Abstract. Security analysis allows one to delimit the security perimeter of a computer system. In service oriented architectures, such task is intrinsically complex, due to the many architectural layers, technologies and communication protocols involved. The security analysis must also consider the particular implementation for a given SOA. In this deliverable we first introduce different kind of attacks that are related to web-services and embedded devices, to then cover threats that appear in presence of service composition. We also present accepted methodology with proposition of a generic attacker model that can be instantiated for different SOA settings, which allow an analysis of specific categories related to SOA. Finally, we carry out the security analysis of the main CESSA use-cases.
Contents

1 Introduction .................................................. 6
   1.1 Purpose and Scope .................................. 6
   1.2 Outline of the Document ......................... 7

2 Web-Services Infrastructure and Embedded Devices Security Analysis .... 8
   2.1 SOA Vulnerability Analysis ....................... 8
   2.2 Web Services Attack Classifications ............ 10
   2.3 Countermeasures for Deployed Web Services ... 12
   2.4 Embedded Devices Analysis ..................... 15
   2.5 Embedded Java Technology Analysis .......... 16
      2.5.1 J2SE State of the Art ..................... 16
      2.5.2 Embedded Specific Technologies ......... 17

3 Service Composition Security Analysis ................................ 18
   3.1 Vulnerability Analysis for Service Compositions .... 18
      3.1.1 Service Orchestration Vulnerabilities ...... 18
      3.1.2 Service Choreography Vulnerabilities ...... 19
   3.2 Service Security Patterns .......................... 20

4 Threat and Risk Analysis Methodology .................................. 22
   4.1 Attacker Meta-Model .................................. 22
      4.1.1 Active View .................................. 23
      4.1.2 Passive View .................................. 24
   4.2 Risk Analysis Methodology ......................... 25
      4.2.1 Asset Definition ............................... 25
      4.2.2 Attack identification ........................ 26
      4.2.3 Risk Analysis ................................ 27

5 Use Cases Security Analysis .............................................. 28
   5.1 Loan Negotiation Use-Case Security Analysis .... 28
      5.1.1 Overall Description of the Scenario with Potential Attacks .... 29
      5.1.2 Sub Scenario 1 - External Audit ............ 31
      5.1.3 Sub Scenario 2 - Regulation Requirement Change Process .... 32
5.2 Home Automation Use-Case Security Analysis

5.2.1 Overview

5.2.2 Potential Attacks

5.2.3 Use cases

6 Conclusion

Bibliography
List of Figures

4.1 Methodology steps .................................................. 26
5.1 Loan negotiation Negotiation Workflow with Attacks .................. 30
5.2 Loan adversary attacks ............................................. 32
5.3 Loan Scenario, a Partial attack tree .................................. 33
5.4 Multi domain logging evolution ..................................... 34
5.5 General schema of the HAP system .................................. 35
5.6 Home Automation Burglary Workflow ................................. 38
List of Tables

4.1 Attacker model for SOA systems ........................................ 22
5.1 Assets per actor .......................................................... 31
5.2 Assets per actor for PF4U ............................................... 39
Chapter 1

Introduction

Security analysis determines which assets should be protected and the level of that protection. It involves the identification of potential threats to assets and the degree of likelihood an attack will occur. This degree will determine the priority in which a threat has to be protected against. The main goal of the present deliverable is to perform the security analysis of the CESSA use cases. By identifying the main vulnerabilities involved in the industrial settings under consideration, we will be able to correctly develop security mechanisms able to mitigate the corresponding vulnerabilities.

1.1 Purpose and Scope

We study two main use cases in this deliverable. The first use case is proposed by SAP and concerns an SOA based system where loan negotiation takes place. The second use case, proposed by IS2T involves state of the art embedded systems technology.

The perspective we have adopted in the current report for the security analysis is to have a unified approach where we start from the business process to identify threats at different architectural levels. We have adopted a set of practices from fundamental works in vulnerability analysis in a new combined methodology that seems to be well-adapted to SOA’s at the same time it is extensible to embedded devices.

In order to achieve that, we survey known vulnerabilities concerning web-service based SOA’s. The survey is structured in two parts: the first reports vulnerabilities related to web-services infrastructure, where services in isolation can be subject to attacks. The second part considers web service compositions, where we discuss both vulnerabilities in service orchestrations and choreographies. A fundamental part of the current document is the study of security (or vulnerability) analysis methodologies, mainly those recently proposed for SOA’s, which served as a basis for the analysis we perform in our case studies.

The core contributions rely on the use case analysis of the scenarios of interest for CESSA. The main outcomes are the threat trees where one can clearly identify the main possible threats to the assets in each sub-scenario.
1.2 Outline of the Document

The deliverable is organized as follows

- Chapter 2 surveys vulnerabilities and attacks concerning the web-service infrastructure: protocols, data formats (XML related vulnerabilities), and architectural components, such as orchestration engines. We also survey vulnerabilities on the embedded devices technological frameworks.

- Chapter 3 is dedicated to known security issues specific to service composition.

- Chapter 4 presents the methodology we have adopted for the security analysis of our different scenarios. It brings a brief overview of existing methodologies and presents the approach we propose for CESSA.

- Chapter 5 presents the detailed analysis of our use cases.

- Chapter 6 concludes the deliverable and discusses next steps and future works in the context of CESSA.
Chapter 2

Web-Services Infrastructure and Embedded Devices Security Analysis

In this chapter we survey known vulnerabilities relating to the elements that compose the web service infrastructure, that is: any known attacks on the web-service specific protocols, software components, and enabling technologies, such as XML and related standards and tools. We have dedicated a special interest in the security issues related to smart items and supporting technologies such as customized virtual machines. The security analysis of web services compositions is explored in Chapter 3.

Security analysis for web-services based SOA’s has been the subject of a growing interest for the security research community. The preoccupation in anticipating possible security flaws in the SOA’s infrastructures is fundamental for increasing the reliability of SOA’s, such that it can be widely adopted, enabling the future Internet of Services. In this section we make a review of the available literature on the “security analysis” of web services. This state of the art allows us to identify the best principles used in the identification of security requirements for SOA.

2.1 SOA Vulnerability Analysis

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. We begin our discussion with the work presented in [29]. It introduces a methodology for performing the vulnerability analysis on SOA business processes called ATLIST. The main advantage of the methodology is to build on the existing knowledge about software, web application, and on protocol vulnerabilities and classifications, to provide an more traceable guidelines for the the security analyst, which would need not to rely only on his/her experience, which is a drawback in previous works [51, 9, 58]. The main originality of ATLIST is to introduce a new element to the classic attack tress, which is the point of view (POV). In [29] it is advocated that the POV of the operating system would indicate different attack effects with respect to the POV of SOA. Determining the POV will influence the choice of the other elements composing an ATLIST attack tree, namely the attack effects, the active components (first component where a vulnerability can
be exploited), the involved standards, and finally the properties (attributes of these elements) that can trigger a vulnerability (e.g. the size of a SOAP message). A nice feature of ATLIST is that the elements forming an attack tree can be instantiated using examples and identified vulnerabilities from other classifications, vulnerability databases, and risk analysis frameworks, such as Microsoft’s DREAD (for Damage potential, Reproducibility, exploitability, Affected users, and Discoverability) [57]. This is justified by the fact that new exploits are not likely to appear. In addition to that, the approach allows to provide input for possible vulnerability scanning tools, provided that attack patterns can be precisely identified and formally specified using some notation (such as Program Query Language [31], for instance). Such feature is not explored further in [29], where only some toy examples and simple code snippets are presented. The analysis of an ATLIST tree must comprise several points that help the security analyst to mitigate the risk: the preconditions under which an attack can happen, the post-conditions (or the changes in the system after a given attack was performed), test cases (for checking that specific vulnerabilities), the protection measures that need to be taken, and the possible search pattern, that can be fed to a scanning tool, provided that it is possible to describe the vulnerability in the tool’s input language. The methodology is well structured but it sounds somewhat orthodox, since not much is said about how the business processes are supposed to be described and studied under the ATLIST perspective - how to identify assets and how to determine the expected correct behavior of the system. It is hard to assess whether the claims are correctly addressed, since the case study presented is not sufficiently detailed.

A previous related work [28] brings a classification of the new vulnerabilities (when compared to the well studied vulnerabilities in operating systems and on web applications) introduced by the implementations of SOA related standards, such as BPEL and SOAP. The goal of [28] is to create a classification that can be applied in the construction of tools to support SOA security - such tools often rely on vulnerability patterns, which are usually derived from classifications. This work separates a SOA in five different layers here given from bottom to top: hardware, software, services, orchestration and presentation. The authors focus on the orchestration layer, exploring vulnerabilities that allow attackers to modify the activities of their sequence in a business process. The vulnerabilities concerning the other layers are classified using previous existing approaches. The authors define what they call the level of “survivability” of a vulnerability in SOA.

This has to do with the technical assumptions necessary to exploit the vulnerability. The evolution of the SOA into new environments would prevent some of the identified vulnerabilities to be further exploited. Along with that, in order to avoid inherent ambiguities in vulnerability classifications, the work appends two dimensions to the classification: the viewpoint and the abstraction level, in this case, the orchestration layer and the viewpoint is that of the business process being executed.

The main contribution of is the (incomplete) classification in [28] is the exploitability assumption dimension based on concrete attacks, which gives some elements to drive the risk analysis phase. The exploitability is provided for each vulnerability. For instance, an XML injection attack is exploitable assuming that there is an incomplete schema validation. Another example is WS-address spoofing with can be exploited under the assumption that there is an unverified endpoint URL in use. At the same time this approach can be seen simply as an al-
ternative manner to display the threat classification, since what they call “vulnerability” in the examples above are in fact attacks, and “exploitability assumptions” are the actual vulnerabilities. The importance of the approach [28] really lies on the focus on service orchestration. 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 [17].

2.2 Web Services Attack Classifications

In this section we briefly overview the list of web services attack classifications. However, more detail web service vulnerability analysis can be found in [24].

- **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.

- **Discovery, Registry, and Publishing Layer Attacks**:
  
  - **Service Discovery Vulnerabilities**: In the case of service discovery, the attacker can perform attack at multi stages:
* 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.

* 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.

* 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.

* 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.

– Service Registration Vulnerabilities: In centralized architectures, we can observe the following kinds of vulnerabilities:

* 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.

* A service can be de-registered by an unauthorized party (registry-side): This occurs when an attacker tries to dereference an active service from the registry which it registered previously.

* Fake registration (registry-side): An attacker can send a fake registration message to the registry containing wrong information with fake attributes.

– 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.

– Matching Process - 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).

• 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.

- RESTful web services Vulnerabilities: 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 [7]. 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.

2.3 Countermeasures for Deployed Web Services

In this section, we are going to focus on the well-know WS-Security standards and will talk about some related standards and models. However, more detail overview of these model and standards is provided in [24, 12]

Web Services Security

WS-Security defines a SOAP Security Header format containing security related information. A SOAP message may include multiple security headers. Each header is targeted at a specific SOAP actor/role that may be either the ultimate recipient of the message or an intermediary. Security headers may encapsulate one or many elements of the following types:

- Security tokens
- Signatures
- Encryption elements
- Timestamps

By providing a common syntax and a flexible processing model for security headers, this specification accommodates a large variety of security models and encryption technologies. Moreover, incorporating security features in the application level ensures end-to-end security.

Web Services Trust

This specification provides a framework built on WS-Security for managing security tokens. In the WS-Trust trust model, a requester examines the policy associated with a Web Service to identify the claims it needs. If the policy statements require security tokens that the requester does not possess, WS-Trust specifies a way of obtaining them: contacting a Web Service referred to
as Security Token Server (STS). A STS may also be used to renew, cancel and validate security tokens. WS-Trust defines abstract formats of the messages used to manage security tokens. To each usage pattern corresponds a specific binding providing concrete semantics to the general security token requests and responses. For complex scenarios, WS-Trust describes flexible mechanisms for trust establishment. In fact, different STS may get involved to broker, exchange or delegate security tokens issuance. A general model for negotiation/challenge extensions is specified to support multi-messages exchanges for security tokens management. The flexibility and extensibility of the specification allows interfacing with a large number of security models, including legacy protocols. In fact, increasing interoperability between trust domains is one of the purposes of this standard.

**Web Services Policy**

The WS-Policy is a policy expression language for describing the capabilities and requirements of a Web Service, i.e. representing whether and how a message must be secured, whether and how a message must be delivered reliably or whether the request must follow a transaction flow. Such requirements are translated into machine-readable policy expressions that are usually provided by the web service developer for the client component to automatically apply the requirements. Basically, WS-Policy is a simple language that defines four elements (Policy, All, ExactlyOne, PolicyReferences) and two attributes (Optional, Ignorable) that suffice to express generic policy expression by combining individual assertions. The policy assertions syntax are outside the scope of WS-Policy specifications. Thus, WS-Policy can be viewed as a meta policy composition language that can express any kind of requirements as long as the policy-aware clients (Web Services endpoints and relays) are capable of understanding the specific syntax of the unitary assertions. An individual policy assertion expresses one requirement, behavior or capability related to messaging (how the message must be built), security (how the message must be secured through authentication or encryption), reliability (how to ensure that the message has been sent/received) and transaction (what transaction flow must be followed to ensure transaction commit).

**Web Services Federation**

The WS-Federation specification [54] defines mechanisms to support federation between different security realms, i.e., the authorized access for principals of one realm to resources managed by another. The WS-Federation framework builds on the specification for WS-Security and WS-Trust. Specifically, WS-Federation relies on the Security Token Service (STS) model defined by WS-Trust, and a protocol (involving Request Security Token and RST Response messages) for handling such tokens, which contain information described by WS-SecurityPolicy [55]. The STS is used to broker an establishment of a trust relationship between resource providers / relying parties and other service providers. The goal is to simplify the development of federated services by reusing the WS-Trust STS model and protocol. Different federation services can be developed as variations of the base STS. Processing in WS-Federation is kept independent of the security token format and the type of token being transmitted. WS-Federation defines a metadata
model and a document format describing how services can be discovered and combined, as well as their access policies.

**Web Services Reliability**

WS-Reliability is a SOAP-based specification for reliable messaging requirements \cite{37}. Subsequently to its standardization in 2004 (v. 1.1), the WSReliable Messaging specification was developed by the same OASIS Technical Committee. WS-Reliability separates reliable messaging issues into a protocol (wire) aspect which deals with the horizontal contract between sender and receiver (e.g., message headers, choreography) and a quality of service aspect which deals with the vertical contract between service provider and service users. The latter defines a set of abstract operations on messages (such as Deliver, Submit, Respond and Notify). The specification assumes transparency of SOAP intermediaries and support for message integrity (e.g., as in WS-Security).

**Restful WS Security**

The RESTful security has been partially covered in \cite{24}, to explain that REST security is still an open question. There is currently no security model specifically defined for RESTful services, as basically there is no stated-definition. Goal being to be self-descriptive, the security model is then to be taken case by case.

There are two categories where security issues are widely discussed: secure coding, and secure communication. The secure coding part tends to be the same as traditional WS-* stack, or how to avoid introduction of vulnerabilities in web applications, such as Cross-Site Scripting (XSS) or Injections (XPath, SQL, LDAP, etc.), Path Traversal, etc. The secure communication is more about how to manage identity federation, or confidentiality and integrity while each service and client are communicating.

The Amazon S3 model is one of the popular model to add access control and authentication mechanisms. Instead of dealing with password fashion protection, X.509 Certificate, the Authentication mechanisms support one shared-secret to add signature to each requests. Then, authentication combined with access control prevents unauthorized users from accessing data, modifying data, deleting data, or using the account for services that cost 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 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 \cite{13}, 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”.

The OAuth model is another model in the category of secure communication. It’s "an open protocol to allow secure API authorization in a simple and standard method from desktop and web applications” [25]. 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 described in [21] 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 [26].

Another example is the FOAF+SSL protocol, action as 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 [33] 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. 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 [56]. This is another example of how we can use RESTful authentication [22].

RESTful security has been covered by [14, 48, 30, 34, 8], and best principle of security is to encrypt the communication every-time there is a sensitive data being transmitted. Also, having a hash-based message authentication code (HMAC) like in Amazon S3 model is a good idea to ensure authenticity and validity of messages. The problem as stated in [47] 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”. Then REST security is also about being able to recognize and protect against classical Web threats [46].

2.4 Embedded Devices Analysis

Beyond the scope of pure WS security, this section deals with hardware-based protections for embedded devices and link them with services and security. The following analysis will be divided in three main components: authentication, encryption and obfuscation. The combined
use of the three techniques ensure a maximum hardware protection for the software.

- **Authentication**: Hardware and software shared an authentication key. The code is then called a signed code. The hardware holds an hardware-based key that cannot be violated without damaging the device. Moreover any non-signed code simply will not run without being properly authenticated.

- **Encryption**: The running code is encrypted with a hardware-known key, event though an attacker get back the binary code inside a flash memory, it stills very hard to get back the original source code.

- **Obfuscation**: The running code is obfuscated it means non human-readable once decompiled. The obfuscation reduces also the code size by reducing variable and method names.

### 2.5 Embedded Java Technology Analysis

This section deals with software protection that can be applied to Services security and code isolation.

As stated on [41], the Java language has some built-in security mechanisms such as:

- Strong data typing
- Automatic memory management
- Byte code verification
- Secure class loading

It provides a safe and secure platform for developing and running applications. Compile-time data type checking and automatic memory management leads to more robust code and reduces memory corruption and vulnerabilities. Byte-code verification ensures code conforms to the JVM specification and prevents hostile code from corrupting the runtime environment. Class loaders ensure that untrusted code cannot interfere with the running of other Java programs.

#### 2.5.1 J2SE State of the Art

In standard edition of the technology some mechanisms are provided to secure Java code through the use of several specifications and API:

- **Encryption (JCA [39])**: The JCA is a major piece of the platform, and contains a ”provider” architecture and a set of APIs for digital signatures, message digests (hashs), certificates and certificate validation, encryption (symmetric/asymmetric block/stream ciphers), key generation and management, and secure random number generation, to name a few. These APIs allow developers to easily integrate security into their application code.
• **Authentication (JAAS [38]):** JAAS can be used for two purposes: for authentication of users, to reliably and securely determine who is currently executing Java code, regardless of whether the code is running as an application, an applet, a bean, or a servlet; and for authorization of users to ensure they have the access control rights (permissions) required to do the actions performed.

• **Secure communications (JSSE [42], JGSS [40], Java SASL [43]).**

• **Code Isolation with OSGi [44]:** OSGi technology is the dynamic module system for Java™. The OSGi technology provides the standardized primitives that allow applications to be constructed from small, reusable and collaborative components. These components can be composed into an application and deployed.

### 2.5.2 Embedded Specific Technologies

New technologies must be developed for security, authentication and code isolation. Specifications that mimics functionalities available in the standard edition of the Java platform but with embedded devices design in mind.

• **AOP:** IS2T has its own aspect-oriented programming mechanism which is smaller; in execution time and also in code size; but equivalent to the well-known AspectJ. This technology allows to insert code after compilation, it means that services security rules can be changed without changing the behavior of the service.

• **Authentication:** IS2T uses a strong authentication module based on matrices and software checksums. This module allows authentication between a user and a device and between devices. It allows also authentication between OSGi bundles.

• **Encryption:** the bouncy-castle API [6] which implements JCA for J2ME profiles can be used.

• **OSGi ME:** OSGi ME provides a general-purpose, secure, and managed Java framework that supports the deployment of extensible and potentially downloadable components known as bundle. Installed bundles can register a number of services that can be shared with other bundles under strict control of the OSGi ME framework. It allows code isolation between bundles. Each services provide can be viewed as bundle in OSGi framework.

These security aspects form a global security mechanism that can be applied to a service-oriented architecture. Indeed each service can be viewed as an OSGi bundle. OSGi bundles are perfectly isolated pieces of code that expose functionalities and import functionalities from and to other bundles. That is the framework which is in charge of making the bundles interact with each other but there are no restriction on the calls made from and to bundles. That is where the aspect programming paradigm is important to the project without changing the business logic of a bundle you can allow or restrain calls to other bundles, in brief you create a trust group between bundles. Moreover to ensure that a bundle has the right to be executed by the framework it needs to be authenticated.
Chapter 3

Service Composition Security Analysis

Composed services are the main contribution the SOA’s bring to enterprise business process automation. As the standards and technology rapidly evolves it is necessary to understand the security threats and requirements in order to provide reliable service compositions, specially in the context of the CESSA project.

This chapter presents the state of the art concerning security vulnerabilities intrinsic to service compositions, focusing on the security issues related to workflow languages and engines. We also survey in Section 3.2 the adoption of strategies for securing SOA’s through the application of security patterns. The use of security patterns anticipates possible vulnerabilities by implementing common mechanisms that are able to increase the resistance of the architecture to security problems and to attacks.

3.1 Vulnerability Analysis for Service Compositions

Service orchestration and choreography are the two main approaches to service composition. They were properly introduced in [12]. From the perspective of the CESSA project, orchestration and choreography are complementary ways of describing rich peer-to-peer SOA environments.

In the following, we separate our analysis in two main classes of vulnerabilities, the first concerns service orchestration, realized in a large extent by the use of BPEL for describing the workflow and BPEL engines for their execution. The second part, presents potential vulnerabilities introduced by the service choreography descriptions, for which the representative language is WS-CDL.

3.1.1 Service Orchestration Vulnerabilities

There is a trend that considers WS-BPEL to be the leading web service composition standard. In [27] one of the demonstrated attacks is called BPEL state deviation. The BPEL engines provides open web service endpoints, which must accept every possible incoming message. An attacker takes profit from this fact to flood the engine with structurally correct messages, but which are not related to any process instance under the control of the BPEL engine. These mes-
sages are discarded by the engine itself, however with a high overhead. A massive service client attack can compromise the availability of the BPEL engine seriously. The resource exhaustion can also be obtained through instance correlation valid messages, which are not expected in a given workflow state, e.g. a receive activity that is not enabled for a certain process instance.

A second attack described in [27] consists in the workflow engine hijacking. This attack relies on the web service-addressing spoofing, which is performed by sending requests to the BPEL engine with false endpoint address, thus leading to a high workload and a DoS. In order the drive the BPEL engine towards a more precise attack, an opponent can redirect attack responses to existing services compromising their availability by flooding them with (valid or invalid) messages.

### 3.1.2 Service Choreography Vulnerabilities

In this section we focus on known security problems involving choreography description of service collaborations. We focus on WS-CDL, for the only reason that it is a candidate recommendation for a W3C standard. Choreography descriptions are not meant to be executable. The main purpose of such descriptions is to provide the clear establishment of a contract among the collaboration participants on the constraints and on the conditions under which messages can be exchanged. From a global view point, the observable behavior of the communicating entities must conform to this specification.

Choreographies are implemented more concretely through the combination of the local, underlying systems involved in the conversations, thus relying on the supporting platforms and programming models of the different hosting environments. Therefore there is no such thing as WS-CDL implementation, or engine, since choreographies are simple descriptions of an agreed protocol, without centralized control.

For this reason, there are currently very few reports about vulnerabilities, bugs, or weaknesses related to WS-CDL or service choreography in general. Several works address the conformance of service compositions to a given collaboration specification [59, 16, 15]. In other words, these works try to provide a formal evidence that a participant violated the contract established in the collaboration description statically.

The work described in [18] presents “misuses” or abuses in service choreographies. Misuses can be understood as classical attacks. In the second category we find illegal business conduction (e.g. Pyramidal schemes, where you have the situation where Bob robs Alice to pay Peter).

In [18] these vulnerabilities are described as follows:

- In **business misuses** an attacker can participate in a choreography with a partner role and deviate from the expected behavior by abusing the choreography model.

  Choreography deviations happen when one or more participants behave in a way not specified in the choreography description. Usually this is done for the misbehaved participant sole benefit in a dishonest manner.

- In **service misuses** defines as the situations where one or more participants exploit design flaws on static choreographic models. In these cases, attackers will try to perform attacks
such as data flow attacks, instantiation flooding, or other kind of DoS attack on the web services.

In order to identify such misuses the work in [18] analysis the message exchanges searching for patterns that indicate either the pyramidal behavior or DoS attack patterns. For that, an evidence generation framework is proposed, where event (evidence) generation are made in a pairwise fashion, under the supervision of a trusted third party. The solution provided is not guaranteed to identify all misuses. It cannot be prevent them either. The technique is based in a heuristic solution, and can be used at runtime, assuming that all services correctly adhere to the evidence generation framework. The feasibility of the attacks is not demonstrated in practice and the evidence generation framework is not implemented.

It is not clear whether the threats (analyzed at runtime) in [18] can be statically detected using some kind of formal verification for checking the conformance of service to a given choreography description, such as in [4, 15]. However this approach seems more promising to counter attacks that may surge in a SOA environment. Suppose for instance that one of the services does not behave accordingly to the choreography description. Such on-line approach may allow for the detection and to take some corrective action at the appropriate time.

### 3.2 Service Security Patterns

The literature suggest a number of approaches for securing SOA’s [10, 20, 19] intended to prevent from a number of general (i.e. not business specific) vulnerabilities. These can be understood as “security patterns”, which are design patterns that provide solution to common security problems. This section surveys these approaches, highlighting their strengths and eventual weaknesses.

Major players in the software industry have published guidelines for assuring SOA security through some security pattern, practice, or process [49, 23]. Not surprisingly, such guidelines are centered on each company’s platform for SOA.

In [19] the PWSSec software process for web-services is proposed. Its main characteristic is to enable the integration of “security stages”, for security requirements, architecture, and technology, in this order. The main contribution of this work is to facilitate the selection of the appropriate web-service security standards addressing a given set of security requirements. Basically, the security requirements can be determined from any input specification (UML use case diagrams, for instance). Attack scenario patterns (selected from a repository) are grouped in attacker profiles and then opposed to the system specification. The next step in the PWSSec process is to define the security architecture, divided in several activities:

- Identification of the security architectural patterns (based on federation environments, including authentication, single sing-on, and domain trust federation);

- Definition of the security policy associated to each pattern (however the paper does not describe clearly how policies are related to patterns);
• Integration of security architectural patterns, which is a methodology for keeping track of the security requirements

• Security architecture validation, to verify whether all attack patterns are covered

• Security architecture specification, which is an activity that produces the written documentation of the choices performed in the previous activities

The process has foundations on previous works [1], adapting them to fit the web service security standards and technologies. However, an experiment for validating the approach against a realistic environment is lacking.

The work in [20] presents a complete approach for ensuring security in SOA’s. The model-driven SECTET process approaches security from an engineering point of view with the goal of automating the generation of security critical services. The methodology starts from the business level functional specification of the system. From the models, the goal is to derive the security configurations for the target architectures. In order to achieve that, the methodology introduces its own domain specific language for defining the SOA workflow (local or global, i.e. distributed, peer-to-peer choreographies). The next step is to create the security objectives themselves according to three basic categories: confidentiality, integrity, and non-repudiation. The DSL meta-model includes a hierarchy of concepts for the corresponding policies for these objectives. After performing a threat and risk analysis, based on CORAS [50], which uses UML diagrams as support for the description, the security policies can be defined.

These policies are then linked to the business process message exchanges, explicitly associating the problem domain and the security requirements in the SECTET methodology. The enforcement relies in the security standards such as XACML (which is well-adapted provided the distributed, peer-to-peer nature of service choreographies). Policy Enforcement Points (PEP’s) are the gateways where basic SOAP processing is performed, including (XML encryption, XML Digital signature). The SECTET also supports more advanced policies such as RBAC and confidentiality policies. The work in [20] also brings an extensive use case in the health care domain where the methodology was successfully applied.

In [32] security patterns are associated to model-driven design in order to generate suitable security configurations relying on the WS-security standard stack. Such automated generation would help avoiding the complex, thus error prone manual configuration. In order to achieve that, models are annotated with labels that indicate the associated security requirements. The annotations on UMLSec [5] diagrams are written in a very simple language for stating security goals, for example identity provisioning, confidentiality, etc. The structure of security patterns is formalized in terms of a meta-model for SOA security. The security configuration is given in terms of a domain specific language based on a relational model (defining the interacting services, the trust relationships, etc) enriched with first-order logic operators (for all, implies, and Boolean operators) on domain specific predicates about security (asserting the security requirements). Such specifications are later compiled, i.e. the patterns are applied to real configurations in a rather straightforward manner given the semantics of the security pattern language described in [32].
Chapter 4

Threat and Risk Analysis Methodology

4.1 Attacker Meta-Model

Over a number of years, researchers in security have proposed a variety of models and formal approaches in terms of attack identification and formally proving the security of these systems such as, Dolev-Yao attacker model [11]. Despite the multitude of proposed attacker models that have hitherto been described in the literature, we take the view that existing attacker models are essentially based on. The essential primitive notions that we identify as common to attacker models are: approaches for categorizing attacks and relationships between them, and a means for specifying a range of modalities. The degree of overlap amongst existing models is significant, and that many novel attacker models can potentially be developed by simply combining. The specific questions of interest to which we are interested is: "Is it possible for a Unifying Attacker Model to be developed given the diversity and types of existent attacker models?". However, our proposed methodology is still work in progress and we are looking for answers to this question. In the following section, we advocate, the categories of attacks that we identify are given a more general interpretation than is usual in attacker models (see Table 4.1).

We consider two classes of attacks: attacks modifying the behavior of the system (Active attacks) and attacks aiming at information retrieval without modifying its behavior (Passive attacks). Passive attacks are frequently a pre-requisite for active attacks; the attacker first analyzes the system in a passive way in order to understand it or recover useful crypto material, and then

<table>
<thead>
<tr>
<th>Active View</th>
<th>Passive View</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A/S</td>
</tr>
<tr>
<td><strong>Physical View</strong></td>
<td>A/S</td>
</tr>
<tr>
<td><strong>Functional View</strong></td>
<td>A/S</td>
</tr>
</tbody>
</table>

Table 4.1: Attacker model for SOA systems
exploits this knowledge in an active attack. We can categorize attacks in two different classes: physical attacks and functional. Each physical attack may indicate which functional attack it can be used for.

4.1.1 Active View

- **Application/Services** Application attacks relative to the functional view directly target flaws in the application logic. An attacker can perform different forms of attacks to compromise a system. Next, the application can be modified by the attacker in order to fulfil his objectives. Denial of service attacks consist of blocking the normal access of the application to authorized users. In this category we can find targeted attacks like a account lockout which is specific to certain a user, or a user category. But these attacks can also be more general, possibly taking a complete domain out of response to shadow a company or some of its departments. Those cases generally involve organized and distributed attacks. Illicit modifications to the application logic performed by attackers are another kind of active attack. The application or service is then unable to perform is task completely, or does it with collateral damages. This situation happens in the case of Cross-Site Scripting for example, where an attacker can force a client to execute an external code - malicious or not, without client’s consent. Thus, the service disrupts its integrity, and divert from its original purpose. A way to take advantage of the service to serve different purposes than original intents is also to use incorrect validation of input and weaknesses in application coding, like buffer overflow attacks. On a physical point of view, a service can be directly replaced by another one which serves other means, modifying the logical behavior.

- **Communication** Communication attacks target communication links. There are two main means to implement attacks against a physical communication link: tampering with it (modifying its topology, jamming it, modifying its main parameters like arbitration policy, frequency, etc.) and injection of forged transactions. Because computing devices and memories are usually connected through buses, attacks against communication links can be used to tamper with the communication or the storage activity. Consequences of communication attacks are on the receiver side only (attacks aiming at modifying or cancelling a message before it is actually sent are in fact attacks against the sending computing node). They comprise the modification of a message between its emission and its reception, the cancellation of a message that will thus never reach its destination or the reception of a message that would never have been received during normal operation. When a memory bus is attacked, it can be to modify the function of a task (software code modification) or the data it processes. There are three classes of memory bus injection attacks: spoofing (the injected information was forged by the attacker), splicing (the injected information was taken at a different location in the memory) and replay (the injected information was taken at the same location in the memory but at a previous moment in time, where it differed from the expected one). The same classes apply to messages. The attacks against communication links are the more powerful of all because, on the functional point of view, these physical attacks can translate in computing, communication, application, or infras-
structure attacks.

- **Computation** Computing attacks are targeting computing nodes (CPUs, hardware accelerators). They consist in physical modifications of the component (like modifying the content of an embedded ROM or the structure of an operator), its replacement or even its destruction. Transient fault injection is another possibility. The consequence is the production of results that differ in some way from those that would have been produced in normal operation, including failure to produce results when expected or the converse. The purpose of a fault attack is very frequently to retrieve an embedded secret, in which case the modification of the behavior is not the actual goal of the attacker but the means. On the functional point of view, these physical attacks can translate into computing, communication, storage or command attacks: modifying the behavior of a task can indeed lead to modifications of the results it produces, of the messages it exchanges with its environment, of the content of the memories it manages or of the orders it sends to actuators.

- **Infrastructure/Resources** Active attacks against Infrastructure/Resources all consist in modifying the regular content of a memory. As a consequence the read operations performed by the tasks accessing the address space do not return the expected information, that is, the last one that was written at the same location. The consequences are very similar to the consequences of attacks against memory buses. The means used to achieve content modification depend on the technology: ROMs can be replaced, non-volatile writable memories (E\(^2\)PROMs, flashes) can be replaced or reprogrammed, volatile memories (static and dynamic RAMs) are much more difficult to attack in a conscious way but more or less random bit flips can be induced by voltage, clock frequency, temperature modifications or more active fault attacks. In some cases, volatile memories can even be cooled, removed from their PCB and plugged on another host without losing their content which can then be read out and / or modified before the component is plugged back in its regular host system.

### 4.1.2 Passive View

- **Application/Services** Passive attacks at the physical view relates to the ability of an attacker to limit the service capabilities, by blocking waves with a scrambler device for example. When it is about functional view, an attacker can take advantage of the service to extract relevant information. It is the case for Injections attacks, like SQL’s one which can allow reading and disclose confidential data to an attacker.

- **Communication** Communication can be spied at and sensitive messages or read/written data exposed. Probing is a very effective and attractive mean for wired communications. Wireless communications are even more sensitive to this kind of attack as they can be conducted in a completely remote and undetectable way. On-chip probing requires package removal, expensive equipment and very skilled attackers.

- **Computation** Passive attacks against the computing activity aim at retrieving either a secret quantum of data (secret key) or the processing definition itself software code extrac-
tion). As every computation is actually performed by a physical device, measurable syndromes are produced, like power consumption, computing time or electromagnetic emissions that can be exploited to guess what operations are performed or what is the value of some sensitive data. This kind of analysis is referred to as side channel attacks in the literature. Observing the external communication or the exchanges with memories is another mean to get information about the computing but fall in the passive communication attacks category.

- **Infrastructure/Resources** Storage passive attack consists in reading the content of a memory. Some very sophisticated analysis tools can be used to investigate memories but they usually imply package removal plus some on-silicon scanning. Memories can also be dumped from their regular I/O. A ROM or a non-volatile memory can be isolated or even removed from its printed circuit board, its address bus driven and its content recorded by a logic analyzer or any similar equipment. In some cases, volatile memories can even be cooled, removed from their PCB and plugged on the recording host without losing their content.

Given an input use case for our methodology, the attacker model helps to both classify identified attacks, but also to think about new ones, given a category. It is a combination of both top-down and bottom-up approach to provide a support tool to security analysts. We will see in following section how the attacker meta-model we defined there can be used in a risk analysis methodology.

### 4.2 Risk Analysis Methodology

In order to identify the most relevant security requirement to be able to prevent or at least detect and contain a threat, we need to assess the level of risk posed by potential attacks. The risk of an attack is seen as a function of the possible severity (i.e. the cost and loss) of the attack for the stakeholders and the estimated probability of occurrence of a successful attack.

This section presents the generic methodology we apply on the use cases to obtain an extensive risk analysis. The approach is mostly inspired by well-known security and risk management processes, widely described in [35], but also in [53, 45]. A brief diagram (cf Figure 4.1) depicts the necessary steps in order to apply this methodology.

#### 4.2.1 Asset Definition

The part of asset definition is in fact larger than just identify assets. To start with, system characterization has to be defined, depending on what kind of software and hardware will be used, as well as what kind of people will be involved in the all process. It helps to identify security objectives that one have to pursue. Therefore, it is necessary to have a global overview of what will be covered and identified in this phase.

In SOA systems, there is different types of attacks, therefore we build an extended attacker model presented in previous section [4.1] The attacker meta-model allows us to define what kind
of categories will be covered by the analysis, like to focus on categories (Active, Functional, *) and (Passive, Functional, *) related-attacks. In this case, we instantiate the model with two instances which contains a set of attacks for a given scenario. The specificity of this model is that it deals with SOA-specific range of attack, that have a wide scope from infrastructure to service attacks, as well as composition-related attacks.

For the vulnerability identification, the goal is to develop a list of system vulnerabilities that could be exploited by the potential users. The definition of vulnerability being ”A flaw or weakness in system security procedures, design, implementation, or internal controls that could be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system’s security policy”.

4.2.2 Attack identification

It is a step which allows the identification of attacks, threats and vulnerabilities for a given system, according to security objectives defined in the previous step. Tools to identify such attacks and threats can be a Threat graph, or Attack tree [52]. It is also possible to start from existing classification, like the one described in next sub-section, or STRIDE which is a classification scheme for characterizing known threats according to the kinds of exploit that are used. In STRIDE, the categories are Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. NIST also proposes to identify threat actions from motivation of source. The source being characterized by the type of users the system can be faced up (Hacker, Insiders, Industrial espionage, etc.).

In SOA systems, there is different types of attacks, therefore we build an extended attacker model presented in previous section 4.1. The attacker meta-model allows us to define what kind of categories will be covered by the analysis, like to focus on (Active, Functional, *) ∧ (Passive, Functional, *) related-attacks. In this case, we instantiate the model with two instances which contains a set of attacks for a given scenario. The specificity of this model is that it deals with SOA-specific range of attack, that have a wide scope from infrastructure to service attacks, as well as composition-related attacks.

For the vulnerability identification, the goal is to develop a list of system vulnerabilities that could be exploited by the potential users. The definition of vulnerability being ”A flaw or
weakness in system security procedures, design, implementation, or internal controls that could be exercised (accidentally triggered or intentionally exploited) and result in a security breach or a violation of the system’s security policy”.

### 4.2.3 Risk Analysis

Once attacks have been classified, it is necessary to estimate the risk that they will occur in a specific system. The complexity required to lead an attack is not the same depending on the threat. We can use a risk matrix to capture and compare potential attacks given *Elapsed Time* or the necessary time to lead an attack, *Expertise* or what kind of speciality is required, *Knowledge of system* or how deep knowledge on the attacked system an attacker need, *Window of Opportunity* at what time an attack can be performed and on which conditions, *Equipment* or what kind of devices and tools attackers need to break into the system.

Also, to understand who might be able to attack the application, we split attacker on different profiles. These profiles have also been developed in different methodologies, like the Risk Assessment in NIST Risk Management report [35] which is illustrated in [53]. The attacker profile, as we discussed in previous sections, is part of identifying a threat, to understand what is its origin. It can be natural - produced by disasters like flooding, fire, etc. or human based - being intentional or not. Also, the intentional property, which differentiate when a threat occurs after an accident behavior or a frontal attack is taken in account. For example, identified profiles are *Hacker*, *Professional criminals*, *Terrorists*, *Internal threats* or even *Accidental Threats*.

The attacker profile helps in the risk assessment to quantify the amount of time and knowledge required to exploit a specific vulnerability. This field is then used in combination with other fields to estimate the likelihood of an attack. Through different scoring systems, like CVSS or DREAD, or risk-level matrix one can rank an attack given identified conditions.
Chapter 5

Use Cases Security Analysis

The goal of this chapter is to provide a detailed analysis of the potential threats concerning the scenarios under consideration for CESSA. From such threat analysis, it will be possible to clearly select the security properties appropriate to mitigate the risks in all levels of corresponding collaborations. The risk mitigation strategy may help to indicate how and where aspects implementing such security properties can be weaved as to eliminate risks.

Section 5.1 presents the security analysis of a loan negotiation scenario, introduced in previous deliverable [12]. Then, in Section 5.2 we present a new scenario, involving embedded systems and its security analysis.

5.1 Loan Negotiation Use-Case Security Analysis

One of the case studies we conduct in CESSA concerns a financial services scenario. The use-case involves multiple partners, whose business process run in different trust domains. We start by recalling the information flow and the role of each partner in the use case.

In the loan negotiation use case, threats can be originated from multiple sources. In Figure 5.1 we show the business process workflow using the BPMN notation. One can easily identify in Figure 5.1 the different entities involved. Firstly, we can mention the real estate agency, which presents properties for sale to the bank client, John. This last is able to apply for loans using a mobile device. It is possible for John to concurrently ask for government aid and to make a loan request to the bank.

The second entity is the bank, represented in the second layer of the Figure 5.1. This layer involves several sub-processes in this business scenario. Some of these sub-processes are carried out by services, and others by human users. From the bank perspective, the goal of its internal processes is to verify that the person asking for the loan has satisfied all requirements given the amount of money and the concerned interest ratings. The assets involved in such kind of transactions are the bank’s reputation and the economical implications if the loan is agreed to an adversary.

The third entity in our scenario is the external credit bureau which is a third party business partner of the financial institution. Its main activities are to processes, store, and safeguard credit
information of individuals and companies. The main assets the credit bureau has to protect the information it maintains. Whenever its database is compromised, the overall process cannot be trusted. Basically, the information provided by the Credit Bureau is used by the other partners to make decisions on whether or not to grant credit, in terms of their own credit granting policies.

Finally, the last entity in this scenario is the government. In this scenario, we assume that policies determining who are the people eligible to government financial aid exist. The asset to protect is again monetary resources: a governmental aid cannot be agreed to a non-eligible citizen.

5.1.1 Overall Description of the Scenario with Potential Attacks

The Figure identifies a set of attacks that can plausibly occur regarding the workflow of the business process. Our intention is to have an exhaustive list of attacks that we will have to refine and consolidate depending the running infrastructure and involved actors. As a matter of ranking, we need to classify attacks with regards to many factors. The methodology that we apply is described in Section and allows us to define the scope of an attack and what kind of resources and time a potential adversary needs to perform it.

In order to cope with a correct threat analysis, we have to come through different steps. The first thing to do is to decide which assets are relevant with regards to the business context in our banking scenario. When we are defining assets, we can focus on information or system assets that allow the different actors to achieve their goals. We present a set of documents that are manipulated by actors and that - once under control of an adversary, can lead to failure in the business purpose of each actor. Assets are then related to the business goal of each of the actors, so we first define what are the goals for these different actors:

- The customer, John, needs to know whether he can be agreed a loan, and under which conditions. Depending on the financial risk of his profile, the loan may have high interest rates.

- The real-estate agency goal is to facilitate the communication between the different actors, and mainly to sell a good in exchange of a commission.

- The government provides financial aid based on specific criteria (income, family composition, etc) that citizens must satisfy. Its goals are to distribute the funds to the people who actually need them. The amount of funding conceded in each case shall never exceed what is defined by the corresponding regulations.

- The business goal of the Credit Bureau is to estimate a risk for a given person and certify these data are accurate.

- Finally, the bank wants to provide a loan at a correct price depending on the customer profile and associated risks.
Figure 5.1: Loan negotiation Negotiation Workflow with Attacks

- Untrusted application launch loan process (malware delivered to mobile device)
- Man in the middle attack (listen, Intercept, alter, inject, replay)
- SQL injection attack
- Exploit vulnerabilities or implementation error
- Insider attack
- Block decision
- Inject bogus information in the system

- Physical Attacks on mobile device
  - Infected mobile devices sending too many message to the system

- Send fake message

- Disable or denial of service
  - Man in the middle attack (listen, Intercept, alter, inject, replay)
Before digging into the various attacks, it is reasonable to categorize assets that are to be protected by the different partners. An asset can be local to a specific domain, or distributed as a result of the collaboration between peers. The table 5.1 lists the different assets we are considering per actor. We can then identify attacks which can lead an adversary to intercept information, disrupt a service, modify data, etc.

<table>
<thead>
<tr>
<th>Actor</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Identity, Personal Data, Mobile Device</td>
</tr>
<tr>
<td>Real Estate Agency</td>
<td>Customer Information, Housing Information</td>
</tr>
<tr>
<td>Bank</td>
<td>Customer File, Government Loan Approval, Internal rating support, Credit Bureau risk report, Bundle solution</td>
</tr>
<tr>
<td>Credit Bureau</td>
<td>Subset of the Customer File, Credit Bureau Risk Report</td>
</tr>
<tr>
<td>Government</td>
<td>Subset of the Customer File, Government Loan Approval</td>
</tr>
</tbody>
</table>

Table 5.1: Assets per actor

In the following, we analyze two scenarios derived from our financial use-case.

### 5.1.2 Sub Scenario 1 - External Audit

On a regular basis, the Bank employs an external audit company to perform a battery of tests to ensure that all processes comply with the different regulations. Security checks through code scan analysis and manual review of the functional logic of the different applications are included in such tests. While auditing the loan origination process, the security experts apply the methodology from Chapter 4 and come up with different threats. The report shows that an adversary can interfere with the system to introduce corrupt data and to fake identity, while exploiting different vulnerabilities at different level of abstraction. The adversary can attack the application by targeting different web services at specific time. One can then take advantage of the overall orchestration of services, which is similar to a race condition attack.

What the security experts have identified is that the government relies on information given by the customer, as well as the bank. And nor the government or the bank verify that the customer express the same context to both peers. The customer can then ask for government financial aid for a good that may not be the same as the one from which the customer ask for a loan. When the bank then asks if the government supports the given loan, only the customer information is transmitted, and the government certifies it (cf Figure 5.2).

The external company followed the methodology in 4.2 to have a global view and to shrink the analysis to specific categories focusing on all active attacks that target the application/service and the communication categories. A partial attack tree is shown in Figure 5.3 to reflect how a customer might be able to take advantage of the bank process for its own interest. Several cases are then identified. At the first level, we identify three different ways to turn the process in favor of the customer: (i) having an higher loan than allowed, (ii) Having a better rate interest than customer risk profile might have and (iii) having a greater aid amount from the government.
5.1.3 Sub Scenario 2 - Regulation Requirement Change Process

The processes established by the bank follow regulations coming from many national and international regulatory bodies, such as SOX\(^1\) or BASEL\(^2\). Since the global business landscape is in constant movement, these regulations are subject to change. For instance, in our financial scenario, the bank must have to modify its processes as quickly as possible because of the banking crisis or to cope with new corporate rules. Such changes should not interfere with the behavior of the applications already in place, unless this behavior is directly concerned with the new regulations.

Modifications are not likely to be localized in the application code, but rather scattered. Let us assume the new regulation extends transactions tracking during collaborations (logging system) \([36]\). This impacts not only the loan negotiation process, but also requires stringent security requirements (i.e., integrity) on the overall system. This evolution has a major impact in the loan negotiation process as it involves modifications at different layers.

At the collaboration level, a logging system has to be implemented in each partner system, modifying the horizontal composition. The Figure 5.4 shows an example of where monitoring

---

\(^2\) [http://bis.org/publ/bcbsca.htm](http://bis.org/publ/bcbsca.htm)
and logging systems should appear across administrative domains. Different security mechanisms have to be coordinated together with the execution of the application in order to ensure communication security. For instance, digital signatures provide the best way to ensure the integrity of a communication protocol. They should be qualified according to legal requirements in order to make it possible to integrate service compositions into legal frameworks. The Process level has to process the tasks whenever messages are exchanged between peers. According to the new regulations, the system has to log the state of its operation. For instance, reporting all critical cryptographic operations. At this level, platform integrity can be enforced by using TPM (Trusted Platform Module) solutions which provides the basis for a larger set of high-level security functions (e.g., platform integrity attestation). Finally, the Resources level is impacted with the new security requirement (e.g., data integrity) which can be ensured by hashing the log files then, storing $hash + log$ file in a secure storage device.

This example explains how a small modification in a regulation can have a huge impact with regards to the application, at both vertical and horizontal composition. With our approach, the goal is to reduce the amount of development and modifications by expressing a cross-cutting
property like the logging, and to have a description of actions to perform at different levels. This way, we can modularize different properties and decide to apply them to different targets whenever needed dynamically. Such a modularization of concerns for SOA’s is even more welcome when it involves heterogeneous systems.

5.2 Home Automation Use-Case Security Analysis

This second use case analysis focused on embedded systems. The context of this case will be home automation for companies.

5.2.1 Overview

The company PF4U is located on the twenty-first floor of a building located in the center of Brussels. The company has a highly confidential activity. Therefore, they need to install a monitoring building system. Since the 80’s the Belgium’s law enforces this kind of company to
have a CCTV (Closed-Circuit TV) system to record what is happening during the opening hours, and an alarm to lock the building down in case of emergency.

This year, the company PF4U wants to change its CCTV system. They also want a global solution, to combine different aspects of a smart building.

The security solution is composed of one CCTV camera to monitor the entrance, and several presence sensors in each place where burglars can break and enter. The system offers the capacity to switch lights at distance thanks to step relays. The system must offer the capacity to manage everything and to operate from a remote access. To offer this kind of services, a central platform is needed. It also requires some extension capacity, such as deploying a new device.

The HAP should also support a traditional Web application for the use and the administration of the whole system by users of the PF4U company. Moreover in some cases the HAP can be use remotely via the Internet.

The HAP keep logs of the images coming from the CC-TV in an external server which provides much more disk space than the embedded device, it keeps also logs from the motion sensors. These are the only logs that are kept.

The central platform is called Home Automation Platform (HAP) and will be an embedded but powerful device based on a ARM9 \[3\] or a ARM11 \[2\] processor. Sensors are really light embedded systems that can interrogate the platform, holds the OSGi ME bundle that understands their behavior and exposes their Web Services. For each added device the HAP has to dynamically detect and deploy the new device by installing the OSGi ME bundle specific to the sensor. This module can expose Web Services to the internal network and internal services to the other bundles of the HAP. These services may be used by other sensors/bundles or by a user. To sum up, the service oriented architecture is driven by the use of OSGi ME, indeed each bundle is viewed as a group of services (Web service or other type of services).

The following schema can sum up the architecture of the system:
5.2.2 Potential Attacks

5.2.2.1 Hardware Attacks

This section focuses on attacks specific to the embedded device and involving modules and services isolation (i.e., calls and code isolation). More common attacks (typically DoS, man in the middle, privileges escalation, ...) are not described in this section.

- **Theft**: The HAP itself or the individual sensors can be stolen in order to get back either the technology that runs the system or sensitive information about the society.

- **Corrupted devices/sensors**: A corrupted device can be inserted in the system in order to take control or destroy it.

5.2.2.2 Software Attacks

- **Remote attacks through the Internet**: An attack can be triggered by the remote access through the Internet by cracking the remote access module. The HAP system should be protected against misuse of the remote control bundle.

- **Internal user attacks**: The attack can come from an insider. It means the user rights management should protect the system from being cracked. It requires strong authentication module.

- **Malicious devices/sensor attacks**: The sensor devices can be valid in terms of hardware and software but is malicious by communicating information to the outside of the company (for example: a GSM communication).

5.2.3 Use cases

5.2.3.1 General Use-case

The different actors of use cases are:

- **The user of the HAP**: The user of the HAP will only consult public services, for example switching lights.

- **The administrator of the HAP**: This user can install services into the HAP, modify rules of services interaction, and access every services.

- **The janitor**: At night the janitor can access the same services as the user in addition to the CC-TV system to monitor activities inside the building.

- **The government**: The government sets the laws that govern what can be done with the CC-TV and the HAP. For example logs of the CC-TV cannot be kept more than 10 days. Laws are likely to change during the time of use of the HAP.
The main scenario focuses on a burglary inside the PF4U company and the evidences that have been collected by the CC-TV system and how it would be provided to the government (the police). Obviously these evidences should be protected against every attacks the HAP can get, especially from the burglars that would like to erase evidences during their robbery. The complaint and evidences can be automatically sent to the police department with a new secure test process the PF4U company is engaged in.
Figure 5.6: Home Automation Burglary Workflow
In this chapter we will discuss about the assets every actor holds, here is the table describing these assets:

<table>
<thead>
<tr>
<th>Actor</th>
<th>Assets</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Identity, Personal Data</td>
</tr>
<tr>
<td>Administrator</td>
<td>Identity, Personal Data, Special credential, Users information, CC-TV images, HAP sensor logs</td>
</tr>
<tr>
<td>Janitor</td>
<td>Identity, Personal Data, Special credential, CC-TV images</td>
</tr>
<tr>
<td>Government</td>
<td>Evidences, company and users information, part of CC-TV images, part of HAP sensor logs</td>
</tr>
</tbody>
</table>

Table 5.2: Assets per actor for PF4U

Following are two sub-scenarios of use of the HAP and mechanisms that are taken into account to secure installation, communication, and real use of the whole system. It will based on the HAP architecture described in the 5.2.1 section. It mimics the test cases provided in 5.1.2 and 5.1.3 by describing an installation of new type of camera and a change in the regulation of the use of a CC-TV system.

5.2.3.2 Sub-Scenario 1: the CC-TV system change

This scenario introduces the installation of a new type of camera into the HAP. The black and white old cameras are replaced by colors cameras with a better refresh rate. The introduction of a new type of camera should not change the business logic of the platform. Moreover, a study should be done on the reliability of the new device and its interaction with the HAP. The new type of camera can lead to the introduction of malwares and potential security threats that should be taken into account. The OSGi ME module that will introduce the new services to the platform should be properly identified.

5.2.3.3 Sub-Scenario 2: the Regulation Test-Case

The regulation change test case involves a change in the law that forces the platform to keep logs of users and administrators actions during 6 months in order to gather proof that the CC-TV was not misused or supposing it was misuse the logs allow to keep a forensics trail. The impacts of this regulation change on the HAP system are the following:

- **Process**: The change of regulation should not change the business logic of other part of the system. A new service that matches the regulation need has to be introduced without impacting the already installed services.

- **Resources**: The new data that has to stored should be well encrypted and stored in trusted area.
Chapter 6

Conclusion

In this deliverable we have presented different security constraints an SOA-based system imply. We identified security relevant issues when we are in presence of Web-Services Infrastructure, also called service isolation to reflect risk inherent to isolated services and their implementations. We then investigated on security vulnerabilities related to web-service compositions at orchestration and choreography level.

Whereas we identified potential attacks and vulnerabilities on services, we presented in Chapter 4 well known methodologies that are used to identify and elicit risks with regards to use cases. On this chapter, we propose an attacker model we developed to cover SOA’s specific. It can be view as a tool to support classification of attacks in specific categories.

Finally, our core contribution for this deliverable is a security analysis for the main use cases of interest for CESSA. We have elicited new security requirements through the use of a the methodology for vulnerability analysis. The effectiveness of this methodology shall be demonstrated in future works to validate our proposition.

The work on this deliverables in combination with previous one were to gather and develop use cases to highlight CESSA interest for composition and evolution of services, in presence of security. We brought a global view of the existing work and defined what is our position. As the aspect model and service model that we defined in [12] still evolves, we are now liven up these concepts. The next phases of the project are then to propose a formal definition for service composition and aspect composition as well as develop a language to correctly describe and certify secure SOAs in presence of aspects. We will study how distributed aspects can be successfully employed in order to eliminate the risks identified in the current deliverable for each sub-scenario we have described here.
Bibliography


[40] Oracle. Java GSS. http://download.oracle.com/javase/6/docs/technote
guides/security/jgss/tutorials/index.html


docs/technote/guides/security/jsse/JSSERefGuide.html


