above reliable protocol such as HTTP. Currently there is a draft for a second version planned in end of 2010 [76].

5.1.3.3 FOAF+SSL

FOAF+SSL is a secure authentication protocol that enables the building of distributed, open and secure social networks: the Social Web which combines efficiently FOAF and SSL.

The basic idea behind FOAF[111] is simple: the Web is all about making connections between things. FOAF provides some basic machinery to help us tell the Web about the connections between the things that matter to us.

Thousands of people already do this on the Web by describing themselves and their lives on their home page. Using FOAF, you can help machines understand your home page, and through doing so, learn about the relationships that connect people, places and things described on the Web. FOAF uses W3C's RDF technology to integrate information from your home page with that of your friends, and the friends of your friends, and their friends...

FOAF+SSL is then the decentralized secure authentication protocol utilizing the FOAF profile information as well as the SSL security layer. FOAF+SSL is using the web of trust concept, but trying to avoid dealing with all key signing parties [157]. This is another example of how we can use RESTful authentication [71].

RESTful security has been covered by [53, 132, 106, 112, 36], and best principle of security is to use SSL over HTTP for message and transport security. A good approach is to introduce mechanisms like the Amazon S3 model for authentication and authorizations if you can afford dealing with key signing parties. Otherwise, good alternatives have been published with FOAF+SSL.

The problem as stated in [130] is that most mechanisms have to be designed according to the specific needs : ”This freedom from choice leads to substantial design and development efforts for decisions with a single do-it-yourself alternative”. One of our goal in CESSA is then to automatize security of RESTful services, to be able to apply the correct pattern whenever needed (as well as other kind of services).

5.2 Identity Management services

Identity Management services is a mean to manage user accounts including role or authorization assignments. Also know under IDM, we'll present in the following sections what it means exactly, and one current application using emerging standards.

5.2.1 Federated Identity Management

Identity federation provides the means to share identity information between partners. To share information about a user, partners must be able to identify the user, even though they may use different identifiers for the same user.

As SOA environments are typically highly decentralized in nature, identity management becomes a significant challenge for web services. Identities can be stored in many directories as well as many different types of directories, including proprietary username/password repositories, LDAP, Active Directory, and X.509 certificate stores. An additional challenge is that SOA may have requests that result in additional requests to many different applications at once. An SOA-ready service may be composed of many service operations from many different services that each has their own identity. As part of a single transaction, many different services may be touched whether in parallel or in serial. Being able to authenticate and be authorized across all of these systems seamlessly improves the user experience as well as performance - driving the need for a federated identity solution for SOA environments.

SAML [35] (Security Assertion Markup Language) is an OASIS Standard. SAML provides a framework for the exchange of security credentials amongst disparate systems and applications in an XML-based format. WS-Security provides a SAML Profile that defines mechanisms to use SAML 1.1 within the context of SOAP messages. The SAML 2.0, which is now an OASIS Standard, incorporated much of Liberty Alliance's work in an effort to become a unified standard for identity federation, which adds account linking, Single Sign-on/off, and improved facilities for establishing trust between organizations.

WS-Federation [154] is an OASIS initiative. WS-Federation defines mechanisms to allow different security domains to federate by brokering trust of identities, attributes, or authentication between participating web services. It is based on WS-Security, WS-SecurityPolicy, and WS-Trust.

5.2.2 Single Sign-On

Single Sign-On (SSO) protocols enable companies to establish a federated environment in which clients sign in the environment once and yet are able to access to services offered by different companies. The Security Assertion Markup Language (SAML) 2.0 Web Browser SSO Profile (SAML SSO, for short) [35] is the emerging standard in this context: it defines an XML-based format for encoding security assertions as well as a number of protocols and bindings that prescribe how assertions should be exchanged in a variety of applications and/or deployment scenarios. SAML SSO is at the core of several SSO solutions like the Liberty Alliance project [121]

and the Shibboleth Project [81]. In addition, established software companies base their SSO implementations on SAML SSO. This is the case of Google that developed a SAML-based Single Sign-On for its Google Apps Premier Edition [60], a service for using custom domain names with several Google web applications (e.g. Gmail, Google Calendar, Talk, Docs and Sites). The security of a SAML SSO solution critically depends on several assumptions (e.g. the trust relationships among the involved parties) and security mechanisms (e.g. the secure transport protocols used to exchange messages). The many security recommendations that are available throughout the bulky SAML specifications are useful in avoiding the most common security pitfalls but are of little help in ensuring their absence. It is therefore very difficult to achieve the needed level of assurance even for very simple instances of the protocol

5.3 Security of Service Discovery Mechanisms

The deployment of ubiquitous computing systems and the trend towards Service Oriented Architectures will undoubtedly generalize the need for discovery mechanisms as essential components for locating ambient and location-based services. Service discovery in a network can be implemented in two manners, first using a decentralized architecture relying on point to point (broadcast) or point to multipoint (multicast) communication, and second using a centralized architecture based on an identified registry relied upon by users and servers to facilitate discovery request matching.

Discovery is very often performed at the initiative of the service requester (e.g. lookup model) but can also be initiated by the service provider (e.g. advert model). The specificity of discovery is that these players, who may pertain to different administrative domains, are by definition initially unaware of their respective existence and security policies. Discovery has to deal with some threats at different levels (Service side, Client side, and Registry side), to name a few : non-availability of a service at service side, request replays at client side, fake registrations at registry side, etc.

To address these threats, requirements for service discovery has been defined as :

Authentication Without the means to authenticate clients and servers, service discovery makes the implementation of a man-in-the-middle attack possible [49], a malicious entity being able to wrongly answer a discovery message.

Authorization Only authorized clients are able to discover restricted services. This authorization must be provided by clients in order to prove their right to discover the service.

Privacy The discovery initiator takes a more important risk than the other party since he does not control the entities which will receive the discovery message, nor the potential usage of the information embedded in his request message. The information disclosed by client requests is likely to reveal a subset of the intentions of the service requester.

Confidentiality In service discovery, confidentiality is used to protect sensitive data (not merely private) contained in the discovery messages; it restricts access to these data to allowed users.

Access control Since client/service authentication is problematic in the initial discovery phase, traditional service oriented architectures do not support access control during discovery. Service providers would ideally advertise their services exclusively to authorized users.

Integrity The integrity of a data or a document concerns the detection (ultimately the correction) of possible errors or modification introduced on the data like for example discovery message modification, alteration, deletion or replay.

Accountability In case of malicious behavior (like fake announcement sent by servers), clients or system administrators must be able keep a trace of this breach for an eventual appeal for a neutral judgment.

Availability Denial of service (DoS) is an attack against the availability of resources preventing the authorized access to a system resource or delaying system operations and functions.

Given these requirements, security solutions have been defined for both centralized and decentralized configuration that we'll present in the following paragraphs. One security solution designed in [162], chapter 3 for a centralized configuration relies on security policies provided by clients and services. Registries have to be considered by services and clients as a trusted third party whose role is no more limited to a basic matchmaker, but which evolves to a security guarantor. In this configuration, clients and services first establish a secure connection (e.g., SSL) with the registry to protect the confidentiality of the exchanged messages. Servers can restrict the discovery of their services to only certified users by specifying a security discovery policy to be enforced by the registry. Clients are also able to restrict the matching scope to some certified services by specifying a security discovery policy also enforced by the trusted registry. Both clients and servers have to provide credentials issued by a known authority that can be used by the registry to authenticate them during the policy verification phase. Another security solution proposed in [162], chapter 2 for a decentralized configuration relies on a particular usage of the Identity Based Encryption mechanism. The server advertises its service capabilities by multicasting its profile to the entire network. Clients can cache service information or ask for a specific service by multicasting its requests to all available servers and only concerned services will respond to him. With no possible reliance on any third party in ad-hoc configurations, clients and servers now must assure their own secure service discovery using a particular encryption scheme. Attribute Based Encryption (ABE) is adopted to make it possible for a server to encrypt its service description according to the restrictions imposed to users (i.e., only a class of users holding corresponding private keys will be able to access to services information). Clients also can use the same encryption mechanism in order to protect their request messages from unauthorized servers (i.e., only a class of servers is able to decrypt the request).

5.4 Security Policies

Access control is a central issue in many systems and applications where multiple parties are represented, either as users or as stakeholders. Access control is entirely about describing resources, which can access them, and controlling that access. Access control policies are defined

as a list of allowed and disallowed states. As defined in [18] *a security policy is a statement of what is and what is not allowed*. Access control policies can be grouped into two main classes as either discretionary, in which case the user is in charge, or mandatory, in which case an authority imposes its decisions. Alternative approaches are role-based access control, rule-based access control, or domain type enforcement. The latter approaches refine the basic two approaches, for instance by making it easier to delegate the decisions of an authority to multiple parties. Various languages have been defined to express security policies, some of them closely tied to access control systems. The most comprehensive one, XACML [43], which is based on XML and is designed to be extensible, is now widely used to implement various access control models. Besides parsing one such authorization policy language, security architectures need to implement authorizations themselves. They generally follow one of two approaches: they either rely on an access control list (ACL), which is *a list of principals that are authorized to have access to some object* [139], or on capabilities, which are *an unforgeable ticket, which when presented can be taken as incontestable proof that the presenter is authorized to have access to the object named in the ticket* [139].

- **Session Assertion Markup Language - SAML :** is one of the protocol candidates. The SAML is an XML standard for assertions regarding identity, attributes and entitlements of a subject [35]. It allows to exchange authentication and authorization data between security domains, that is, between an identity provider (a producer of assertions) and a service provider (a consumer of assertions). The service provider relies on a SAML assertion from the identity provider about the principal to make an access control decision. This setup requires the existence of local authorization services from an identity provider, however, SAML provides a level of abstraction, since it does not specify how they are implemented. Generally, a SAML assertion is a statement made at a given time by an issuer regarding a subject provided that certain conditions hold. SAML assertions can contain three types of statements: authentication, attribute and authorization decision statements. Each of these corresponds to a type of query which forms part of a SAML request-response protocol. The structure of an assertion is described by a SAML profile, which may be defined dependent on the desired application. At the implementation level, SAML messages admit bindings to several standard message types and protocols [103]. SAML provides XML formats for transmitting security information, defines how they work with underlying protocols, and specifies message exchanges for common use cases. In addition, it supports several privacy protection mechanisms (providing means to determine security attributes without revealing identity), and specifies a schema that allows systems to communicate the SAML options they support.

- **XACML :** XACML (eXtensible Access Control Markup Language) is a declarative XML-based access control policy language used to describe the access control restrictions to actions on objects. Usually, access control models involve a subject (that is, either a user, a user on behalf of another user, a service, or a service on behalf of a user) making some access request and the system either authorizes this access request or denies it. XACML also defines a processing model, which describes how to operationally interpret the policies.
  XACML defines both an access control policy language (to express the access control

conditions) and a canonical XML language to communicate with a Policy Decision Point (PDP), to send to the PDP decision requests and obtain decision responses. This canonical form or language is called the XACML context. The current version of XACML, Version 2.0, was released by OASIS in February 2005. Version 3.0 is in preparation at the time of preparation of this document (June 2008). It is chartered to add generic attribute categories for the evaluation context and policy delegation profile (administrative policy profile). There are already some prototypical implementations of XACMLv3.0.

Chapter 6

Security Vulnerabilities in Web Services

Web Services represent the most widely deployed type of SOA and also probably represents one of the most exposed service oriented system with respect to potential attacks. This section highlights some attacks currently in use and the specific countermeasures that are used to address them.

6.1 Known Attacks on Web Services

The research that focuses on the security of single Web Services studied a number of attacks [108, 57] ranging from integrity and confidentiality violations to Denial of Service attacks (DoS) [83]. According to previous research [159, 1, 40], the attacks against web services can be classified in the following categories:

- *XWS1, Web Service interface probing*: The revealed access information of targeted web sites results the attackers leverage this information for their malicious purposes. This attack can also be called WSDL Scanning attack.
- *XWS2, Brute force XML parsing system attacks*: The heavy XML parsing can allow an attacker to perform DoS attacks that may be realized in different ways such as sending XML bombs or recursive payloads.
- *XWS3, Malicious Content attacks*: Malicious content in an XML file can exploit known vulnerabilities with various techniques such as buffer overflow.
- *XWS4, External Reference attacks*: Poor configuration and improper use of external resources may allow an attacker to set different DoS scenarios or to perform inform information theft.
- *XWS5, SOAP/XML Protocol attacks*: The SOAP messaging infrastructure can be employed for devising more sophisticated attacks such as replay or man-in-the middle.

- *XWS6, XML Security Credentials tampering*: XML credentials and assertions that are crucial elements for service authentication can be manipulated for application session hijacking.
- *XWS7, Secure key/session negotiation tampering*: In the case the session and authentications keys are not generated strong enough, the content might be manipulated by an attacker.

In addition to this classification, Web Services security is actively investigated by organizations like the OWASP Foundation, which in particular periodically publishes a guide for testing deployed Web Services that encompasses a rather comprehensive list of known vulnerabilities [55].

The rest of this section highlights additional attacks on Web Services that provide more insight on the security requirements and related security mechanisms.

6.1.1 Service Discovery Vulnerabilities

This section provides a list of threats and the possible attacks [162], chapter 1 that can be built against the data and resources of service discovery players. Although this list is non-exhaustive most of the discovery protocols (centralized and decentralized) are studied and analyzed in order to find out weaknesses and vulnerabilities that could be exploited by attackers. For each threat, a possible countermeasure is proposed that can be applied in order to prevent the disruption of the service discovery service. The following description lists threats to the centralized and decentralized service discovery architectures together.

6.1.1.1 Protocol Messages and Entities

- The registry is not available (service-side): the attacker performs a Denial of Service attack by flooding registration messages. He intends to force the registry to consume its resources in such away as it can no longer provide its intended service. One of the possible countermeasures is to modify the protocol by adding anti-clogging messages, if message parsing is too costly, then blacklist originators of bogus messages.
- Client request disclosure (client-side): client intentions, activity, or identity may be revealed, directly or indirectly by his service lookup queries. An appropriate countermeasure consists in setting up secure channels (encryption). For instance, during an industrial social event, companies could provide Job services for people that want to apply for a job. Some companies by intercepting job lookup messages sent by users applying for a job are able to know people that may leave their company for another competitor company
- Interception of request (client-side): the discovery request reveals private information about service discovery clients. A possible attack consists in faking the identity of a registry that is known and trusted and forwarding to that registry. Registry certificate distribution might be an adapted countermeasure to prevent this type of attack. The process of

distributing certificates of trusted registries should be protected during the configuration phase of the mobile device. A fake Bank Server could play a masquerade attack in order to obtain private banking information from users.

- Message modification or drop (client side): if the attacker compromises a router from the network, he can intercept and modify or drop the client's lookup message to the registry. The client should protect the message he sends with respect to its integrity and to the authentication of its origin, for instance with a signature or a message authentication code. A redundancy mechanism can be configured to guarantee the delivery of the messages in case of dropping.

- Replay of Request message DoS (client-side): the attack consists in replaying a lookup message coming from a legitimate client. A sequence number could be added to the message in order to drop the previously processed messages.

- Replay of registration message (registry-side): the attacker replays the registration message of a properly authenticated service in order to update the service profile with wrong information. A signed sequence number must be added to the registration message in order to take into account the processed messages and drop the relayed ones. An attacker could reuse a real banking service publish message in order to setup a fishing attack.

6.1.1.2 Service Registration (centralized architecture only)

- Registration to a malicious registry (server-side): an attacker might fake being a registry whose identity (and implicitly matching behavior) is known and trusted. Subsequent attacks include preventing clients from matching the registered service. Registry authentication is one possible countermeasure. This can be achieved by ensuring a properly protected distribution of the certificates of trusted registries during the configuration of mobile device, which also requires an initial authentication phase during discovery; or by ensuring that registry keys are distributed to mobile devices and that communication with the registry is encrypted with that key.

- A service can be deregistered by an unauthorized party (registry-side): this occurs when an attacker tries to dereference an active service from the registry which it registered previously. The use of a nonce (e.g., sequence number) with a signature (MAC) by the registered service to certify the origin of a de-registration message constitutes a possible countermeasure to such attacks.

- Fake registration (registry-side): An attacker can send a fake registration message to the registry containing wrong information with fake attributes. To prevent this attack, the registry has to include a verification of the proper certification of attributes of registering services by appropriate authorities together with a proof of identity of the registering party (e.g., signature of registration request).

6.1.1.3 Matching process

- Client lookup disclosure (client-side): client intentions or activity might be disclosed if the matching process is open to all services registered. A service may have been established to gather statistics about users trying to access a certain profile of services. More dangerously, an attacker might try to get access to confidential information sent by the client at the access phase subsequent to service discovery. The countermeasure to this attack consists in restricting the services whose description matches the client lookup with additional constraints on some of their certified attributes. This specification can take the form of a policy submitted by the client together with his lookup request, and which may refer to the same or to attributes of the services different from those specified in the lookup request.

- Service discovered by unauthorized party (service-side): a typical example of this threat is the possibility for an attacker to determine the identity or content served by a service which wants to be seen or accessible only by a restricted set of other services (service trapping). The countermeasure consists in the delegation of a trusted (and authenticated) registry, the enforcement of a restrictive policy provided by the service that will allow the service discovery by authorized clients only. We also recommend the use of restrictive cryptographic mechanisms. This kind of threat can be extremely dangerous in case of industrial spying. A spy might discover all the interfaces and the architecture topology of an automated factory relying on web services for the workflow management of industrial machines.

6.1.2 Payload Attacks

Web Services are potentially exposed to attacks through the payloads of the messages they exchange. This is in particular true for SOAP based Web Services, whose messages are rather complex due to their structure. In fact, since the SOAP protocol is based on XML and it is usually transported over the HTTP protocol, the security of both of them is essential to protect web-services against possible misuse.

[134] describes for instance an XML rewriting attack which may result in the successful injection of a bogus payload into a SOAP message due to a problem in the semantics of the WS-* stack. In general, the term *XML rewriting* is used to describe any attacks based on "the malicious interception, manipulation, and transmission of SOAP messages" [133]. In this category, a certain class of attacks targets the XML parsers as a way to cause a denial of service. For example, Coercive Parsing [100] is based on deeply nested XML documents, while Oversize Payload [178] relies on sending very large requests containing an extremely high number of elements, aiming at exhausting the server's resources.

RESTful web services also exhibit specific vulnerabilities through parameter passing with HTTP. These Web Services have recently been shown to be vulnerable to HTTP Parameter Pollution (HPP) attacks [24]. Those attacks, which have only been recently highlighted, aim at exploiting the absence of a specification for parameter passing on HTTP and at subverting the parsing of the URI referencing the access to a particular resource mediated by a Web Service,

which may result in successful attacks at the application layer.

All those attacks may also be used for implementing specific denial of service attacks targeting the service infrastructure protocols, in addition to traditional TCP/IP-based denial of service attacks. Denial of Service attacks are an attempt to make a certain resource unavailable or unresponsive to its legitimate users.

6.2 Countermeasures for deployed Web Services

Generally the defense against the attacks we listed above are typically at lower network layers [22]. However, if attack payloads can be located inside the XML headers that are associated with SOA, they can effectively be used to extract signatures to be deployed to traditional intrusion detection systems. Of course, such mechanisms require a detailed analysis of new attacks that emerge everyday. Apart from attack detection, the most common approaches deployed today are access control and the systematic verification of software.

6.2.1 Access Control

Specific Access Control Models. Shalka et al. [146], propose a more comprehensive defense technique called "trust-but-verify". The approach consists in dividing the authorization phase into two different phases: namely online and offline phases. Compared to earlier access control systems, a finer grained approach [179] introduces an attribute based access control for web services. Finally, Tari et. al [160] go beyond access control and builds a security model based on information flow control, particularly focusing on the confidentiality and integrity of the data.

SOAP/XML firewalls. A common solution against protocol-based attacks consists in relying on firewall or network filters. The term *firewall* is commonly associated to packet filter devices that operate up to the layer 4 of the ISO/OSI model. The task of analyzing and filtering higher layer protocols (such as HTTP) is instead entrusted to the so-called Application-Level Gateways (ALG). ALGs can, for example, protect against malformed requests or they can be used to limit the access to certain resources.

The same concepts have also been extended to the SOAP protocol. Web-Server firewalling has been studied by academia [63, 21], industry [137], and standard bodies [145]. In fact, most of the vendor in the network security area already propose solutions tailored for the protection of web-services environment (e.g., Citrix Netscaler [118] or CISCO XML Gateway [33]). These firewalls are essentially application gateways that inspect the payload. This approach is less appropriate when payloads are encrypted as the encrypted content necessarily escapes the firewall analysis. However this approach is rather fit for sanitizing cleartext payloads and may for instance be adapted to address XML rewriting attacks.

6.2.2 Systematic verification

Fuzzing [110] is a black-box approach often used to test the security of software components. The technique consists in providing invalid (or anomalous) input to an application to discover software bugs and vulnerabilities. Even though it has been very effective to find security problems in many applications, fuzzing, unlike other approaches, cannot provide any guarantee on the ability to find flaws.

A number of tools and techniques focus on fuzzing various network protocols (e.g., Spike [148] and Protos [92]). Recently, fuzzing techniques have also been applied to the testing of web services and service oriented architectures. Huang et al. [74] presented a fuzzer for web forms, while Looker et al. [105] focused on the evaluation of SOAP components. Finally, WSDigger [175] is an open source tool that can automatically generate web-service requests starting from a WSDL description.

Chapter 7

Secure Design

7.1 Best Practices

7.1.1 Statement on Auditing Standards No. 70: Service Organizations

Commonly named SAS 70 [122], it's an auditing statement developed by American Institute of Certified Public Accountant (AICPA). It defines professional standards, used by service auditor to assess the internal controls of a service organisation. SAS 70 is the authoritative guidance that allows service organizations to disclose their control activities and processes to their clients and clients auditors. It signifies that a service organisation has had its control objectives and control activities examined by an independent accounting and auditing firm.

The main objective is to issue an independent opinion on a service organization's report about an organization's description of controls through a service auditors report but also to present that a service organization has been through an in-depth audit of their control activities, which includes control over information technology and related processes.

Using such audit statement helps to (i) provide outside parties with independent third party verification regarding the state of their internal controls, (ii) build up trust with user organisation, (iii) identify opportunities for improvements in many operation areas, (iv) distinguishes service organisation over its competitors and (v) receive detailed description of the service organization's controls and an independent assessment of whether the controls were placed in operation, suitably designed, and operating effectively.

SAS 70 reports are service auditor reports are primarily used as auditor-to-auditor communication. Auditors of the service organization's customers can use the service auditor's report to gain an understanding of the internal controls in operation at the service organization. Use of the report is typically restricted to the service organization's management, its customers, and the financial statement auditors of its customers. Unless stated that other parties must use the report.

The audit guide is designed to provide guidance to auditors of companies that use service organizations as well as service auditors who perform SAS 70 examinations.

7.1.2 Common Criteria - CC

Initially, various countries had different IT security assurance standards. Information Technology Security Evaluation Criteria (ITSEC) of European Union, Trusted Computer Security Evaluation Criteria (TCSEC) of U.S. government, Canadian Trusted Computer Product Evaluation Criteria (CTCPEC) of Canada and US Federal Criteria (DC), all these criteria′s were serving for the same propose in different countries [17]. Then these nations agreed to set a common set of criteria for the corporate users of assured IT products and as a result , Common Criteria - CC came into existence. CC standardized documentation is CC V3.1 [48] and its its ISO equivalent is ISO/IEC 15408 [52]. CC understanding and successful implementation became evident for IT security products after the publication of NSTISSP No. 11. According to U.S. must be evaluated and assured according to CC or equivalent [119]. CC is now being used in IT products as a guild line for security needs. Firewalls, network servers, operating systems and many other IT products are acquiring CC certification.

This section gives a brief description of CC. The philosophy of CC [48] is to carefully articulate the threats to organizational security policy commitments and products security requirements and propose sufficiently demonstrable security mitigations for their intented purpose. These security requirements can be implemented in software, hardware and in farm ware. Security evaluation process not only considers software for vulnerability assessment but also the hardware, the networking environment and third party services to accomplish the required task. The name given to IT product and its environment is Target of Evaluation (TOE). The documents written to state consumer′s requirements is a standard CC document names as Protection Profile (PP). Developer is the entity responsible for implementing the security requirements of PP. Developer defines security components for implementing the security requirements of PP and provides details of Security Functional Requirements (SFR) in document names as Security Target (ST). CC has throughly identified common IT security requirements and grouped them into categories formally known as Functional Classes. Both consumer and developers take guidance from these classes to specify security requirements and security components respectively. Evaluator is the entity, basically a third party, responsible for assuring the accurate implementation of security components and mapping of SFR in accordance with PP. Degree of formality in evaluation of the TOE is based on CC defined criteria named as Evaluation Assurance levels (EALs). Products are being evaluated at predefined EAL. CC has comprehensively marked the Security Assurance Requirements (SAR) that work as guidelines for both developers and evaluators [69].

Although CC is remarkable in identifying security requirements but it does not give answers to many questions such as, mapping of security requirements in software development life cycle (SDLC).

7.1.3 Six Sigma - 6σ

This section aims to enlighten quality practices introduced by 6σ [161]. Six Sigma seeks to improve the quality of process outputs by identifying and removing the causes of defects (errors)

and minimizing variability in manufacturing and business processes. It uses a set of quality management methods, including statistical methods, and creates a special infrastructure of people within the organization who are experts in these methods.

Sixσ methodology provides basis to monitor, control and analyze organization and project goals. These methodologies help in classifying critical and non critical tasks, transform qualitative measures in quantitative procedures and define quality metrics. Cause and effect diagram are used to identify risks and their impact, failure mode effect analysis (FEMA) to identify detective controls and statistical tools to measure performance level are included in 6σ kit to ensure best quality assurance practices. Basic principal of 6σ is to define quality functions and identify inputs and outputs of functions.

The 6σ is not only a methodology or supportive tool but also a thought, more precisely a philosophy. It tries to find out customer requirements and reduce gaps in implementation of the requirements and minimize variation in processes [69].

7.1.4 OCTAVE

OCTAVE [25] (Operationally Critical Threat, Asset, and Vulnerability Evaluation) is a suite of tools, techniques, and methods for risk-based information security strategic assessment and planning developed by Carnegie Mellon Software Engineering Institute. This approach is driven by two of the aspects: operational risk and security practices. Technology is examined only in relation to security practices, enabling an organization to refine the view of its current security practices. By using the OCTAVE approach, an organization makes information-protection decisions based on risks to the confidentiality, integrity, and availability of critical information-related assets. All aspects of risk (assets, threats, vulnerabilities, and organizational impact) are factored into decision making, enabling an organization to match a practice-based protection strategy to its security risks.

OCTAVE is a self-directed information security risk evaluation. This core concept of OCTAVE is defined as a situation where people from an organization manage and direct an information security risk evaluation for their organization. The organization's people direct risk evaluation activities and are responsible for making decisions about the organization's efforts to improve information security. In OCTAVE, an interdisciplinary team, called the analysis team, leads the evaluation.

The organizational, technological, and analysis aspects of an information security risk evaluation lend it to a three-phased approach. OCTAVE is organized around these basic aspects , enabling organizational personnel to assemble a comprehensive picture of the organization's information security needs. The phases [26] are :

- Phase 1: Build Asset-Based Threat Profiles
- Phase 2: Identify Infrastructure Vulnerabilities

- Phase 3: Develop Security Strategy and Plans

The essential elements, or requirements, of the OCTAVE approach are embodied in a set of criteria, which is a set of (i) Principles, (ii) Attributes, (iii) Outputs. The principles are a fundamental concepts driving the nature of evaluation. It shapes the approach andprovides basis for evaluation. The attributes are distinctive qualities or characteristics of evaluation, defines what makes the evaluation successful. Attributes are derived from principles. Finally, the outputs are the required result of each phase. Unique activities are not set because more than one activity can be done to achieve the outputs of OCTAVE.

In conclusion, we can say that OCTAVE enables an organization to sort through both organizational and technological issues to understand and address its information security risks. It provides a snapshot in time, or a baseline, that can be used to focus mitigation and improvement activities. Thus, OCTAVE can be viewed as a tool that facilitates information security improvement.

7.2 Security Engineering with Patterns

Comparing and contrasting between design and security patterns is a bit difficult, since the first is functional and the second is usually not. For this purpose, the only aim of our comparison is to identify interesting studies for security patterns that are not yet accomplished and that through them security pattern community would grow larger. After all, the only real evaluation of security patterns would be when research papers provide methodologies for teaching security patterns in computer science curriculum as already it is the case for design patterns. In addition to the books [141, 142, 135] there is recently a very informative review [116] that present the different security patterns published in each phase of the Software Development Lifecycle. Starting with (i) requirement phase to (ii) design phase and (iii) then implementation phase. Their main concern behind the shy adoption of security patterns was the unavailability of security patterns covering the full spectrum of software engineering, as they show in their concluding figure of their analysis section. We do not believe this is the unique reason, as we have shown in Section 2.3 about the security patterns format. Since their review is indeed complementary we will first summarize it and then resume with our review methodology.

Security engineering using security patterns is crucial for the scalability of the security patterns approach. In [51] they provided a methodology to build secure systems using security patterns. They propose to specify the required security constraints and then in three stages include them into application sequentially. First they analyze the domain to discover it and its regulatory constraints. Then during the requirements stage, to determine the possible threats and propose appropriate countermeasures, and finally the analysis stage where they applied their analysis patterns to build the conceptual model in a more reliable and efficient way.

The biggest difficulty with security engineering with patterns is that the security mechanism or solution to be implemented vary with the context. For example, a developer who would like to implement an access control in his application may have completely different patterns given

its platform environment, the programming language, or even libraries he is using. Covering the whole possibilities is like an utopia, but efforts have been done to automate part of it. One example is with SAP. Security requirements [3] are an integral part of SAP's development model (Product Innovation Lifecycle - PIL). Compliance to these requirements is mandatory for all major development projects at SAP. The PIL Security Standard focuses on legal compliance, Total Cost of Ownership (TCO) reduction, and the avoidance of potential vulnerabilities. The security requirements are taken in account during the whole development lifecycle (especially in the planning, development, and testing phase). Although the PIL Security Standard is SAP internal, some best-practice excerpts have been published. We can see the SAP PIL as guidelines and link to resources, like [77, 123, 47, 127] for security.

7.3 A State of the Art regarding Approaches for Extracting Security Requirements :

Modeling and validating requirements is a topic at stake in system engineering [101][59][61] more specifically to propose standardized environments / languages. For example, proposes to follow the ISO-17799 standard. Many specification approaches have followed the road of defining profiles [101][61][171][93][70][113] based on the Unified Modeling Language (UML) [125] : in addition to the graphical specification, portability and interoperability are usually among the strength of such profiles. The specification also being semi-formal provides a great deal of flexibility at an early stage of engineering. The next paragraphs discuss the ins and outs of several of those profiles.

A UML profile has been introduced based on the KAOS methodology, one of the most advanced approaches to the specification of security requirements. KAOS provides a language and method to goal-driven requirements design [38] yet was not originally devised with a UML centric approach in mind. KAOS provides semantic elements to represent time, agents, events, goals, goal patterns, goal categories and subgoals as well as conflicting goals and constraints. A relevant feature of KAOS is that goals and related constraints can be defined formally using temporal logic [70]. However, in order to define such expressions accurately, design should be as precise as possible. KAOS properties also span several classes and may properly express non-functional requirements like security ones [70]. However KAOS lacks a systematic coverage of threats and does not provide any support for software-hardware co-design nor code generation and testing in that setting.

The Enterprise Distributed Object Computing profile (EDOC) [124] relies on the Object Constraint Language (OCL) to represent and check requirement satisfaction. However, this profile has been designed to specify functional requirements rather than security ones, and also to improve business processes. It thus provides no appropriate support for security requirements specification. The Refinement Calculus for Object System (rCOS) is an object-based language with a rich variety of features including subtypes, inheritance, type casting, dynamic binding, and polymorphisms [170]. rCOS permits the mathematical characterization of objects through labeled Transition Systems. Despite its benefits, rCOS is inherently software orientated, which is

perfectly fit for service oriented design [171] but not appropriate for the specification of security requirements at system level.
UMLsec [91] introduces a security-oriented methodology based on activity diagrams, state charts, sequence diagrams, class diagrams, and deployment diagrams. As shown in [90] the UMLsec approach can be complemented with automated verification of security properties using an industrial tool. The examples provided with UMLsec outline that the methodology is more about security mechanism design than about security requirements linked with use cases: guidelines might be provided for handling secrecy, secure key management, and security protocol specification. In addition, the specification might be ambiguous due to the UML diagrams used: for instance, defining integrity on top of a link between execution nodes in the deployment diagram may be interpreted either as a property that the link should implement or as a mechanism that the link already implements independently from requirements. Another drawback of UMLsec comes from the fact that it is an extension of UML, and therefore not recognized by most existing tools. Some authors [131] also even claimed that OCL constraints were enough to introduce similar specifications without extending UML.
SecureUML is a profile that aims to provide security again using an extension of the UML specification [104]. SecureUML however only specifies security polices for Role-Based Access Control (RBAC) using graphical notation and logical constraints, so it is essentially adapted to application security. Since these UML profiles are design-oriented, they are generally more suited to describing security mechanisms than security requirements. Other non UML-based environments can also be mentioned, that rely directly on a formal framework like for instance the Symbolic Trace Analyser (STA). STA is a model checker for cryptographic protocols relying on symbolic techniques [20] . Protocols are described in a dialect of the spi calculus. Intruders can be modelled based on the well-known Dolev-Yao model. STA allows to express and verify authentication and secrecy properties. The lack of parametrization of STA leads to large specification when a more instances are needed. Another example is the On-the-fly Model-Checker (OFMC). OFMC implements bounded verification of protocols by exploring its transition system described in a specification in a demand-driven way . OFMC also support the specification of algebraic properties of cryptographic operators. To our knowledge STA and OFMC, as well as other non UML-based security-oriented environments, do not have specific constructs to represent security properties. An exception to the latest statement could be ST-Tool [59]. ST-Tool provides a Graphical User Interface, a Data Modeller and Formal Language and Analysis components that are based on the formal language TROPOS. This profile is mainly intended for Security Requirements Engineering, however the usability and reliability of the tool. A general problem with such approaches comes from the need to define cryptographic protocols between the elements of the system in the first place, before security properties of the system be specified.

Unfortunately, the models defined above work essentially for extracting requirements out of completely defined systems, in particular based on the security mechanisms that they contain and the attacker model. Those approaches therefore do not apply in our case.

Chapter 8

Security Requirements

The increasing complexity of large-scale heterogeneous systems such as distributed business processes has made requirements engineering the most critical phase during system conceptualization. Security requirements in particular should be specified and taken into account before the system architecture is fully defined. However, determining such requirements within heterogeneous systems generally and paradoxically necessitates a detailed enough knowledge of the system components and interactions, like how functions are mapped onto hardware, if some communication might be seen by an attacker, etc. Security requirements in fact constitute the most abstract documentation of the expected system behavior. As such, security requirements should provide a specification that has to be satisfied at every subsequent stage of the system design, validation, development, and testing. Establishing relationships between requirements and such later phases of engineering should thus receive appropriate support: for instance, it should be possible to document the fact that some security mechanism is introduced in order to satisfy one security requirement, or to point at some test over the implementation in order to verify that it is compliant with the same requirement. In the case of embedded systems, the need for hardware protection to satisfy security requirements should be supported by the methodology used. Security requirements should furthermore constitute a manageable documentation for the average system engineer.

8.1 Security Requirements Originating From the Attacker Model

We will consider two classes of attacks:

1. attacks modifying the behavior of the system (active attacks) and
2. attacks aiming at information retrieval without modifying the behavior (passive attacks).

Note: passive attacks are frequently a prerequisite of active attacks; the attacker first analysis the system in a passive way to understand it or recover useful crypto material and then exploits this knowledge to actively attack. There are two possible ways to identify the different types of attacks: the physical one and the functional one.

We will consider active attackers, meaning that we cannot restrict security to protecting from eavesdropping messages but we should consider the active exploitation of design flaws (notably in protocols) and implementation vulnerabilities.

8.2 Security Requirements Originating From the Deployment model (Horizontal compositions)

- No one single organization can decide on security processes that have to take place at another organization. Typically, the certification process may differ from one organization to another and cross-certification might need to be agreed before different organizations can exchange messages securely.
- Standard protocols and security policies should be used whenever appropriate. In particular, the expression of abstract security properties should be linked with specific technologies at lower layers, according to a predefined or negotiated referential between the collaborating organizations. This will typically translate into format translations or reencryption requirements for communications taking place between more than two organizations.
- The security policy will essentially be based on a mandatory model whereby organization-wise policies are defined for all employees, or at least according to the organization layout (hierarchy or matrix organization).

8.3 Security Requirements Originating From the Infrastructure (Vertical compositions)

- Different security mechanisms have to be coordinated together with the execution of the application in order to give access to services and resources of the infrastructure. There typically is a requirement for coordinating the security offered by execution environments through for instance stack inspection techniques with the opening of firewalls at the network level whenever communication between two organizations is required. This should result in a distributed enforcement of security properties, which may require complex synchronizations (both for the enforcement itself and for policy updates).
- Confidentiality properties defined at a high level should be translated at the resource level and will result in defensive security requirements with respect to data encryption, access control policies to data, and data placement.
- non repudiation requires the implementation of an asymmetric encryption scheme in the execution environment supporting the computation
- Freshness properties defined at a high level require the use of a synchronization protocol feasible on a potentially large scale between the relevant parties. This may rely on counters or logical clocks rather than physical clocks.

- Availability properties may impose requirements on the security mechanisms that can be implemented on a particular execution environment depending on the CPU, memory, networking capabilities of the considered environment or the availability of specific cryptographic functions (accelerated or not). Embedded devices are likely to impose some constraints on the cryptographic primitives available.
- Key management should encompass the deployment of keys not only to employees but to customers and onto appropriate devices (typically installation of new keys onto embedded devices).
- Translations between high-level security properties down to infrastructure related security mechanisms should be traceable in order to allow both proactive auditing and a posteriori analysis of an incorrect security policy enforcement. In particular, those traceability mechanisms should make it clear whether the issue stems from an incorrect security policy or a problem in the translation process.

8.4 Security Requirements Originating From the Use Case

- Personal information, as typically identified in a loan application (e.g., details about income, personal situation, bank account), should be kept private, even though other details of the loan application may be publicly disclosed (or at least revealed to other parties like the credit bureau or governmental body) for instance for statistics. The data structures of the message payloads should provide the means to distinguish between those two types of data.
- It should be possible to pseudonymize requests (pseudonyms being generated either by some organization or self-generated). The mechanisms used for this anonymization should be auditable and traceable even as the information flow goes across multiple organizations.
- The origin of the data submitted by the interested parties should be authenticated and non-repudiatable, especially in order to start some official process. Signatures provide the best way to do that. They should be qualified according to legal requirements in order to make it possible to integrate service compositions into legal frameworks. In particular, this may require the use of a specific hardware or specific execution environment preventing untrusted applications from performing a phishing attacks on the signer (typically when signing for the loan).

8.5 Security Requirements Originating From the Service Model

Deliverable D1.1 defines a service model that we will use within the project which consist in various levels of abstraction. This model requires the alignment of some security objectives described in Section 1. We will essentially focus on the following security objectives for the rest of the project:

- **Integrity.** Integrity relates both to communication, storage, and execution integrity. The execution environment integrity is thus an important security objective together with other data integrity measures.
- **Authorization.** The service model specifically highlights the fact that services are defined at a high level by end-users but result in complex vertical compositions. Authorizations should therefore also be defined at an abstract level even though their enforcement takes place at a low level using different mechanisms resulting of the composition.
- **Data origin authenticity.** Authenticity should be considered in a multi-step fashion. The data exchanged may result from the composition of multiple service, which means that usual point-to-point authentication protocols should be adapted.
- **Non-repudiation.** Since operations result from the composition of services from different origins, it might be necessary to define a transactional model for committing parts of a composition based on non-repudiation.

Chapter 9

Conclusion

This deliverable described a list of security properties and identified security requirements that will be used together with the CESSA service and aspect models.

We have provided an extensive analysis of the state-of-the-art of security mechanisms and related analysis techniques, with a special highlight on aspect-oriented software design and development. This analysis encompasses both academic approaches and industrial systems, including existing industrial standards.

We also introduced a comprehensive set of security properties that need to be evolved for SOAs in the presence of horizontal and vertical composition as well as evolution. The security requirements that were derived in this deliverable are also based on an analysis of the service model presented in D1.1, the use case, and the attacker model. This document provides a basis for the analysis and definition of the security properties that will be studied as part of the CESSA project.

These identified security requirements will help select the mechanisms for securing service composition considering both horizontal and vertical composition, i.e., the evolution of large-scale SOAs involving many stakeholders, and the evolution of the local services and their orchestration. Those security requirements constitute a first abstract specification of the expected evolution features that will have to be satisfied at subsequent stages of the project.

Bibliography

[1] A Spire Research Report. Attacking and Defending Web Services. http://www.forumsystems.com/papers/Attacking_andDefending_WS.pdf, 2004.

[2] SAP AG. Security Standards. http://www.sdn.sap.com/irj/sdn/standards-security.

[3] SAP AG. Software Lifecycle Security. http://www.sdn.sap.com/irj/sdn/security?rid=/webcontent/uuid/e0422d71-b23e-2a10-35bd-862787ccae26.

[4] Mehmet Akşit, Siobhán Clarke, Tzilla Elrad, and Robert E. Filman, editors. *Aspect-Oriented Software Development*. Addison-Wesley Professional, September 2004.

[5] D. Alhadidi, N. Belblidia, M. Debbabi, and P. Bhattacharya. lambda saop: A security aop calculus. *The Computer Journal*, page bxn065, 2009.

[6] Dima Alhadidi, Amine Boukhtouta, Nadia Belblidia, Mourad Debbabi, and Prabir Bhattacharya. The dataflow pointcut: a formal and practical framework. In *AOSD '09: Proceedings of the 8th ACM international conference on Aspect-oriented software development*, pages 15–26, New York, NY, USA, 2009. ACM.

[7] Bowen Alpern and Fred Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[8] K. Ashcraft and D. Engler. Using Programmer-Written Compiler Extensions to Catch Security Holes. In *IEEE Symposium on Security and Privacy*, 2002.

[9] N. Asokan, J.E. Ekberg, A.R. Sadeghi, C. Stüble, and M. Wolf. Enabling fairer digital rights management with trusted computing. *In Proc. of the 10th International Conference on Information Security (ISC)*, 2007.

[10] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux. A declarative approach for qoS-aware web service compositions. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Service-Oriented Computing - ICSOC 2007, Fifth International Conference, Vienna, Austria, September 17-20, 2007, Proceedings*, volume 4749 of *LNCS*, pages 422–428. Springer, 2007.

[11] Fabien Baligand, Nicolas Rivierre, and Thomas Ledoux. QoS policies for business processes in service oriented architectures. In Athman Bouguettaya, Ingolf Krüger, and Tiziana Margaria, editors, *Service-Oriented Computing - ICSOC 2008, 6th International Conference, Sydney, Australia, December 1-5, 2008. Proceedings*, volume 5364 of *Lecture Notes in Computer Science*, pages 483–497, 2008.

[12] Davide Balzarotti, Marco Cova, Viktoria Felmetsger, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 387–401. IEEE Computer Society, 2008.

[13] Gilles Barthe and César Kunz. Certificate translation for specification-preserving advices. In *FOAL '08: Proceedings of the 7th workshop on Foundations of aspect-oriented languages*, pages 9–18, New York, NY, USA, 2008. ACM.

[14] M. Bartoletti, P. Degano, and G. Ferrari. Security-aware program transformations. *In Proc. 8th Italian Conference on Theoretical Computer Science*, 2003.

[15] Luis Daniel Benavides Navarro, Mario Südholt, Wim Vanderperren, and Bart Verheecke. Modularization of distributed web services using awed. In *Proc. of the th Int. Conf. on Distributed Objects and Applications (DOA'06*, volume 4276 of *LNCS*, pages 1449–1466. Springer Verlag, October 2006.

[16] Hossein Bidgolie, editor. *Handbook of Information Security, 1st edition*. Wiley, 2005.

[17] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley-Longman, December 2002.

[18] M. Bishop, editor. *Computer Security: The Art and Science*. Addison-Wesley, June 2003.

[19] J. Boch. A puf based hardware architecture for secure car2x application. Technical report, Technical University of Munich, 2009.

[20] M. Boreale and M.G. Buscemi. Experimenting with sta, a tool for automatic analysis of security protocols. *Symposium on Applied Computing, ACM, Lecture Notes in Computer Science*, pages 281–285, 2002.

[21] R. Bunge, S. Chung, B. Endicott-Popovsky D., and McLane. An operational framework for service oriented architecture network security. In *hicss*, page 312. IEEE Computer Society, 2008.

[22] Robert Bunge, Sam Chung, Barbara Endicott-Popovsky, and Don McLane. An operational framework for service oriented architecture network security. *Hawaii International Conference on System Sciences*, page 312, 2008.

[23] Brett Cannon and Eric Wohlstadter. Enforcing security for desktop clients using authority aspects. In *AOSD '09: Proceedings of the 8th ACM international conference on Aspect-oriented software development*, pages 255–266, New York, NY, USA, 2009. ACM.

[24] Luca Carettoni and Stefano di Paola. Information flow analysis for type enforcement policies. *PrÃ©sentation donnÃ©e au SE Linux Symposium*, 2005.

[25] CERT. Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE). `http://www.cert.org/octave/`.

[26] CERT. OCTAVE Criteria, Version 2.0. `http://www.sei.cmu.edu/reports/01tr016.pdf`, 2001.

[27] Anis Charfi and Mira Mezini. Using aspects for security engineering of web service compositions. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 59–66, Washington, DC, USA, 2005. IEEE Computer Society.

[28] Anis Charfi and Mira Mezini. Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, 10(3):309–344, 2007.

[29] Feng Chen, Marcelo d'Amorim, and Grigore Roşu. A formal monitoring-based framework for software development and analysis. In *Formal Methods and Software Engineering, 6th International Conference on Formal Engineering Methods, ICFEM 2004, Proceedings*, volume 3308 of *Lecture Notes in Computer Science*, pages 357–372. Springer-Verlag, 2004.

[30] Feng Chen and Grigore Roşu. MOP: An Efficient and Generic Runtime Verification Framework. In *Object-Oriented Programming, Systems, Languages and Applications, Proceedings (OOPSLA 2007)*, pages 569–588. ACM press, 2007.

[31] B. Chess and G. McGraw. Static analysis for security. *IEEE Security & Privacy*, 2(6):76–79, 2004.

[32] A. Christensen, A. Møller, and M. Schwartzbach. Precise Analysis of String Expressions. In *International Static Analysis Symposium (SAS)*, 2003.

[33] CISCO ACE XML Gateway. `http://www.cisco.com/en/US/products/ps7314/index.html`, 2010.

[34] Jeremy Condit, Matthew Harren, Scott Mcpeak, George C. Necula, and Westley Weimer. Ccured in the real world. *In Proc. of the ACM SIGPLAN Conference on Programming Language Design and Implementation,*, 2003.

[35] SOASIS Consortium. Security assertion markup language v2.0 technical overview. http://wiki.oasis-open.org/security/Saml2TechOverview, 2008.

[36] Roger L. Costello. REST resources. http://www.xfront.com/files/rest.html, 2002.

[37] Carine Courbis and Anthony Finkelstein. Weaving aspects into web service orchestrations. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 219–226, Washington, DC, USA, 2005. IEEE Computer Society.

[38] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: An environment for goal-driven requirements engineering. In *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, pages 612–613, NY, May 17–23 1997. ACM.

[39] Wouter De Borger, Bart De Win, Bert Lagaisse, and Wouter Joosen. A permission system for secure aop. In *AOSD '10: Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, pages 205–216, New York, NY, USA, 2010. ACM.

[40] Y. Demchenko, L. Gommans, C. de Laat, and B. Oudenaarde. Web services and grid security vulnerabilities and threats analysis and model. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 262–267, Washington, DC, USA, 2005. IEEE Computer Society.

[41] Simplice Djoko Djoko, Rémi Douence, and Pascal Fradet. Aspects preserving properties. In Robert Glück and Oege de Moor, editors, *PEPM*, pages 135–145. ACM, 2008.

[42] R. Douence, H. Grall, I. Mejía, et al. Survey and requirements analysis. Deliverable D1.1, The CESSA project, June 2010.
http://cessa.gforge.inria.fr/lib/exe/fetch.php?media=publications:d1-1.pdf .

[43] T. Moses (ed.). extensible access control markup language (xacml) version 2.0. Technical report, OASIS Standard, 2005.

[44] Enterprise JavaBeans. http://java.sun.com/products/ejb.

[45] D. Engler, B. Chelf, A. Chou, and S. Hallem. Checking System Rules Using System-Specific, Programmer-Written Compiler Extensions. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2000.

[46] D. Engler, D. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. In *Symposium on Operating System Principles (SOSP)*, 2001.

[47] Common Weakness Enumeration. CWE/SANS Top 25 Most Dangerous Software Errors. http://cwe.mitre.org/top25.

[48] Defence Signal Directorate et al. CC Part 1: Introduction and general model. http://www.commoncriteriaportal.org/, 2006.

[49] M. Ghader et al. Secure resource and service discovery in personal networks. In *Wireless World Research Forum Meeting*, 2004.

[50] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999.

[51] Eduardo B. Fernandez and Xiaohong Yuan. Securing analysis patterns. In *ACM-SE 45: Proceedings of the 45th annual southeast regional conference*, pages 288–293, New York, NY, USA, 2007. ACM.

[52] International Organization for Standardixation. Information technology – Security techniques – Evaluation criteria for IT security. `http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html`, 2005.

[53] Dan Forsberg. Restful security, 2009.

[54] J. Foster, M. Faehndrich, and A. Aiken. A Theory of Type Qualifiers. In *Conference on Programming Language Design and Implementation (PLDI)*, 1999.

[55] OWASP Foundation. The open web application security project (owasp foundation) owasp testing guide v3.0. `http://www.owasp.org/index.php/OWASP_Testing_Project`.

[56] Pascal Fradet and Stéphane Hong Tuan Ha. Aspects of availability: Enforcing timed properties to prevent denial of service. *Sci. Comput. Program.*, 75(7):516–542, 2010.

[57] S. Gajek, L. Liao, and J. Schwenk. Breaking and fixing the inline approach. In *SWS '07: Proceedings of the 2007 workshop on Secure web services*, 2007.

[58] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. *In Proc. of the Computer and Communication Security Conference*, 2002.

[59] Paolo Giorgini, Fabio Massacci, John Mylopoulos, and Nicola Zannone. St-tool: A case tool for security requirements engineering. *13th International Conference on Requirements Engineering, IEEE*, 2005.

[60] Google. Web-based reference implementation of SAML-based SSO for Google Apps. `http://code.google.com/apis/apps/sso/saml_reference_implementation_web.html`, 2008.

[61] Susanne Graf, Ileana Ober, and Iulian Ober. A real-time profile for UML. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(2):113–127, April 2006.

[62] Trusted Computing Group. Trusted computing group (tcg). `http://www.trustedcomputinggroup.org`, 2003.

[63] N. Gruschka and N. Luttenberger. Protecting web services from dos attacks by soap message validation. *Security and Privacy in Dynamic Environments*, pages 171–182, 2006.

[64] Charles B. Haley, Robin C. Laney, and Bashar Nuseibeh. Deriving security requirements from crosscutting threat descriptions. In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 112–121, New York, NY, USA, 2004. ACM.

[65] W. Halfond and A. Orso. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In *International Conference on Automated Software Engineering (ASE)*, pages 174–183, November 2005.

[66] Sylvain Hallé, Tevfik Bultan, Graham Hughes, Muath Alkhalaf, and Roger Villemaire. Runtime verification of web service interface contracts. *Computer*, 43(3):59–66, 2010.

[67] Kevin Hamlen, Greg Morrisett, and Fred Schneider. Computability classes for enforcement mechanisms. *ACM Transactions on Programming Languages and Systems*, 28(1):175–205, 2006.

[68] Eran Hammer-Lahav. OAuth : Security Architecture. `http://hueniverse.com/2008/10/beginners-guide-to-oauth-part-iii-security-architecture/`, 2008.

[69] Ubaid Hayee. A six signa approach to assure it security. Technical report, The Royal Institute of Technology (KTH), 2008.

[70] W. Heaven and A. Finkelstein. A uml profile to support requirements engineering with kaos. *IEE Proceedings, 2004.*, 2004.

[71] Ian Jacobi Henry Story, Bruno Harbulot and Mike Jones. Foaf+tls: Restful authentication for distributed social networks, 2009.

[72] Y. Huang, F. Yu, C. Hang, C.. Tsai, D. Lee, and S.. Kuo. Securing Web Application Code by Static Analysis and Runtime Protection. In *12th International World Wide Web Conference (WWW)*, 2004.

[73] Y. Huang, F. Yu, C. Hang, C. Tsai, D. Lee, and S. Kuo. Verifying Web Applications Using Bounded Model Checking. In *Conference on Dependable Systems and Networks (DSN)*, 2004.

[74] Y.W. Huang, S.K. Huang, T.P. Lin, and C.H. Tsai. Web application security assessment by fault injection and behavior monitoring. In *Proceedings of the 12th international conference on World Wide Web*, pages 148–159. ACM, 2003.

[75] The Internet Engineering Task Force (IETF). The OAuth 1.0 Protocol. `http://tools.ietf.org/html/rfc5849`, 2010.

[76] The Internet Engineering Task Force (IETF). The OAuth 2.0 Protocol. `http://tools.ietf.org/html/draft-ietf-oauth-v2-10`, 2010.

[77] SANS Institute. SANS Institute. `http://www.sans.org/`.

[78] IntelliGrid, editor. *General Security Process*. Electric Power Research Institute, 2004.

[79] IntelliGrid. Security organization in the intelligrid architecture. `http://intelligrid.ipower.com/IntelliGrid_Architecture/Technology_Analysis/Anl_Security_Overview.htm`, 2004.

[80] International IT security evaluation community. *Common Methodology for Information Technology Security Evaluation (CEM), Evaluation Methodology, Version 3.1, Rev. 2, CCMB-2007-09-004, September 2007*.

[81] Internet2. Shibboleth Project. Available at `http://shibboleth.internet2.edu/`, 2007.

[82] Cynthia Irvine. The reference monitor concept as a unifying principle in computer security education. In *Proceedings of the First World Conference on Information Systems Security Education*, pages 27–37, 1999.

[83] M. Jensen, N. Gruschka, and N. Luttenberger. SOA and Web Services: New Technologies, New Standards-New Attacks. In *the 5th IEEE European Conference on Web Services*, 2007.

[84] Jif: Java + Information Flow. `http://www.cs.cornell.edu/jif/`, 2007.

[85] Jamie Pole John Wack, Ken Cutler. Guidelines on firewalls and firewall policy. *Recommendations of the National Institute of Standards and Technology*.

[86] R. Johnson and D. Wagner. Finding User/Kernel Pointer Bugs With Type Inference. In *13th USENIX Security Symposium*, 2004.

[87] Micah Jones and Kevin W. Hamlen. Disambiguating aspect-oriented security policies. In *AOSD '10: Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, pages 193–204, New York, NY, USA, 2010. ACM.

[88] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In *IEEE Symposium on Security and Privacy*, 2006.

[89] N. Jovanovic, C. Kruegel, and E. Kirda. Precise Alias Analysis for Static Detection of Web Application Vulnerabilities. In *ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, 2006.

[90] Jan Jürjens and Pasha Shabalin. Automated verification of UMLsec models for security requirements. In Thomas Baar, Alfred Strohmeier, Ana M. D. Moreira, and Stephen J. Mellor, editors, *"UML" 2004 - The Unified Modelling Language: Modelling Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings*, volume 3273 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2004.

[91] Jan Jürjens. Umlsec: Extending uml for secure systems development. *5th International Conference on the Unified Modeling Language*, pages 412–425, 2002.

[92] R. Kaksonen, M. Laakso, and A. Takanen. Software security assessment through specification mutations and fault injection. *Proceedings of Communications and Multimedia Security Issues of the New Century*, 2001.

[93] O. Kath, M. Soden, M. Born, T. Ritter, A. Blazarenas, M. Funabashi, and C. Hirai. An open modelling infrastructure integrating edoc and ccm. *7th International Enterprise Distributed Object Computing Conference, IEEE.*, 2003.

[94] Emilia Katz and Shmuel Katz. Verifying scenario-based aspect specifications. In *Proc. of the Int. Symposium of Formal Methods Europe (FM'05)*, pages 432–447. Springer Verlag, 2005.

[95] Shmuel Katz and Marcelo Sihman. Aspect validation using model checking. In Nachum Dershowitz, editor, *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 373–394. Springer Verlag, 2003.

[96] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Aksit and Satoshi Matsuoka, editors, *11th Europeen Conference on Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer-Verlag, 1997.

[97] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(2):2–16, 2005.

[98] Jay Ligatti, Lujo Bauer, and David Walker. Enforcing non-safety security policies with program monitors. In *Computer Security – ESORICS 2005: 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 355–373. Springer-Verlag, 2005.

[99] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information and System Security*, 12(3):1–41, 2009.

[100] Pete Lindstrom. Attacking and Defending Web-Services. *Spire Research Report*, 2003.

[101] Jing Liu Ling Yin and Xiaoshan Li. Validating requirements model of a b2b system. In *8th International Conference on Computer and Information Science, IEEE/ACIS*, 2009.

[102] B. Livshits and M. Lam. Finding Security Vulnerabilities in Java Applications with Static Analysis. In *14th USENIX Security Symposium*, pages 271–286, August 2005.

[103] H. Lockhart. Demystifying SAML. http://dev2dev.bea.com/pub/a/2005/11/saml.html, 2005.

[104] T. Lodderstedt, D.A. Basin, and J. Doser. Secureuml: A uml-based modeling language for model driven security. *5th International Conference on the Unified Modeling Language*, pages 426–441, 2002.

[105] N. Looker and J. Xu. Assessing the Dependability of SOAP RPC-Based Web Services by Fault Injection. In *Proceedings of the 9th international Workshop on Object-Oriented Real-Time dependable systems*. IEEE, 2003.

[106] Boris Lublinsky. Dealing with REST Services Security. `http://www.infoq.com/news/2010/03/REST_security`, 2010.

[107] Karl MacMillan. Ahttp parameter pollution. *AppSec Europe 2009 Web Security Conference , OWASP*, 2009.

[108] Michael McIntosh and Paula Austel. Xml signature element wrapping attacks and countermeasures. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 20–27, New York, NY, USA, 2005. ACM.

[109] Chuck Mcmanis. The basics of java class loaders: he fundamentals of this key component of the java architecture. http://www.javaworld.com/javaworld/jw-10-1996/jw-10-indepth.html, 1996.

[110] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of UNIX utilities. *Communications of the ACM*, 33(12):32–44, dec 1990.

[111] Libby Miller and Dan Brickley. Friend of a Friend Project (FOAF). `http://www.foaf-project.org`.

[112] Richard Mooney. Security and REST Web Services . `http://2007.xtech.org/public/schedule/detail/37`, 2007.

[113] Alan Moore. Extending the rt profile to support the osek infrastructure. *5th International Symposium on Object-Oriented Real-Time Distributed Computing, IEEE.*, 2002.

[114] A. Myers. JFlow: Practical Mostly-Static Information Flow Control. In *Symposium on Principles of Programming Languages (POPL)*, 1999.

[115] Andrew C. Myers. JFlow: practical mostly-static information flow control. In ACM, editor, *POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, pages 228–241, pub-ACM:adr, 1999. ACM Press.

[116] H. Washizaki N. Yoshioka and K. Maruyama. A survey on security patterns. *Progress in Informatics*, 5:35–47, 2008.

[117] George Necula, Scott McPeak, Westley Weimerand Matthew Harren, and Jeremy Condit. CCured Documentation. `http://hal.cs.berkeley.edu/ccured/`.

[118] CITRIX NetScaler. http://www.citrix.com/english/ps2/products/product.asp?contentid=21679, 2010.

[119] NSTISSP. National Information Assurance Acquisition Policy. http://www.cnss.gov/Assets/pdf/nstissp_11_fs.pdf, 2003.

[120] Scott Oake, editor. *Java Security, 2nd edition*. O Reilly, 1998.

[121] OASIS. Identity Federation. Liberty Alliance Project. Available at http://www.projectliberty.org/resources/specifications.php, 2004.

[122] American Institute of Certified Public Accountants (AICPA). Statement on Auditing Standards No. 70. http://www.sas70.us.com/.

[123] National Institute of Standards and Technology. NIST. http://nist.gov/.

[124] OMG. The object management group (omg), The EDOC Specification. http://www.omg.org/mda.

[125] OMG. *OMG UML Specification*. OMG, June 13 1999. Version 1.3.

[126] Guadalupe Ortiz and Behzad Bordbar. Aspect-oriented quality of service for web services: A model-driven approach. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 559–566, Washington, DC, USA, 2009. IEEE Computer Society.

[127] OWASP. OWASP Top 10. http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[128] R. Pappu. Physical one-way functions. Technical report, Massachusetts Institute of Technology, 2001.

[129] Don Park. Building Web Services the REST Way. http://www.stylusstudio.com/xmldev/200207/post70140.html, 2002.

[130] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 805–814, New York, NY, USA, 2008. ACM.

[131] Karine P. Peralta, Alex M. Orozco, Avelino F. Zorzo, and Flávio M. Oliveira. Specifying security aspects in uml models *.

[132] Gunnar Peterson. REST Security (or lack thereof). http://1raindrop.typepad.com/1_raindrop/2006/12/rest_security_o.html, 2006.

[133] M.A. Rahaman, A. Schaad, and M. Rits. Towards secure soap message exchange in a soa. In *Proceedings of the 3rd ACM workshop on Secure web services*, page 84. ACM, 2006.

[134] Mohammad Ashiqur Rahaman, Rits Marten, and Andreas Schaad. An inline approach for secure soap requests and early validation. *European Conference on Open Web Application Security Project , OWASP*, 2006.

[135] Christopher Steel Ramesh Nagappan and Ray Lai. *Core security patterns: Best practices and strategies for j2ee, web services, and identity management*. Prentice Hall PTR, 7 2003.

[136] MISSOURI RESEARCH. An Introduction to Network Firewalls and the Firewall Selection Process. `http://www.more.net/technical/netserv/tcpip/firewalls/`.

[137] SANS Institute InfoSec Reading Room. XML Firewall Architecture and Best Practices for Configuration and Auditing. `http://www.sans.org/reading_room/whitepapers/firewalls/xml-firewall-architecture-practices-configuration-auditing_1766`, 2007.

[138] A. Ruddle, D. Ward, B. Weyl, S. Idrees, Y. Roudier, M. Friedewald, T. Leimbach, A. Fuchs, S. Gürgens, O. Henniger, R. Rieke, M. Ritscher, H. Broberg, L. Apvrille, R. Pacalet, and G. Pedroza. Security requirements for automotive on-board networks based on dark-side scenarios. Technical Report Deliverable D2.3, EVITA Project, 2009.

[139] Saltzer, Jerome H., Schroeder, and Michael D. The protection of information in computer systems. *Proc.of the IEEE*, 63(9), September 1975.

[140] Fred Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[141] Markus Schumacher. *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[142] Markus Schumacher, Eduardo Fernandez, Duane Hybertson, and Frank Buschmann. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2005.

[143] Amazon Web Services. Authentication for Amazon Simple Storage Service. `http://docs.amazonwebservices.com/AmazonS3/S3_Authentication.html`, 2006.

[144] U. Shankar, K. Talwar, J. Foster, and D. Wagner. Detecting Format String Vulnerabilities with Type Qualifiers. In *10th USENIX Security Symposium*, 2001.

[145] Anoop Singhal, Theodore Winograd, and Karen Scarfone. Guide to Secure Web Services. NIST Publication.

[146] C. Skalka and X. Wang. Trust by verify: Auhorization for web services. In *ACM Workshop on Secure Web Services*, 2004.

[147] Michael G. Solomon and Mike Chapple, editors. *Information Security Illuminated*. Jones and Bartlett Publishers, Inc., 2005.

[148] Spike. http://www.immunitysec.com/ resources-freesoftware.shtml.

[149] OASIS Standard. OASIS Web Services Security (WSS). http://www.oasis-open.org/committees/wss.

[150] OASIS Standard. Kerberos Token Profile 1.1. http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf, 2006.

[151] OASIS Standard. SAML Token Profile 1.1. http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf, 2006.

[152] OASIS Standard. UsernameToken Profile 1.1. http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf, 2006.

[153] OASIS Standard. X.509 Certificate Token Profile 1.1. http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf, 2006.

[154] OASIS Standard. Web Services Federation Language (WS-Federation) V1.2. http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html, 2009.

[155] OASIS Standard. WS-SecurityPolicy 1.3. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.html, 2009.

[156] OASIS Standard. WS-Trust 1.4. http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.html, 2009.

[157] Henry Story. FOAF+SSL: creating a web of trust without key signing parties. http://blogs.sun.com/bblfish/entry/more_on_authorization_in_foaf, 2009.

[158] Sun. The Java Virtual Machine Specification. http://java.sun.com/docs/books/jvms/.

[159] Forum Systems. Anatomy of a Web Services Attack: A Guide to Threats and Preventative Countermeasures. http://www.forumsystems.com/, 2004.

[160] Z. Tari, P. Bertok, and D. Simic. A dynamic label checking approach for information flow control in web services. In *International Journal of Web Services Research*, 2006.

[161] Geoff Tennant, editor. *SIX SIGMA: SPC and TQM in Manufacturing and Services*. Gower Publishing, Ltd, 2001.

[162] Slim Trabelsi. *Spontaneous security services for ubiquitous computing*. PhD thesis, Thesis, 07 2008.

[163] Trusted Computing Group. *Trusted Computing Group (TCG): TCG Software Stack (TSS) Version 1.2, Level 1, Errata A, Technical Report, March 2007*.

[164] Bill Venners, editor. *Inside The Java Virtual Machine, 2nd edition*. McGraw-Hill Companies, 2000.

[165] W3C. XML Encryption Syntax and Processing. `http://www.w3.org/TR/xmlenc-core/`, 2002.

[166] W3C. XML Signature Syntax and Processing. `http://www.w3.org/TR/xmldsig-core/`, 2008.

[167] D. Wagner and D. Dean. Intrusion Detection via Static Analysis. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2001. IEEE Press.

[168] D. Wagner, J.S. Foster, E.A. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In *Network and Distributed System Security Symposium*, pages 3–17. Citeseer, 2000.

[169] Wallach, D.S., Felten, and E.W. Understanding java stack inspection. *In Proc. IEEE Symposium on Security and Privacy*, 1998.

[170] Zheng Wang, Xiao Yu, Geguang Pu, Libo Feng, Huibiao Zhu, and Jifeng He. Execution semantics for rcos. *15th Asia-Pacific Software Engineering Conference, IEEE*, 2008.

[171] Sun Wenhui, Wu Jinzhao, and Xiong YuQing. Methodological support for service-oriented design with rcos. *International Symposium on Information Engineering and Electronic Commerce, IEEE.*, 2009.

[172] J. Whaley and M. Lam. Cloning-Based Context-Sensitive Pointer Alias Analysis Using Binary Decision Diagrams. In *Conference on Programming Language Design and Implementation (PLDI)*, 2004.

[173] WS-I. The Web Services Interoperability Organization (WS-I). `http://www.ws-i.org/`.

[174] WS-I. Basic Security Profile Version 1.1. `http://www.ws-i.org/Profiles/BasicSecurityProfile-1.1.html`, 2010.

[175] Wsdigger. `http://www.foundstone.com/resources/s3i_tools.htm`.

[176] Y. Xie and A. Aiken. Static Detection of Security Vulnerabilities in Scripting Languages. In *15th USENIX Security Symposium*, August 2006.

[177] Dianxiang Xu, Vivek Goel, Kendall E. Nygard, and W. Eric Wong. Aspect-oriented specification of threat-driven security requirements. *Int. J. Comput. Appl. Technol.*, 31(1/2):131–140, 2008.

[178] Andree Yee. Protecting your Web Services Deployment. http://www.ebizq.net/topics/security/features/4840.html, 2004.

[179] E. Yuan and J.Tong. Attribute based access control (ABAC) for web services. In *2005 IEEE International Conference on Web Services*, 2005.

[180] X. Zhang, A. Edwards, and T. Jaeger. Using CQUAL for static analysis of authorization hook placement. In *11th USENIX Security Symposium*, 2002.