



Norwegian University of
Science and Technology

Security in SOA-Based Healthcare Systems

Richard Sassoon

Master in Security and Mobile Computing

Submission date: June 2009

Supervisor: Danilo Gligoroski, ITEM

Co-supervisor: Jostein Jensen, SINTEF
Peeter Laud, Tartu University

Norwegian University of Science and Technology
Department of Telematics

Problem Description

National ICT is an institution coordinating ICT initiatives in the Norwegian specialised health services, and to achieve a common platform for those health services they suggest using service oriented architectures (SOA). Personal privacy in eHealth applications is protected by law, thus security is a crucial quality attribute for healthcare platforms. However, as pointed out by Epstein et al., implementing security in SOA is not trivial.

A European research project called MPOWER has already developed a home care service platform, including security functionality, based on SOA. In this thesis, the student will use the MPOWER platform as test case to act in the role as hacker, and perform a penetration test on the system. Solutions to increase the security of the platform should be suggested and possibly implemented to improve user privacy. A discussion related to information security aspects of SOA Web Services as building blocks for healthcare platforms is expected, where results from the penetration test and vulnerability elimination can be used to exemplify theories and claims.

Assignment given: 15. January 2009

Supervisor: Danilo Gligoroski, ITEM

Abstract

Healthcare organizations need to handle many kinds of information and integrate different support systems, which may be accessed from external corporations. Service Oriented Architecture (SOA) provides the means to achieve a common platform to deploy services that can be used across the organization and its boundaries, but introduces new security concerns that need to be evaluated in order to implement a secure system, while still suffering from standard threats. Web Services are the common way to implement SOA applications, having several standards related to security (such as XML Encryption, XML Signature and WS-Security). Still, other security mechanisms such as input validation and SSL/TLS need to be thought of as well.

A penetration test based on recognized methodologies and guidelines, such as the NIST Technical Guide to Information Security Testing and Assessment, OWASP Testing Guide and SIFT Web Services Security Testing Framework, was performed on a case study system. A proof of concept application making use of a set of middleware (web) services, the MPOWER platform, was audited in order to expose vulnerabilities.

After conducting the penetration test on the system, 10 out of 15 scenarios presented security issues. The vulnerabilities found were described, demonstrating several risks from misusing, or not implementing at all, security mechanisms. As a consequence, countermeasures and recommendations were proposed in an attempt to improve the overall security of SOA-based (healthcare) systems.

The results of the assessment show us how important is to validate the security of a system before putting it into production environment. We also come to the conclusion that security testing should be an inherent part of a secure software development life cycle. Moreover, not only healthcare systems may benefit from this study, and also not only SOA-based ones.

Preface

This thesis concludes my Master of Science degree at the Norwegian University of Science and Technology (NTNU), with a project carried out within SINTEF.

I would like to thank my supervisor at SINTEF, Jostein Jensen, for his valuable input throughout the development of this thesis. His knowledge, comments and support were really important during the whole process. I also would like to thank my co-supervisor at the University of Tartu, Peeter Laud, for his feedback during the project. In addition, I wish to thank professor Danilo for the opportunity to work with this thesis.

I wish to give my special thanks to Mona Nordaune for her assistance during the NordSecMob program. Always helpful and thoughtful person.

I also would like to thank my fellow students with whom I shared great experiences and had interesting talks and support during these two years of master's.

I am very grateful to my parents Jairo and Jacqueline who motivated me along all my studies, and to my girlfriend Pia who helped and supported me since the beginning.

Trondheim, June 2009

Richard Sassoon

Contents

List of Figures	v
List of Tables	vi
Abbreviations	vii
1 Introduction	1
1.1 SOA, Healthcare and Security	1
1.2 Research Goals	2
1.3 Limitations	2
1.4 Chapters' Outline	3
2 Background Information	5
2.1 Important Definitions	5
2.1.1 Encryption	5
2.1.2 Hash Functions	6
2.1.3 Digital Signatures	6
2.1.4 Public Key Infrastructure - PKI	7
2.2 SOA and Web Services	7
2.2.1 SOA	8
2.2.1.1 Security Challenges	9
2.2.2 Web Services	13
2.2.2.1 Basic Standards	14
2.2.2.2 Security Related Standards	18
2.3 MPOWER Platform	27
2.3.1 About the Project	27
2.3.1.1 Architecture	29
2.3.2 Security Requirements	30
2.3.2.1 Security Design	32
3 Methodologies and Guidelines for Security Assessment	34

CONTENTS

3.1	Benefits of Using a Methodology	35
3.2	Open Source Security Testing Methodology Manual - OSSTMM	36
3.3	Technical Guide to Information Security Testing and Assessment - NIST-SP800-115	40
3.4	OWASP Testing Guide	43
3.5	SIFT Web Services Security Testing Framework	45
3.6	Chosen Method	46
4	Preparations	49
4.1	Testing Environment	49
4.2	Tools	50
4.3	Test Cases	51
5	Assessment Evaluation	66
5.1	Problems	66
5.2	Roles and Interfaces	67
5.3	Results and Countermeasures	69
6	Recommendations	85
6.1	Summary of Risks and Security Review	85
6.2	Configuration Aspects	87
6.3	Distribution Model	88
6.4	Final Comments	89
7	Discussion	90
8	Conclusions and Further Work	93
8.1	Conclusions	93
8.2	Further Work	94
	Bibliography	95
	Appendices	105
A	Fuzzing	105
A.1	Attack Vectors' File	105
A.2	SOAP Template	106
A.3	Script for Generating SOAP Requests	106
A.4	Script for Sending SOAP Requests	107

List of Figures

2.1	Digital signature process	7
2.2	Consumer-Producer interaction	9
2.3	Traditional client/server approach to security	10
2.4	A possible SOA setting, with services being called from different applications/organizations boundaries	10
2.5	The mechanics of the SSO concept	22
2.6	The MPOWER framework	29
2.7	MPOWER reference architecture	30
2.8	MPOWER security components	32
4.1	Expected testing environment	50
5.1	Doctor's homepage interface (before a patient is selected)	67
5.2	Doctor's interface after a patient is selected	67
5.3	Patient's interface	68
5.4	Doctor's interface - send message	68
5.5	The soapUI tool	69
5.6	Doctor as a victim of XSS	77
5.7	Session does not expire	83
6.1	MPOWER distribution model	88

List of Tables

2.1	Security requirements and the associated standards	18
2.2	Security requirements and the associated laws and regulations	31
2.3	MPOWER security components	33
3.1	A summary of methodologies, indicating how they are related to our chosen method	48
5.1	Use of salted hash on passwords	80
6.1	Summary of risks	85

Abbreviations

ACL Access Control List

CEN/TC 251 European Committee for Standardization of Health Informatics

DOM Document Object Model

DoS Denial-of-Service

DTD Document Type Definition

ebXML e-business XM

FTP File Transfer Protocol

HL7 Health Level Seven

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated Development Environment

IDS Intrusion detection system

IETF Internet Engineering Task Force

MPOWER Middleware Platform for eMPOWERing cognitive disabled and elderly

NIST National Institute of Standards and Technology

OASIS Organization for the Advancement of Structured Information Standards

OMG Object Management Group

OSSTMM Open Source Security Testing Methodology Manual

ABBREVIATIONS

OWASP	Open Web Application Security Project
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
POCA	Proof of Concept Application
RBAC	Role-Based Access Control
REST	Representational state transfer
SAML	Security Assertion Markup Language
SDLC	Software Development Life Cycle
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SQL	Structured Query Language
SSL	Secure Sockets Layer
SSO	Single Sign-On
STS	Security Token Service
TLS	Transport Layer Security
UDDI	Universal Description, Discovery, and Integration
W3C	World Wide Web Consortium
WS-I	Web Services Interoperability Organization
WSDL	Web Services Description Language
WSS	Web Services Security
XAMCL	eXtensible Access Control Markup Language
XDoS	XML Denial-of-Service
XKMS	XML Key Management Specification
XML	Extensible Markup Language

Chapter 1

Introduction

The purpose of this chapter is to give some insight on SOA and the problem to be solved, what gives us the motivation for this thesis. The goals and scope are presented as well as an outline of the chapters.

1.1 SOA, Healthcare and Security

Information systems are part of every organization nowadays. They are needed for the most varied purposes such as Intranet, accounting, business intelligence, etc.

Since Service Oriented Architecture (SOA) was introduced to the business scenario, it has been adopted by more than fifty percent of companies worldwide for their IT systems and architecture, particularly in Europe and North America, according to independent reports by Computer Economics [1] and Gartner [2]. SOA provides the means for integrating people, processes and information in an organization, by integrating the company's different applications. This is accomplished by using a set of technologically independent linked services that can be accessed over a network [3]. One of the main focus is the reuse of these services, in order to avoid redundancy and thus reduce development time and costs, while maintaining consistency in the company's processes.

This also holds for healthcare organizations that have to deal with many different kinds of information and support systems and need some way to integrate them. Often these systems need to be contacted from outside the organization and interoperability issues arise. By using a service oriented approach, it is possible to standardize data management and use different system capabilities, as services, across the organization's units and beyond [4].

The National ICT is a Norwegian institution coordinating ICT initiatives in specialized health services, and they recommend SOA as a mean to achieve a common platform for those services [5].

The use of SOA introduces some security concerns, as pointed out by Epstein et al. [6] and the New Rowley Group [7], that are of special interest for the healthcare sphere, one of the main ones being data handling of patients, since services can be accessed inside and outside of the organization. One would expect that this data should only be accessed by personnel with the right authorization and should not be tampered with. Besides that, each country has its own laws and regulations that influence the security requirements for processing personal data, mainly concerned with availability, confidentiality and integrity of the data. The European Union Directive 95/46/EC [8] regulates these issues with private data for the member states of EU, giving some freedom for them to define more precisely how to transpose this directive into their internal law.

We see that information security has already an important role in healthcare systems [9] in order for them to be trusted to deal with sensitive data, and the above mentioned factors make us think how SOA can be beneficial to current and future healthcare systems, as long as the security aspects are verified thoroughly to make sure these systems are compliant with the regulations.

1.2 Research Goals

The purpose of this thesis is to identify security risks that affect SOA-based healthcare systems, more specifically Web Services based ones, and propose solutions to mitigate these risks. As a case study, a proof of concept application (POCA) making use of a set of middleware (web) services, the MPOWER platform [10], will be assessed in order to identify vulnerabilities.

Through the use of different security testing techniques, the system will be audited, to verify the compliance to the security requirements, and the vulnerabilities exposed. In the end, a detailed report reflecting the actual findings will be generated and used as guidelines and future improvement of such systems. The results will be relevant for supporting the secure development of applications based on such a system architecture in the health domain, where data privacy is a big concern.

1.3 Limitations

The scope of the tests has to be narrowed down, since there is a great number of attack vectors. We need to select the most relevant ones to be able to perform the assessment within the given time frame. There will be no tests

concerning operating system's specific flaws, wireless transmissions, physical security¹ or social engineering². The focus will be on the MPOWER services, along with the application making use of them, and the security they provide.

To be able to implement SOA, there is a need for an infrastructure that allows the use of the services. This is the enterprise service bus (ESB) [12]. We are not concerned with peculiarities of the ESB.

SOA can be realized through different technologies. Web Services is the most adopted one, and as such it will be the focus when dealing with SOA security in this thesis.

Due to time constraints, the proposed solutions to the issues found will not be implemented, and their effectiveness will not be verified in practice.

1.4 Chapters' Outline

This thesis is organized in eight chapters. The following list contains a short description of each one:

- **Chapter 1 - Introduction**

The first chapter presents the motivation for this study, along with the desired goals and the self imposed limitations to conduct our work.

- **Chapter 2 - Background**

In this chapter, we present the relevant background information to support our study. Important definitions are given, including SOA and Web Services. Common security challenges are seen in a SOA perspective, together with some possible ways to deal with them. Some web services standards are described, the most important for us being the security related ones. Finally, we have an overview of the MPOWER platform, which will serve as a case study to reach our goals.

- **Chapter 3 - Methodologies and Guidelines for Security Assessment**

Here we review some methodologies and testing frameworks for performing security tests, and discuss what are the benefits for using a standardized approach. At the end, we present the method that will be used to conduct the tests.

¹Controlled physical access to systems.

²Social engineering is an attempt to trick someone into revealing information such as credit card numbers or passwords [11].

- **Chapter 4 - Preparations** This chapter presents how the testing environment is set up and also the tools that will be used to perform the required analysis. A list of test cases is introduced, each dealing with different threats and security aspects.
- **Chapter 5 - Assessment Evaluation**
Here we describe the problems faced to carry out the assessment, relevant information to understand some test cases, and the results obtained, together with countermeasures.
- **Chapter 6 - Recommendations**
This chapter proposes recommendations, based on the test results, to improve the overall level of security of the MPOWER platform, which may be useful for other SOA-based (healthcare) systems as well.
- **Chapter 7 - Discussion**
This chapter has the objective to reflect about some aspects of the work conducted in this thesis.
- **Chapter 8 - Conclusions and Further Work**
In this chapter we summarize the achievements of our work and discuss further possibilities related to this project.

Chapter 2

Background Information

This chapter has the objective to present background information on some important definitions, SOA, Web Services and the security challenges introduced by using a SOA-based implementation. Secondly, the MPOWER platform is described along with its security aspects.

2.1 Important Definitions

To better understand this thesis it is relevant to introduce some concepts that will be mentioned in later sections. These definitions are not comprehensive and, if needed, the reader should look for the appropriate mentioned references.

2.1.1 Encryption

Encryption, in cryptography¹, is a mechanism/process that uses specific algorithms to transform some plaintext into a ciphertext², through the use of a key, to achieve confidentiality [14]. The reverse process is called decryption. There are two types:

- Symmetric: Encryption and decryption are performed using the same key. One difficult problem with this approach is the key distribution, since both parties, involved in some communication, need to know the key.

¹Cryptography, as defined by the Internet Security Glossary [13], is *"the mathematical science that deals with transforming data to render its meaning unintelligible ..., prevent its undetected alteration, or prevent its unauthorized use"*.

²Usually, the plaintext input is cleartext, but in some cases it can be ciphertext resulting from another encryption, which is known as super encryption[13].

- **Public key:** Also known as asymmetric, this scheme uses a public key for encryption and a private key for decryption. This key-pair is related in a way that is easy to perform encryption and decryption operations but computationally infeasible to derive the private key from the public key and the encryption algorithm. It is also infeasible to recover the plaintext from the public key and the ciphertext.

With public key it is possible to perform a session key exchange in a secure way so symmetric encryption can be used, since public key encryption is computationally expensive [15]. It is also possible to perform one's authentication, by means of digital signatures.

Key distribution is also of importance in this case. Public keys are supposed to be public, so there should be a way to announce them. The most interesting for us is the use of certificates, that are issued by a trusted third party (certificate authority), after receiving a user's public key. The certificate contains the public key, is signed by the authority and can be used to verify the user's identity. X.509³ is a widely accepted standard for formatting these certificates.

2.1.2 Hash Functions

A hash function H produces a fingerprint of a message M . It receives a block of data and outputs a fixed-length digest h through a one-way mathematical function [16]. $H(M)$ should be easy to compute but, given h , it is computationally infeasible to find M such that $H(M) = h$ [14]. A hash function is a useful mechanism for verifying the integrity of the message, because any change in it would produce a completely different hash value, what is known as the avalanche effect.

2.1.3 Digital Signatures

Resulting from the public key encryption scheme, digital signature is a mechanism that provides sender's authentication and message integrity [14]. To create a signature, the hash of the message is generated and encrypted with the sender's private key. To verify the signature, the recipient generate the hash of the message, decrypt the attached signature with the sender's public key and compare the hash values. This process guarantees the identity of the sender as well as the integrity of the message. Figure 2.1 shows the necessary steps.

³<http://tools.ietf.org/html/rfc4158>

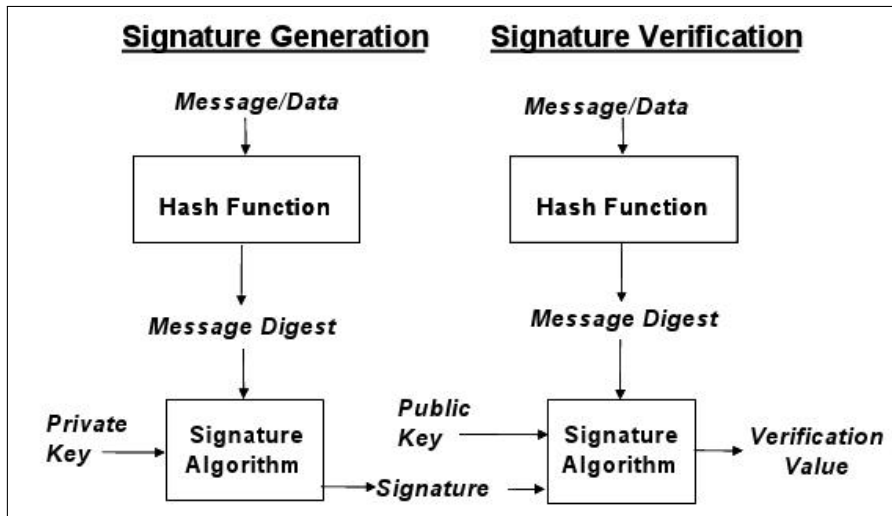


Figure 2.1: Digital signature process. Figure extracted from NIST SP 800-21 [17].

2.1.4 Public Key Infrastructure - PKI

The Internet Security Glossary [13] defines Public Key Infrastructure (PKI) as *"the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography"*. A Certificate Authority (CA) is responsible for authenticating an entity, such as a person, organization, account, or site, and generating a certificate binding its identification information to its public key [18]. This method makes it possible to acquire public keys efficiently and securely. The Public Key Infrastructure X.509 (PKIX) is the model based on X.509 certificates and is the one of interest here.

2.2 SOA and Web Services

There is still people who think that SOA and Web Services are synonymous and that the first needs the second to be implemented, but that is just a misconception. To make it clear we say that SOA is a design principle, or paradigm, and Web Services is an implementation technology [12, 19]. This confusion happens because Web Services are considered as the de-facto standard for realizing SOA, but there are other possibilities⁴.

The next sections will present some basic definitions of both concepts in order to understand the security threats that are related to each one. Detailed

⁴See, for example, http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci1261398,00.html.

information can be found in the relevant bibliography at the *References* section.

2.2.1 SOA

There are many definitions of what SOA means, and IBM states [19]:

"SOA is an approach to build distributed systems that deliver application functionality as services to end-user applications or to build other services"

It is possible to see it through an ecosystem perspective [20] or as a business community [12]. In the end it comes to the main concept of producing reusable components, each of them encapsulating some kind of logic. The size and scope of this logic vary according to the context being considered. The service can provide some authentication mechanism or a complete sales environment, for example. The latter service could use the functionality of the former.

Important definitions to help understand the relationship between services are the following [21]:

- Service Consumer (Requestor): *"An entity which seeks to satisfy a particular need through the use [of] capabilities offered by means of a service"*.
- Service Producer (Provider): *"An entity (person or organization) that offers the use of capabilities by means of a service"*.

Business processes are comprised of several steps that are executed according to predefined sequences and rules. A service can be responsible for an individual step or a set of steps. As said before, services can be used by other services, what it is seen as a service composition or service chaining [22], and in order to have such an interaction they have to know about each other. This is made possible due to the use of service descriptions.

The service description represents the information needed in order to use, or consider using, a service [21]. The purpose of the description is to facilitate interaction and visibility. While this makes available information on service reachability, functionality, interface and policies, it does not include technical details of the implementation, what gives us the concept of *loose coupling* between services. This allows software on each side of the communication to change with no impact on the other, thus reducing dependencies between different systems [12].

The service provider publishes the description of its service in some *Service Registry*. This registry is looked up by a service requestor in order to obtain the necessary information to interact with a service.

Comparing to the traditional client/server model we can say that a service provider behaves as a server and a service requestor as a client. Figure 2.2 shows the relation between provider, requestor and registry:

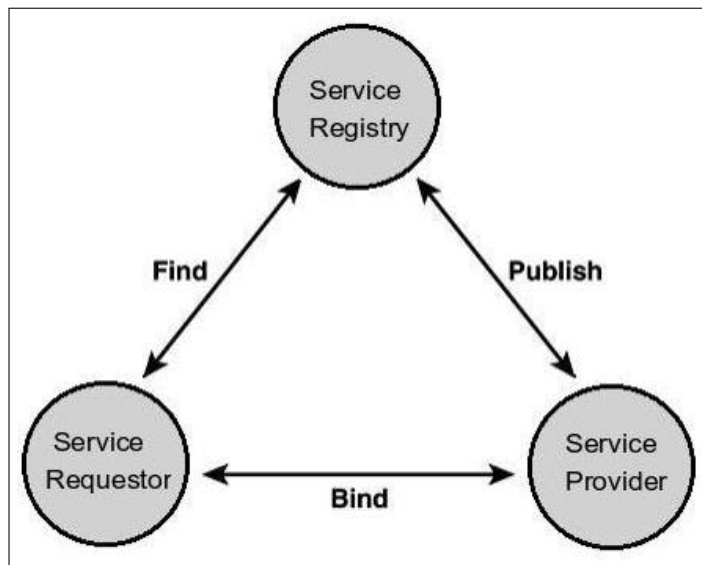


Figure 2.2: Consumer-Producer interaction. Figure adapted from Graham et al. [23]

2.2.1.1 Security Challenges

In sections 1.1 and 2.2.1 we saw the motivation and the basic principles of a SOA-based system, what makes it easier to understand this section. The way SOA is structured introduces new challenges regarding security that have to be addressed by non standard means [24]. The main idea is that services are meant to be discovered by other services, including from different applications and even organizations, and to inter-operate regardless of technology, blurring the concept of boundaries.

In traditional approaches, there is not much doubt about the security model to adopt. Figure 2.3 shows how a server application provides its functionalities to clients, via secure channels, and how it relies on one security module responsible for all security decisions, such as authentication, authorization and firewall policies.

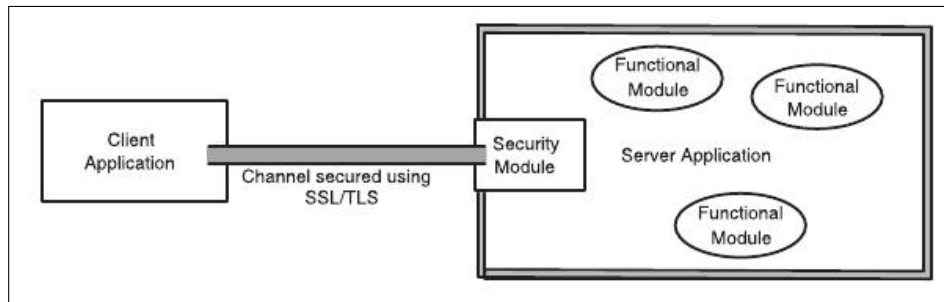


Figure 2.3: Traditional client/server approach to security. Figure extracted from Kanneganti et al. [24]

Applications in a SOA context can be exemplified by the figure 2.4, where we see "client application 4" requesting service "p2", that in turn uses the service "2b". These services are made available by different organizations and thus there is a need to impose limits on what the corresponding applications can see and manipulate, since there could be sensitive information meant to be read only by the last service/application in the chain.

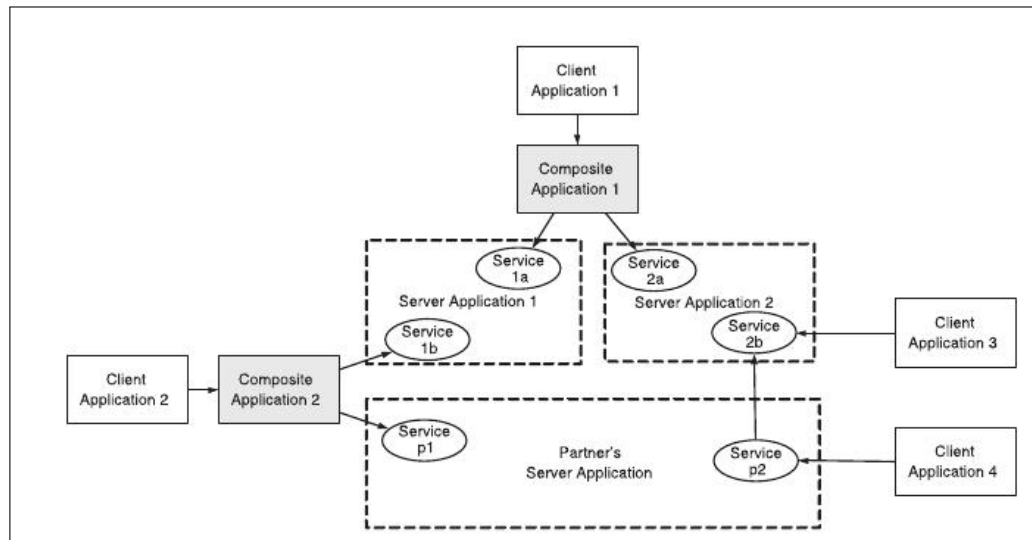


Figure 2.4: A possible SOA setting, with services being called from different applications/organizations boundaries. Figure extracted from Kanneganti et al. [24]

The key security requirements are the same as for traditional applications, but they are tackled in different ways [24]. The following list presents these requirements and some alternatives to deal with them in SOA:

- **Authentication:** Is the act of confirming one's legitimate identity to a system. In a SOA context, services can be invoked in many ways, affecting the usual authentication procedure. When called directly by a client application, the user can be authenticated via a standard username/-password approach against the application's user database/directory. If the call comes from another service within the same organization, the requested service can trust the previous authentication or perform a re-authentication. In the last case, where the service requested is invoked by another organization's application, relying on this other application's authentication could be a solution.

The issue here is that the SOA model, where a user requests services across different trust domains (organizations), needs a common authentication process, in order to attain a seamless interaction with the system. An identity that is verified in a trust domain and wants to prove itself and its rights in another trust domain, is called portable [25]. In section 2.2.2.2 we will see how SAML addresses these portable identities.

Some standard approaches include the use of *Kerberos*⁵, a network authentication protocol, and a PKI to provide authentication across services [24].

- **Authorization:** Determines if an authenticated user has the rights to access a requested functionality. One of the most common access control models is the *Role Based Access Control (RBAC)* [24], where roles are granted permission to perform actions on resources. Users can be assigned roles in order to interact with an application. Access control rules are usually defined for each service, what implies that a composite application should be able to verify the rules for each of its services in order to perform some action.

A central service that manages users and their associated roles is called an identity provider/manager [12] and is a way to provide authorization across the services of a composite application, via a security token that contains an unified user credential [26], also referred as *single sign-on (SSO)* method. The use of a security token, together with a username/password or PKI certificates [22], for example, can also address the authentication requirement outlined above, via SAML. Important to note that the integrity of this token should be verified before authorization decisions are performed.

⁵<http://web.mit.edu/Kerberos/>

- **Confidentiality:** Information should be only visible to parties with the authorization to do so. When data is exchanged over a network, it is necessary to provide a mechanism to protect this data from eavesdroppers. The usual way to provide data confidentiality is by using encryption. The traditional approach is to establish secure communication channels by means of *Secure Sockets Layer (SSL)/Transport Layer Security (TLS)* [27]. While this ensures data confidentiality between two points (point-to-point encryption), it does not prevent an intermediary service, from some other organization for example, from reading/manipulating data that is meant to be used by another service. One solution to this problem is using *message-layer security* as an end-to-end encryption solution⁶ [12], where only the contents of the message are encrypted, so the information regarding routing can be processed separately. For this to be possible, all endpoints need to be able to communicate with each other securely, and to achieve this a PKI can be employed [22].
- **Data integrity and non-repudiation:** A receiver has to be sure that the message received is the same as the one sent by the sender, and not something else, that could have been modified during the transmission. Applications are responsible for verifying the integrity of data received over a network. On top of that, a sender cannot repudiate/deny that it sent a message, as this could be the cause of disputes if, for example, this sender commits to perform some contractual job and later says that there was never such an agreement, i.e., a unilateral modification should not be possible. Similarly, the receiver should be able to verify that the message was indeed sent by the claimed sender.

Here SSL/TLS also plays an important part, by verifying data integrity and ensuring non-repudiation, but suffer from the same problem of not being end-to-end. In this case, the use of digital signatures is a way to solve this issue, as long as all endpoints have the right certificates to verify the signatures of received messages, once again using PKI [24].

- **Protection against attacks:** Applications available over the Internet need to be protected against attacks that target common vulnerabilities such as insecure code (not performing input validation), poor administrative practices (weak passwords), and operating systems and networking infrastructure (TCP/IP implementation, for example), which have their own specific flaws. The usual defense strategy makes use of firewalls to protect web servers and isolate internal applications from

⁶Digital signatures are also part of message-layer security.

outsiders, run applications within sandboxes ⁷ in order to limit risk/damage of compromise, perform audition of the system and use intrusion detection systems (IDS) to look out for strange activities [24].

With SOA, services and code get exposed to external parties, making room for attack opportunities. Standard network perimeter firewalls do not protect the interfaces of the available services of an application, since they can be reached via ports usually left open for web traffic. With web services, for example, communication between services is done via SOAP messages sent over protocols such as HTTP/HTTPS⁸, SMTP, FTP, in order for them to flow without restrictions through network firewalls [29], which do not detect threats inside the application layer SOAP messages [6]. Application layer firewalls (e.g., XML firewalls) could be used to address this problem [30].

Information available through a service's description can be used by attackers to gather interesting information about an application, and formulate attacks, as explained by Yunus et al. [30] with regard to web services and WSDL. In another study [31], WSDL is also described as a good way to learn about functions, and parameters, in order for an attacker to break the application. In this case, access control policies to prevent unauthorized access to business logic, and strict input validation to prevent malicious inputs are of great importance.

Security in SOA should keep the services as open and simple to use as possible [24], and preferably be implemented as a service, as argued by Josuttis [12] and Kanneganti et al. [24], in order to deal with the different security contexts among other services. Security should not be a barrier to the interoperability aspect of SOA services. In an environment where services can be used by an organization and its partners, without boundaries, the complexity of designing and implementing a secure system rise considerably and need to be well thought and taken care of.

The next section present the specifics about Web Services, including the security standards that help addressing the issues mentioned above.

2.2.2 Web Services

Web Services are the most used approach to implement SOA applications. They can be seen as a set of standards that cover interoperability between systems communicating via a network [12] or as *"a software system designed*

⁷A sandbox provides a tightly-controlled set of resources for programs to run.

⁸HTTPS designates HTTP over SSL/TLS [28].

to support interoperable machine-to-machine interaction over a network”, as defined by the World Wide Web Consortium (W3C) ⁹ [32].

2.2.2.1 Basic Standards

There are many standards related to Web Services, specified by different standards bodies, such as the W3C, Organization for the Advancement of Structured Information Standards (OASIS) ¹⁰, and the Web Services Interoperability Organization (WS-I) ¹¹ [33], but to be able to understand how they are realized the fundamental ones are just five [12]:

- **XML:** The Extensible Markup Language (XML) lets you structure information in a hierarchical way, via the use of tags, incorporating semantic meaning to it [23]. XML can be used to describe models, formats and data types, which in turn makes it easier to exchange data between different applications [24]. One can define his own markup language to structure data based on specific needs, for example:

```
<rec:recipes xmlns:rec="http://international-recipes.com/myRecipe">
  <rec:recipe id="1">
    <rec:name>Brigadeiro</name>
    <rec:description>This is a brazilian candy</description>
    <rec:ingredient id="1" qt="1.can">Condensed milk</ingredient>
    <rec:ingredient id="2" qt="5_spoons">Powder chocolate</ingredient>
    <rec:HowToPrepare>Do this and that</HowToPrepare>
  </rec:recipe>
  <rec:recipe id="2">
    ...
  </rec:recipe id="2">
  ...
</rec:recipes>
```

Listing 2.1: Example of an XML data structure

In the above example we see how a simple data structure can be accomplished, by using the top level tag element `<recipes>` which groups all the individual `<recipe>` declarations. A recipe is then defined by its five sub-elements. The *id* attribute represents the unique identifier of the elements *recipe* and *ingredient*. The prefix *rec* represents the namespace (or vocabulary), identified by "http://international-recipes.com/", that is associated with an element, and it is necessary to avoid ambiguity between elements with the same name but are defined differently, by different organizations for example. In this way, it is possible to share, or even merge, recipes between applications that understand a predefined format, which bring us the concept of XML Schema.

XML Schema is a model document that defines the structure of XML documents, specifying what kind of elements are allowed, as well as the

⁹<http://www.w3.org>

¹⁰<http://www.oasis-open.org>

¹¹<http://www.ws-i.org>

order they are supposed to appear and with what content [34]. This is useful for using schema validators in order to verify if the XML document follows the expected structure and can be used by an application.

- **HTTP:** The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems [35]. It has been in use in the World Wide Web as a request/response standard between a client and a server since 1990. It is one possible protocol that can be used to send SOAP messages from service requestor to service provider, besides SMTP, FTP, among others [23].
- **WSDL:** Web Services Description Language (WSDL) defines what information a service provider requires to perform its service, by describing its signature (name and parameters) and its binding and deployment details (protocol and location) [12]. At the moment of this writing there are versions 1.1 and 2.0 of WSDL.

A Web Service is described in two stages [36]:

- Abstract description: Establishes the interface characteristics of the Web Service without any reference to the technology used to host or enable a Web Service to transmit messages. In that way the description continues valid even if there are changes to the underlying technology. This *interface* (called "portType" in WSDL 1.1) consists of operations with input and output parameters. These parameters are defined in <message> sections (or <types> section) [12].
- Concrete description: Defines the communication aspects of the Web Service. A *binding* specifies the protocol and data formats for the service operations. An *endpoint* (or "port") defines the physical location where the service is available. Finally, a *service* groups together related endpoints.

It is easier to understand these concepts with an example (in WSDL 1.1):

```
<definitions name="RecipeExchange"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:rec="http://international-recipes.com/myRecipe"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema ...>
      ...
    </xsd:schema>
  </types>
  <message name="getRecipeRequest">
```

```

    <part name="recipeId" type="xsd:string"/>
  </message>
  -->
  <message name="getRecipeResponse">
    <part name="recipeResp" type="rec:recipe"/>
  </message>

  <portType name="RecipeExchangePortType">
    <operation name="getRecipe">
      <input message="getRecipeRequest"/>
      <output message="getRecipeResponse"/>
    </operation>
  </portType>

  <binding name="RecipeExchangeSOAPBinding" type="RecipeExchangePortType">
    <soap:binding style="rpc"/>
    ...
  </binding>

  <service name="RecipeExchangeService">
    <port name="RecipeExchange" binding="RecipeExchangeSOAPBinding">
      <soap:address
        location="
          "http://localhost:8080/services/RecipeExchange"/>
    </port>
  </service>
</definitions>

```

Listing 2.2: Example of a WSDL document

We can see how a WSDL file is defined and its relation with listing 2.1, where the element *recipe* is used as the return type for the *getRecipeResponse* message. The operation *getRecipe* triggers this output message after being called and receiving a *recipeId*.

- **SOAP:** Before version 1.2, SOAP was an acronym for "Simple Object Access Protocol", but this is no longer valid [37]. SOAP provides a standard model for exchanging structured information between applications in a distributed environment. It uses XML to define an extensible messaging framework providing a message format that can be exchanged over different underlying protocols. The framework was designed to be technology independent.

The SOAP processing model assumes that a SOAP message originates at an initial SOAP sender and is sent to an ultimate SOAP receiver via zero or more SOAP intermediaries [37].

A SOAP message has up to four components [26]:

- Envelope: It is the outermost part of the message and identifies it as a SOAP message rather than any other kind.
- Header: Provides a way to add user defined extensions to SOAP. These can be activities like authorization and authentication, that are not normally included for a service to be carried out.
- Body: Contains the information, for a final SOAP receiver, that is generated from a service request or response.

- Fault: Used to report an error and related information, it is a direct child of the SOAP body element, and should also be the only one [37].

Below we can observe an example of a SOAP request message and a SOAP response message, both related to the WSDL example in listing 2.2.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    <rec:getRecipeRequest xmlns:rec="http://international-recipes.com/myRecipe">
      <rec:recipeId>1</rec:recipeId>
    </rec:getRecipeRequest>
  </soap:Body>
</soap:Envelope>
```

Listing 2.3: Example of a SOAP request message

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    <rec:getRecipeResponse xmlns:rec="http://international-recipes.com/myRecipe">
      <rec:recipeResp>
        <rec:name>Brigadeiro</name>
        <rec:description>This is a brazilian candy</description>
        <rec:ingredient id="1" qt="1_can">Condensed milk</ingredient>
        <rec:ingredient id="2" qt="5_spoons">Powder chocolate</ingredient>
        <rec:HowToPrepare>Do this and that</HowToPrepare>
      </rec:recipeResp>
    </rec:getRecipeResponse>
  </soap:Body>
</soap:Envelope>
```

Listing 2.4: Example of a SOAP response message

- **UDDI:** The Universal Description, Discovery, and Integration (UDDI) is a standard for a platform-independent, XML-based services registry [26]. It provides means for discovery of services and retrieval of their WSDL descriptions [24]. As long as a service is supposed to be used, it needs to be discovered somehow, but first need to be published. The UDDI can be seen as the registry box in figure 2.2.

A standard that can be considered as an alternative to UDDI is the ebXML [5], but it is also possible to extend other registry solutions such as Lightweight Directory Access Protocol (LDAP) ¹² or Java Naming and Directory Interface(JNDI) ¹³ [12], for example.

This concludes the fundamental information about Web Services. The next section deals with the security standards that need to be taken into account when developing secure services.

¹²<http://tools.ietf.org/html/rfc4511>

¹³<http://java.sun.com/products/jndi/>

2.2.2.2 Security Related Standards

The fundamental Web Services standards don't deal with security, they were designed to provide connectivity [29] and, as we have seen in section 2.2.1.1, this can affect the system's security considerably. Considered as a gap, web services security was brought to life through several standards, introduced by different organizations, that do not necessarily lead to interoperability [12]. A possible solution is to use a WS-I¹⁴ Basic Security Profile, that is *"based on a set of non-proprietary web services specifications, along with clarifications and amendments to those specifications which promote interoperability"* [38].

Before going into details, it is relevant to have an overview of which security aspects the main standards cover. We can see in the table 2.1 the functional security requirements, the standards that apply to each of them, and the organizations behind the specifications.

Authentication	OASIS WS-Security, OASIS WS-Trust, OASIS SAML, OASIS WS-SecureConversation, WS-Federation ¹⁵ , W3C XML Key Management Specification(XKMS)
Authorization	OASIS XAMCL, OASIS SAML, WS-Security
Confidentiality	W3C XML Encryption, WS-Security
Integrity	W3C XML (Digital) Signature, WS-Security
Non Repudiation	W3C XML (Digital) Signature, WS-Security

Table 2.1: Security requirements and the associated standards

WS-Security (WSS) was initially developed by Microsoft, IBM and Verisign, being submitted to OASIS for standardization [39]. We see that WSS is related to all the key requirements mentioned in the table 2.1, the reason is that it explains how to extend SOAP messages to accommodate security, based on other standards designed for XML security, such as XML Encryption and XML Signature (derived from a joint work between IETF and W3C [25]).

Starting with the XML security standards, we have:

¹⁴The WS-I is an open industry organization that promotes best practices for web services interoperability. They create profiles and supporting testing tools based on best practices for selected sets of web services standards. More information at <http://www.ws-i.org/>.

¹⁵Several organizations contributed to this specification. As indicated in <http://www.ibm.com/developerworks/library/specification/ws-fed/>, these are: BEA Systems, BMC Software, CA, Inc., IBM, Layer 7 Technologies, Microsoft, Novell, VeriSign.

- **XML Signature:** It is a standard for signing all or part of an XML document or even external documents [25]. As specified by the W3C [40], "*XML Signatures provide integrity, message authentication, and/or signer authentication services for data of any type*". Signer authentication is what we call a mechanism for non-repudiation. Clearly, we see how the basic concept of XML Signatures is identical to regular digital signatures.

One important issue that is taken care of in this standard is the fact that syntactically equivalent XML messages could produce different signatures. The solution is the use of the XML canonicalization (c14n) specification, that transforms any well-formed XML document in a standard equivalent representation [24].

Listing 2.5 show a simple example of a signature. Later we will see how this signature, and other XML security standards, can be incorporated in SOAP messages through WS-Security.

```

<rec:recipes xmlns:rec="http://international-recipes.com/myRecipe">
  <rec:recipe id="1">...</rec:recipe>

  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

    <ds:SignedInfo>
      <ds:CanonicalizationMethod Algorithm="..." />
      <ds:SignatureMethod Algorithm="..." />

      <ds:Reference URI="#1">
        <ds:DigestMethod>...</ds:DigestMethod>
        <ds:DigestValue>...</ds:DigestValue>
      </ds:Reference>

    </ds:SignedInfo>

    <ds:SignatureValue>...</ds:SignatureValue>

    <ds:KeyInfo>...</ds:KeyInfo>

  </ds:Signature>
</recipes>

```

Listing 2.5: Example of an XML signature

To understand this example, let us explain the three sub-elements that the *Signature* element has [25]:

- **SignedInfo:** Contains information of what is actually signed, indicating it via the *URI* attribute of the *Reference* element, that points to the *id* of *recipe*. The *DigestMethod* specifies the algorithm used to calculate the digest (hash) of the information to be signed. The *DigestValue*, as the name says, represents the digest value. *SignatureMethod* defines the algorithm used to sign and verify signatures.
- **SignatureValue:** Is the signature itself, resulting from the encryp-

tion of the digest of the *SignedInfo* element.

- KeyInfo: Provides the key that will be used to validate the signature or the necessary information to look up the key.
- **XML Encryption:** Similarly to XML Signature, XML Encryption lets you encrypt all or part of an XML document or other arbitrary data [41], using one or more keys. In a context where messages are exchanged between different parties, it is possible that parts of a message are visible to one party while completely hidden to another [25]. As mentioned before, SSL/TLS provides point-to-point confidentiality but it is not sufficient to protect a message's contents from possible eavesdroppers (intermediary nodes), on the other hand, XML Encryption is a useful mechanism for secure end-to-end communication, by providing persistent encryption, which also allows for encrypted messages to be stored [39].

Using the example presented in the listing 2.1, we can think of encrypting only the ingredients and the preparation method, as shown in listing 2.6:

```
<rec:recipe id="1">
  <rec:name>Brigadeiro</name>
  <rec:description>This is a brazilian candy</description>

  <xenc:EncryptedData id="EncRec"
    xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
    Type='http://www.w3.org/2001/04/xmlenc#Content'>
    <xenc:EncryptionMethod Algorithm='...' />
    <ds:KeyInfo xmlns:ds='http://www.w3.org/2000/09/xmldsig#'>
      ...
    </ds:KeyInfo>
    <xenc:CipherData>
      <xenc:CipherValue>B3F28AC0</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>

  <xenc:EncryptedKey Id='EncKey' ...>
    <xenc:EncryptionMethod Algorithm='...' />
    <ds:KeyInfo ...></ds:KeyInfo>
    <xenc:CipherData>...</xenc:CipherData>
    <ReferenceList>
      <DataReference URI='#EncRec' />
    </ReferenceList>
  </xenc:EncryptedKey>

</rec:recipe>
```

Listing 2.6: Example of an XML Encryption

Explaining the main elements [25]:

- EncryptedData: This element replaces the information to be encrypted, in our case the *ingredient* and *HowToPrepare* elements, which are *contents* of the *recipe* element. We can also see how the encryption algorithm is specified, and the information regarding the encryption key. The element *CipherData* contains the encrypted value.

- EncryptedKey: Provides an encrypted encryption key used in the *EncryptedData* block. The usefulness of this is to use a symmetric key to encrypt (parts of) a message and transmit this key encrypted with the recipient's public key. The element provides all the information necessary for the receiver to decrypt the symmetric key, along with a list of elements that were encrypted with that key. This element is optional, in case the involved parties already agreed on some key.
- **XKMS**: The W3C defines it as a set of *"protocols for distributing and registering public keys, suitable for use in conjunction with the standard for XML Signatures ... and companion standard for XML encryption"* [42]. XKMS provides the means to access a trusted PKI as a service, allowing a simple client to use complex key management functionality [25]. There are two services specified by XKMS:
 - XML Key Information Service Specification (X-KISS): Is *"a protocol to support the delegation by an application to a service of the processing of key information associated with an XML signature, XML encryption, or other usage of the XML Signature"* [42].
 - XML Key Registration Service Specification (X-KRSS): Is *"a protocol to support the registration of a key pair by a key pair holder, with the intent that the key pair subsequently be usable in conjunction with the XML Key Information Service Specification or a Public Key Infrastructure (PKI)"* [42].
- **SAML**: Security Assertion Markup Language is, as defined in the SAML Executive Overview [43], *"an XML-based framework for communicating user authentication, entitlement, and attribute information"*. SAML messages **assert** the (portable) identity and authorizations of a subject, and can be used by different trust domains to grant access to resources to this specific entity [25]. This process establish a *shared identity* between different domains and is called *identity federation*. There are three types of assertions that can be provided by an assertion issuer (SAML authority), defined by the core standard [44] as:
 - Authentication: *"The assertion subject was authenticated by a particular means at a particular time"*.
 - Attribute: *"The assertion subject is associated with the supplied attributes"*.
 - Authorization Decision: *"A request to allow the assertion subject"*

to access the specified resource has been granted or denied”.

SAML specifies different profiles [45] that define the use of its assertions and request-response messages. A profile defines constraints and/or extensions for the use of SAML in an application, with the goal of improving interoperability [43]. The web browser SSO profile is the most relevant for us, enabling a scenario where a web user either accesses a resource at a service provider, or accesses an identity provider for authentication [45]. The identity provider generates assertions that can be verified by the service provider to establish a security context for the user, including access to desired resources. Figure 2.5 presents the SSO concept.

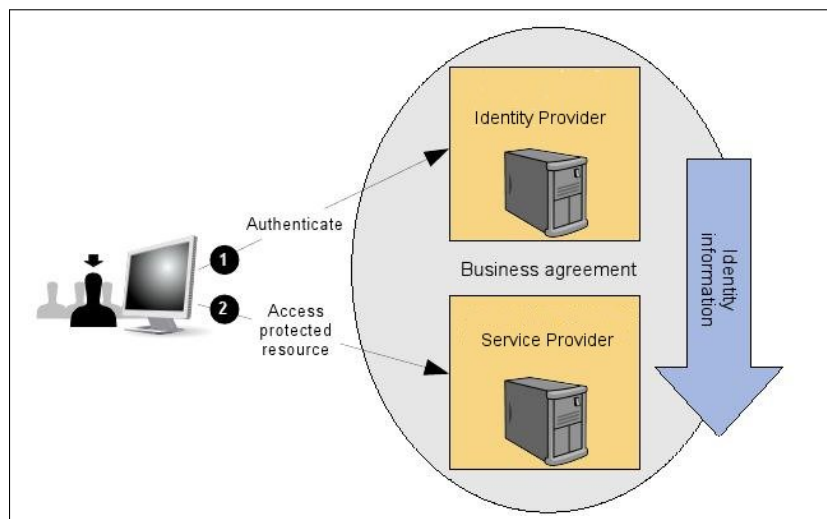


Figure 2.5: The mechanics of the SSO concept. This figure was based on the SAML V2.0 Technical Overview [46].

There are several authentication mechanisms supported by SAML, such as passwords, Kerberos, Public Key – X.509, Public Key – PGP and smartcards [47]. The following listing presents an example where a user was authenticated via a password:

```
<saml:Assertion ...>
  <saml:AuthenticationStatement
    AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password"
    AuthenticationInstant="2009-03-17T14:02:00Z">
    <saml:Subject>
      <saml:NameIdentifier
        Format="urn:oasis:names:tc:SAML:1.0:assertion#emailAddress">
        johndoe@somewhere.com
      </saml:NameIdentifier>
      <saml:SubjectConfirmation>
        <saml:ConfirmationMethod>
```

```

    ...
    </saml:ConfirmationMethod>
  </saml:SubjectConfirmation>
</saml:Subject>
</saml:AuthenticationStatement>
</saml:Assertion>

```

Listing 2.7: Authentication assertion for a "johndoe" using a password. Example adapted from Rosenberg et al. [25]

- **XAMCL:** eXtensible Access Control Markup Language is a specification for representing authorization and entitlement policies [25]. OASIS defines it as *"an XML-based language for access control"* [48]. The standard describes two languages:
 - Access control policy language: This language *"is used to express access control policies ('who can do what when')"* [48].
 - Request/Response language: This language *"expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses)"* [48]. Access requests are handled by a policy enforcement point (PEP). A policy decision point (PDP) is contacted to check if the request is allowed before granting access to services [49].

The need for a standard way to define security policies is one of the motivations for XACML. In general, a large enterprise has several points of enforcement for its security policy and different information systems, each one requiring their own set of rules. The usual approach is to manage the configuration of each point of enforcement independently, what makes the process of modifying the security policy an expensive and unreliable one. XACML allows the enterprise to manage the enforcement of all the elements of its security policy in all the components of its information systems [49].

XACML defines three top-level policy elements [49]:

- `<Rule>`: This element contain a boolean expression that can be evaluated alone. It is a basic unit meant to be used in several policies.
- `<Policy>`: This element contains a set of `<Rule>` elements and a procedure to join their evaluations.
- `<PolicySet>`: This element contains a set of `<Policy>` or other `<PolicySet>` elements, and a a procedure to join their evaluations. It is the standard way to combine different policies into a unified policy.

The newest versions of XACML and SAML have been designed to complement each other; for example, an XACML policy can specify what a service provider should do when it receives a SAML assertion, and XACML-based attributes can be expressed in SAML [48].

Starting with WS-Security and using it as a base for extension, we have some other standards of the WS-* branch, which can be composed to solve specific security needs [25]. The main ones are:

- **WS-Security:** The goal of WS-Security is to provide mechanisms for securing web services via a set of SOAP header extensions [50]. Three main mechanisms are defined by the standard: *"ability to send security tokens as part of a message, message integrity, and message confidentiality"*.

Message integrity is provided by XML Signature while message confidentiality is achieved by using XML Encryption; both can be used in conjunction with security tokens. A security token represents a set of claims, where a claim is a statement about an entity (e.g. name, identity, key, etc). The specification allows a number of technologies to be used for including security tokens, such as X.509 certificates, Kerberos tickets, Username or SAML assertions [50]. These mechanisms enable the desired end-to-end (message-layer) security.

The following example shows how to include a signature in a SOAP message:

```
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken
        ValueType="wsse:X509v3"
        EncodingType="wsse:Base64Binary"
        wsu:Id="X509Token">
        FigEzZCRF1EgILBAgIQEmtJZc0rqrKh5i ...
      </wsse:BinarySecurityToken>

      <ds:Signature>
        <ds:SignedInfo>
          ...
          <ds:Reference URI="#body">
            ...
          </ds:Reference>
        </ds:SignedInfo>

        <ds:SignatureValue>...</ds:SignatureValue>

        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#X509Token"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </soap:Header>

  <soap:Body wsu:Id="body">
    ...
  </soap:Body>
</soap:Envelope>
```

Listing 2.8: Inclusion of a digital signature in a SOAP header.

Example adapted from Rosenberg et al. [25]

We see the elements with the namespace *wsse* representing the WS-Security extensions. The `<BinarySecurityToken>` element identifies a X.509 public key certificate that is to be used to verify a signature. The element `<SecurityTokenReference>` points to the token related to its enclosing signature, which references the `<Body>` element.

The listing 2.9 presents the use of encryption on a SOAP message:

```
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod Algorithm="..." />
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier EncodingType="wsse:Base64Binary"
              ValueType="wsse:X509v3">
              F2JfLa0GXsq...
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>...</xenc:CipherData>
      <xenc:ReferenceList>
        <xenc:DataReference URI="#body" />
      </xenc:ReferenceList>
    </xenc:EncryptedKey>
  </wsse:Security>
</soap:Header>
<soap:Body>
  <xenc:EncryptedData Id="body">
    ...
  </xenc:EncryptedData>
</soap:Body>
</soap:Envelope>
```

Listing 2.9: Use of encryption on a SOAP body element. Example adapted from Rosenberg et al. [25]

In this example, the public key used to encrypt the session key is inside the `<SecurityTokenReference>` element.

- **WS-Policy** and **WS-SecurityPolicy**: The WS-Policy standard [51] specifies a framework that allows web services to exchange their constraints and requirements, which are expressed as policy assertions [52], in order for them to discover what is necessary to interact. Such information can include the supported algorithms for encryption and signatures, which fields should be encrypted and/or signed, and what kind of security token is required [25]. These requirements are used for both ways of a communication. WS-SecurityPolicy defines the security policy assertions for the WS-Policy framework. The following example illustrates a policy defined with such assertions:

```
<wsp:Policy
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
```

```

    xmlns:wsp=" http://www.w3.org/ns/ws-policy" >
    <wsp:ExactlyOne>
    <wsp:All>
    <sp:SignedParts>
    <sp:Body/>
    </sp:SignedParts>
    </wsp:All>
    <wsp:All>
    <sp:EncryptedParts>
    <sp:Body/>
    </sp:EncryptedParts>
    </wsp:All>
    </wsp:ExactlyOne>
    </wsp:Policy>

```

Listing 2.10: A WS-Policy defined by WS-SecurityPolicy assertions.
 Example extracted from the WS-Policy standard [51]

Policy operators, such as "ExactlyOne", group policy assertions into policy alternatives, and therefore the policy in listing 2.10 defines that the SOAP body will be either signed or encrypted when a web service is invoked [51].

- **WS-Trust:** The goal of this specification is to enable applications to participate in trusted SOAP message exchanges. As defined in the standard [53], WS-Trust provides:
 - *"Methods for issuing, renewing, and validating security tokens"*.
 - *"Ways to establish, assess the presence of, and broker trust relationships"*.

A security token service (STS) is a web service responsible for issuing security tokens, and can be seen as a trust broker [53]. In this case, when a service requestor, from a different (possibly unknown) trust domain, needs a security token to access a service provider, the requestor can ask the STS for such a token, which can be used to request a service from the provider. The provider will, then, determine if it will accept this request, provided that it trusts the issuing STS and can verify the token in some way, such as requesting another token service to validate it or performing the validation itself [53]. During token acquisition it is possible to use a challenge-response protocol to guarantee message freshness and verification of authorized use of the token by a requestor.

- **WS-SecureConversation:** Build on top of both WS-Trust and WS-Security, WS-SecureConversation provides mechanisms for establishing and sharing a security context, as well as session key derivation. WS-Security focuses on individual message authentication, while WS-SecureConversation is used to establish a security context between two parties, which allows the authentication of several messages, and improves the communication performance. Another advantage is the ne-

gotiation of session key(s), derived from a shared secret associated with the context, for encryption/decryption and signing/verification in the secure context. One possible way to establish a security context is the use of a challenge-response protocol [54].

- **WS-Federation:** This specification was defined by several organizations and, at the time of this writing, is in process to be a OASIS standard [55]. WS-Federation is a framework that builds on the WS-* specifications, such as WS-Security and WS-Trust, to provide an extensible mechanism for federation, a concept already mentioned for SAML. A federation should be able to accommodate different security domains, making it possible to authenticate users in one domain and use its declaration (brokering) of identity, attribute, authentication and authorization assertions to request access to resources in other domains. Trust relationships between domains will influence the final access control decision. Each security domain can use different security tokens, e.g., X.509 certificates, Kerberos tickets or SAML assertions, and as in WS-Trust it is possible to contact an STS to acquire security tokens from the resource's domain or validate security tokens from the requestor's domain via the resource's STS [56].

2.3 MPOWER Platform

In this section we present the subject of our test case, the MPOWER project, including the goals, the architecture, security aspects and two *proof-of-concept applications* (POCAs).

2.3.1 About the Project

The MPOWER project (Middleware Platform for eMPOWERing cognitive disabled and elderly) aims to develop an open middleware platform to simplify and speed up the task of developing and deploying services for persons with cognitive disabilities and elderly. The platform was defined within an iterative process including end-user requirements, design, platform development, development of proof-of-concept applications and end-user trials [10].

One of the main motivations is the, already mentioned, fact that healthcare services depends on different systems that need to share information. Due to lack of interoperability, this task can be quite challenging. By applying an agile model-driven development process, the MPOWER project developed a service framework that provides reusable, flexible, and interoperable service specifications and implementations [57].

Two end user scenarios have been described in order to span out the needs in

relation to the MPOWER architecture, middleware components and proof-of-concept applications. They cover the focus areas [10]:

- Norwegian POCA: A collaborative environment for distributed and shared care, providing requirements for information security, information models, context awareness, usability and interoperability.
- Polish POCA: A SMART HOUSE environment, providing requirements for information security, information models and usability.

The platform should make it commercially feasible for the IT industry to develop distributed integrated applications offering innovative services to cognitive disabled and elderly. The focus is to make it possible to develop services where [10]:

- Bio-sensors and SMART HOUSE technology are integrated.
- Interoperability between systems is central.
- The system has to cope with various user contexts (e.g., change of terminal used for interacting with the system), where usability is central.
- Information security is enabled through security components facilitating storage security, communication security, access rights and client security.

The project promotes standardization by aligning its work with OMG¹⁶/HL7¹⁷ (Object Management Group and Health Level Seven) and CEN/TC 251¹⁸ (European Committee for Standardization of Health Informatics).

The figure 2.6 presents the MPOWER framework:

¹⁶<http://www.omg.org/>

¹⁷<http://www.hl7.org/>

¹⁸<http://www.cen.eu/CENORM/Sectors/TechnicalCommitteesWorkshops/CENTechnicalCommittees/CENTechnicalCommittees.asp?param=6232&title=CEN\%2FTC+251>

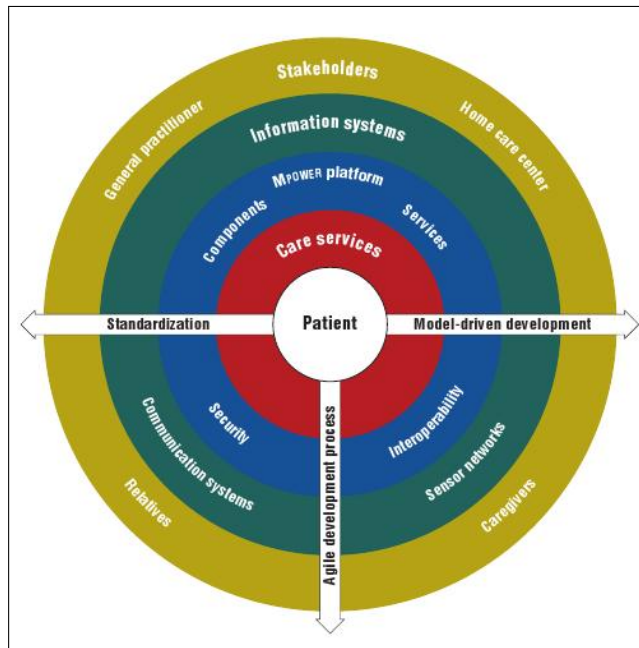


Figure 2.6: The MPOWER framework. Figure extracted from Mikalsen et al. [57]

2.3.1.1 Architecture

The MPOWER platform uses SOA as a reference architecture, and implements its services as Web Services. The figure 2.7 shows the MPOWER architecture.

The specified layers are organized into three groups [58]:

1. Application Specific Components

These components are, usually, designed for a specific purpose in an organization. Reusability is not the primary concern in this case. The two layers included in this group are:

- Application Layer: It provides a user interface for the use of the underlying services.
- Business Process Layer: It includes compositions of services, which then act together as a single application.

2. Domain Specific Components

These components are specific for the operational domain, e.g., smart-house solutions and homecare systems. The services are designed with focus on reusability and loose coupling. This group has only one layer:

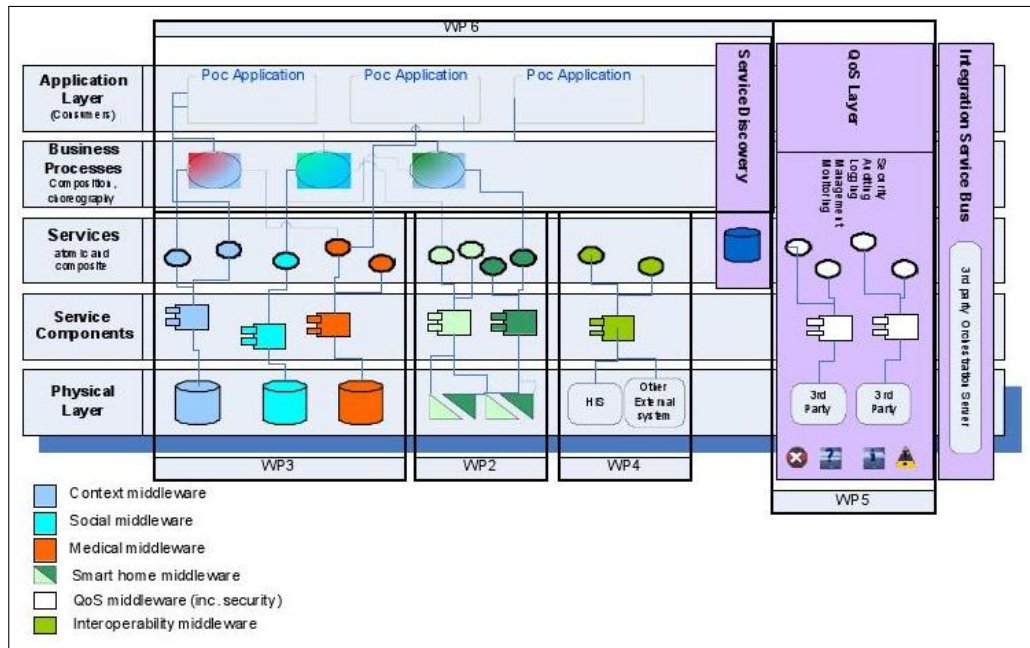


Figure 2.7: MPOWER reference architecture. Figure extracted from the MPOWER Project Deliverable D1.1 [58]

- Services Layer: Implemented, and other exposed, services reside in this layer. They can be discovered or statically bound and then invoked, or included in a composite service.

3. System Specific Components

These components are independent of both application and domain and can be used by many applications in different domains. However, they are strongly coupled with the underlying system, being a software component or a hardware interface. This group includes two layers:

- Service Components Layer: This layer provides a high-level access to the functionality of components and services in the resource layer.
- Resource Layer: It consists of existing applications, such as databases, and information from (smart) sensors, e.g., temperature sensors.

2.3.2 Security Requirements

As mentioned before, healthcare systems require strict measures to protect sensitive personal data. The MPOWER project makes use of the Norwe-

gian implementation of the European Union’s Data Protection Directive 95/46/EC, together with other relevant laws and regulations, in order to derive a set of security requirements based on a service oriented architecture [59]. As a direct product of this work, a list of security requirements was generated, which could help in future projects related to healthcare. The table 2.2 presents them.

The inclusion of references to the laws and regulations related to the definitions of these requirements is vital to provide traceability to their origins, and verify the desired compliance [59].

Requirement type	Description
Identification and authentication requirements	Services should identify and verify the identity of all its human users before allowing them access to their resources. [60] §21, [61] §14, [62] §3-6, [63] §2-11, §2-12
	Services should identify and verify the identity of corresponding services before they are allowed to communicate. [64] §5-4, [61] §14
Authorization requirements	Services should verify the authorization level of users before access to sensitive data can be given. [60] §21, §22, [62] §3-6, [63] §2-11, §2-12
Integrity requirements	The platform should support integrity protection of sensitive personal data while it is stored. [64] §5-5, [65] §16, [66] §13, [63] §2-13
	The platform should be able to detect unauthorized manipulation of data that is being transmitted. [64] §5-5, [65] §16, [66] §13, [63] §2-13
Privacy requirements	The platform must protect any stored sensitive personal data from unauthorized access. [62] §3-6 and §5-3, [65] §16, [66] §13, [63] §2-11
	Personal sensitive data must be confidentiality protected while transmitted over open, untrusted communication lines. [64] §5-5, [62] §3-6 and §5-3, [65] §16, [66] §13, [63] §2-11
Security Auditing Requirements	The platform should be able to log security incidents, such as failed login attempts or unauthorized access attempts to services in order to discover and trace system abuse. [63] §2-14
	The platform should be able to log activities related to access of sensitive information. [63] §2-14
Survivability requirements	Input validation should be performed at time of data reception to reduce threats represented by malicious content and malformed packets. [64] §5-9
	Multiple levels of security should be ensured to avoid a single point of failure. [64] §5-4
	Data freshness should be controlled to prevent chances of replay attacks. [64] §5-5
Non-repudiation requirements	A patient journal should show who has added content, e.g., through electronic signatures. [60] §40, [61] §7

Table 2.2: Security requirements and the associated laws and regulations.
Table adapted from Jensen et al. (2009) [59]

2.3.2.1 Security Design

Based on the security requirements identified previously, the security components were designed, and their dependencies mapped, as the figure 2.8 shows us.

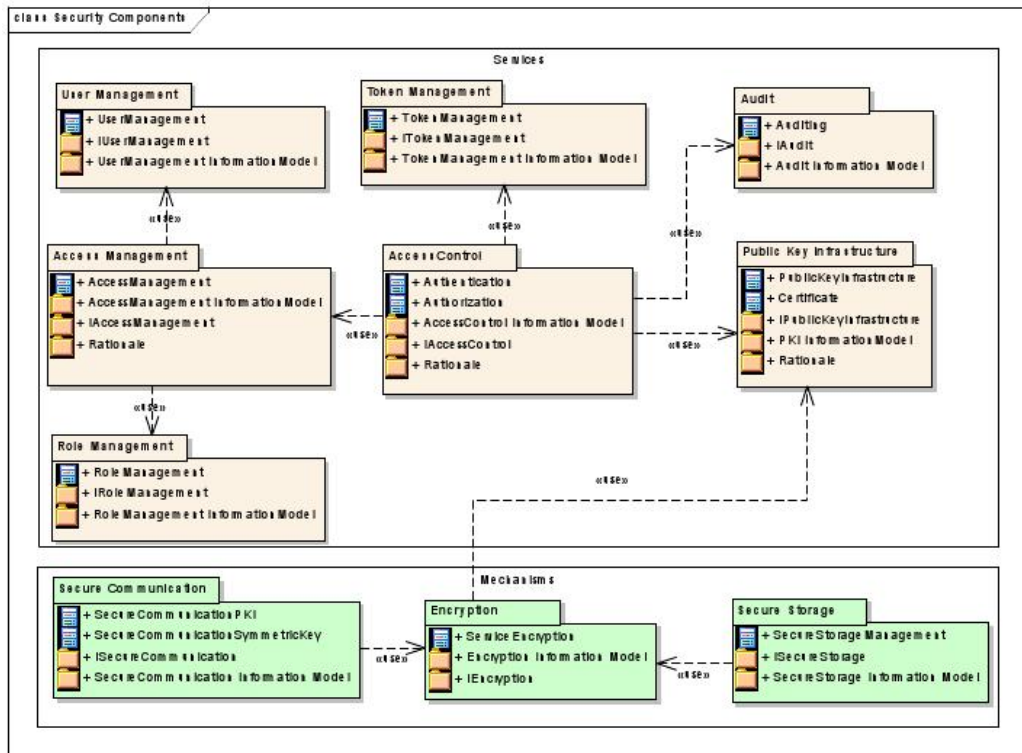


Figure 2.8: MPOWER security components. Figure extracted from the MPOWER Project Deliverable D5.2 [67]

Important to notice the difference between security services and security mechanisms, as defined in the MPOWER Project Deliverable D5.2 [67]:

- **Services** are security components that can be implemented as web services.
- **Mechanisms** describe functionality that is desirable, and in many cases required, as an inherent part of MPOWER services to ensure a satisfying level of security.

The table 2.3 provides us with a short description of what each component in figure 2.8 is responsible for:

Component	Description
Access Control	Access control includes the Authentication and Authorization services. The Authentication service verifies a user's credentials and allows access to the system only if they are valid. The Authorization service determines which operations and data an authenticated user can access, allowing access to resources only to legitimate, authorized users. Authorization in MPOWER is based on a Role-Based Access Control (RBAC) scheme, extended by a Context-Based protection ¹⁹ .
User Management	The user management service enables the administrator to manage the users of the system. The administrator may: <ul style="list-style-type: none"> • Add/delete users • Update the user's roles • Get the user information
Role Management	The role management service enable the administrator to manage the roles of the system. The administrator may: <ul style="list-style-type: none"> • Add/delete roles • Assign users to roles • Get the role information • Get the users assigned to a role
Access Management	The access management service manages the permissions and access profiles associated with the access control system.
Token Management	The token management service is used by the Authentication and Authorization services to manage the login sessions.
Public Key Infrastructure	The PKI service provides the interface for general management of certificates, i.e, issue, renew and revoke, and verification of the validity of a certificate.
Audit	The audit service provides an interface enabling other services to store data that needs to be logged for future audit purposes. Additionally, it provides operations to retrieve logged data that is necessary for auditing.
Encryption	The encryption mechanism describes functionality related to confidentiality and integrity protection of data.
Secure Storage	The secure storage mechanism describes functionality for storing data securely and for retrieving secured data from storage.
Secure Communication	Secure communication is the mechanism describing functionality needed to secure data being transmitted between two endpoints.

Table 2.3: MPOWER security components. Table adapted from the MPOWER Project Deliverable D5.2 [67]

The implementation of these components should make use of a selection of the standards presented at section 2.2.2.2 to achieve the required level of security, measured in levels of *availability, confidentiality and integrity* [67].

¹⁹Context information is important for limiting a role's access to a subset of target objects, depending on context elements associated to the user that was assigned the role [68]. Such elements include the relationship this user has with the requested information, e.g., a user with a *doctor* role that should have access only to his patients' data.

Chapter 3

Methodologies and Guidelines for Security Assessment

This section presents the motivation for a security assessment methodology, along with some information about one of the most widely accepted, the OS-STMM, and a few recommendations and views from NIST. We also present a testing guide from OWASP and a testing framework from SIFT. At the end, the approach chosen to perform the tests will be described.

As NIST [11] defines:

- *Information security assessment* is a process that determines how effectively an object, or target, being assessed (e.g., host, system, network) meets security requirements. There are three methods that may be used to accomplish this (testing, examination, and interviewing), of which we describe the two most relevant for us:
 - *Testing* is a process that exercises one or more assessment objects, under specific conditions, to compare actual and expected behaviors.
 - *Examination* is the process of checking, inspecting, reviewing, observing, studying, or analyzing one or more targets to facilitate understanding, achieve clarification, or obtain evidence.

Before continuing, it is valid to have in mind the following definitions:

- **Asset:** Everything that has some value to an organization such as a person, information, hardware or software. Each organization should identify what it considers as assets [69].
- **Vulnerability:** "A flaw or weakness in a system's design, implementa-

tion, or operation and management that could be exploited to violate the system's security policy" [13].

- Threat: *"A possible danger that might exploit a vulnerability" [13].*

One major reason for the presence of vulnerabilities in an application is the fact that security is, many times, not considered as part of the software development life cycle (SDLC). The opposite approach could potentially reduce the number of these vulnerabilities as the software is developed [69, 70]. Even when care is taken in the development process, testing the overall security should be part of it, e.g., a component may be secure, but when it interacts with other components, a security compromise can happen [70].

3.1 Benefits of Using a Methodology

Every organization should have some standard methods for performing its activities, including the security tests of its applications. A poorly conducted test, i.e., one that does not find critical vulnerabilities, could greatly put in risk the company's assets, e.g., causing leakage of sensitive data and, consequentially, financial loss. As Herrman points out [71], the disclosure of private information makes the company liable for not complying to government and industry regulations, and to lawsuits from clients or customers, besides the damage to its credibility. At the time of this writing, recent data-breach cases include Heartland [72] and RBS Worldplay [73], not to mention insiders' threats, as it happened with Sprint¹ [74].

The use of a consistent, documented and repeatable methodology allows for benefits such as [11]:

- Structured security testing, minimizing testing risks, where the results are properly documented for future reference, providing the means to assure the test has been conducted as it should be.
- Facilitate the transition of new assessment personnel.
- Effective planning of resources (e.g., staff, hardware, software) to use for performing the assessments, thus reducing overall costs and time to conduct them.

¹Other cases can be found at <http://www.privacyrights.org/ar/ChronDataBreaches.htm>.

3.2 Open Source Security Testing Methodology Manual - OSSTMM

The Open Source Security Testing Methodology Manual (OSSTMM) [75, 76] provides a methodology for a thorough security test, referred to as an OSSTMM audit. As the manual claims, an OSSTMM audit is an accurate measurement of security at an operational level. Since it was created in 2000, the OSSTMM was improved and ended up as a standard for performing reliable security tests. The latest version, 3.0, is not yet available with its full contents to the open public at the time of this writing, so the version 2.2 will be also referenced.

3.2.1 Overview

The primary purpose of the OSSTMM is *to provide a scientific methodology for the accurate characterization of security through examination and correlation in a consistent and reliable way* [75]. The methodology is adaptable to several audit types, such as vulnerability scanning, ethical hacking² and penetration testing³, which is one of the most used approaches to test a system's security [70].

Another purpose is the establishment of guidelines, in order to assure [75]:

1. The test was conducted thoroughly.
2. The test included all necessary channels.
3. The posture for the test complied with the law.
4. The results are measurable in a quantifiable way.
5. The results are consistent and repeatable.
6. The results contain only facts as derived from the tests themselves.

3.2.2 Phases

There are four defined phases in the methodology [75]:

1. Regulatory Phase: The audit requirements, the scope, and the constraints to the auditing of this scope need to be understood in this

²According to the OSSTMM: *"Ethical Hacking refers generally to a penetration test of which the goal is to discover trophies throughout the network within the predetermined project time limit"* [76].

³According to the OSSTMM: *"Penetration Testing refers generally to a goal-oriented project of which the goal is the trophy and includes gaining privileged access by pre-conditional means"* [76].

phase.

2. Definitions Phase: This stage defines the scope in relation to interactions with the targets.
3. Information Phase: In this phase the auditor uncovers several types of misplaced and mismanaged information, such as the target's configuration.
4. Interactive Controls Test Phase: The tests in this stage are focused on penetration and disruption.

3.2.3 Channels and Metrics

The OSSTMM 3.0 considers tests in five channels - Human , Physical, Wireless, Telecommunications, and Data Networks. A set of security metrics, called Risk Assessment Values (RAVs), is used to provide a graphical representation of the security state. The accuracy of these metrics is higher if the security test is thorough, and is also influenced by the auditor's experience. The objective is to quantify three areas (operational security, controls, and limitations) within the scope of the test which together define the *Actual Security* state, presented as a percentage [76].

The methodology identifies a number of possible errors during the testing process, related to observed states and assumptions made by the auditor [76]. Such errors are responsible for lesser accurate results. The following list presents some of them:

- False positive: The target indicates some kind of anomalous action, when this action is, in fact, legitimate [77].
- False negative: The target indicates that an actual intrusive action is a normal one [77]. The main problem is the identification of a non-existent secure state [76].
- Sampling error: The target is a biased sample of a larger system or a larger number of possible states. This may be a result of time constraints on the test or a bias of testing only that which is designated as "important" within a system [76].
- Human error: Errors caused by lack of ability, experience, or comprehension, are always present regardless of methodology or technique. There is an indirect relationship between experience and human error [76].

3.2.4 Rules of Engagement

To better understand this concept, let us define the role of a penetration tester. A penetration tester is an ethical hacker⁴ responsible for conducting security tests on a company's systems and network, by attempting to compromise it, exploiting vulnerabilities [78, 79]. The importance of assets, together with the level of impact of vulnerabilities found, can be related to some metric in order to perform a risk analysis process and better evaluate the security state of the system.

Rules of engagement set the acceptable and ethical practices for marketing and selling testing, performing testing work, and handling the results of testing engagements [76]. Mainly, these rules specify boundaries that a tester should not cross when offering his services to a client. If contractual boundaries are not respected, the tester will be committing illegal actions [78].

There are several factors to take into account, including, but not limited to [76]:

- The scope of test, indicating what parts of a system should be assessed.
- Signed agreement by the client allowing the tester to trespass within the scope.
- Specific permissions for tests involving denial of service and social engineering.
- High risk vulnerabilities must be reported, along with a solution, as soon as they are found.

3.2.5 Tests

Sticking to some concepts from the version 2.2, the security model can be broken up into sections, each one containing a collection of modules, which in turn defines security tests, or tasks. The results of the tasks are considered the output of the module, while the input is the information used to perform each task. One module and some of its tasks are presented [76]:

- Vulnerability research and verification:
 1. Integrate the currently popular scanners, hacking tools, and exploits into the tests.

⁴An ethical hacker is the one that uses the same tools and hacking skills as a malicious hacker, but for defensive purposes.

2. Measure the target organization against the currently popular scanning tools.
3. Attempt to match vulnerabilities to services.
4. Identify all vulnerabilities according to applications.
5. Verify all vulnerabilities found during the exploit research phase for false positives and false negatives.

3.2.5.1 Types of Tests

The manual allows individual testing practices, as long as the requirements defined in it are followed. Six common types of test are presented [75]:

- **Blind:** The auditor engages the target with no prior knowledge of its defenses, assets, or channels. The target is prepared for the audit, knowing in advance all the details of the audit.
- **Double Blind:** The auditor engages the target with no prior knowledge of its defenses, assets, or channels. The target is not notified in advance of the scope of the audit, the channels tested, or the test vectors.
- **Gray Box:** The auditor engages the target with limited knowledge of its defenses and assets and full knowledge of channels. The target is prepared for the audit, knowing in advance all the details of it.
- **Double Gray Box:** The auditor engages the target with limited knowledge of its defenses and assets and full knowledge of channels. The target is notified in advance of the scope and time frame of the audit but not the channels tested or the test vectors.
- **Tandem:** The auditor and the target are prepared for the audit, both knowing in advance all the details of it.
- **Reversal:** The auditor engages the target with full knowledge of its processes and operational security, but the target knows nothing of what, how, or when the auditor will be testing.

3.2.6 Results

To measure both the thoroughness of the test and the security of the target, version 3.0 introduces the Security Test Audit Report (STAR) [75], while version 2.2 has other report templates [76]. The report should contain, among others, all the findings, as well as how and when the tests were conducted for every target in the scope, and possible issues during the tests. The auditor must accept responsibility and limited liability for inaccurate reporting.

3.3 Technical Guide to Information Security Testing and Assessment - NIST-SP800-115

This document provides guidelines for planning and conducting an information security assessment via technical testing and examination techniques. The objective is to identify, validate, and assess technical vulnerabilities, assisting organizations to understand and improve the security posture of their systems and networks [11]. The recommendations offered by this guide can be used for other methodologies, e.g., OSSTMM.

3.3.1 Phased Methodology

NIST recommends at least three phases for an information security assessment methodology [11]:

- **Planning:** Used to gather information for assessment execution, such as assets to be assessed, relevant threats against the assets and security controls to mitigate them; and to define the assessment approach. A plan should address goals and objectives, scope, requirements, team roles and responsibilities, limitations, success factors, assumptions, resources, timeline, and deliverables.
- **Execution:** Main goals in this phase are to identify vulnerabilities and validate them.
- **Post-Execution:** This phase focuses on analyzing identified vulnerabilities to determine root causes, establish mitigation recommendations, and develop a final report.

3.3.2 Technical Assessment Techniques

The guide groups the most common techniques used to conduct security assessments in three categories [11]:

- **Review Techniques:** These are passive techniques, offering minimal risk to systems and networks. They are used to examine systems, applications, networks, policies, and procedures to discover security vulnerabilities. Review techniques include documentation, log, ruleset, system configuration review, network sniffing, and file integrity checking. Code review is also regarded as an important technique for finding security flaws.
- **Target Identification and Analysis Techniques:** These testing techniques can identify active devices and their associated ports and

services, and analyze them for potential vulnerabilities. They may be performed manually or via automated tools. Such techniques include network discovery, network port and service identification, vulnerability scanning, wireless scanning, and application security examination.

- **Target Vulnerability Validation Techniques:** These techniques assure the existence of vulnerabilities, and demonstrate the security exposures that occur when they are exploited. They may be performed manually or via automated tools, depending on the specific technique used and the skill of the test team. These techniques include password cracking, penetration testing, social engineering, and application security testing. The risk involved in this case is the highest, since they have more potential to impact the targets than other techniques.

NIST explains how these techniques can be performed, providing the means for an organization to choose the ones that best fit to its needs. As an example, penetration testing is a technique that performs network port/service identification and vulnerability scanning to identify vulnerable targets [11].

3.3.3 Comparing Examinations and Tests

Examination deals with the review of documents such as policies, security plans, security requirements, standard operating procedures, architecture diagrams, engineering documentation, asset inventories, system configurations, rulesets, and system logs. They are conducted to verify if a system is properly documented, and to analyze aspects of security that are only available through documentation [11].

Testing deals with the activities performed on systems and networks to identify vulnerabilities, and can be executed on the whole enterprise or just selected targets. The use of scanning and penetration techniques can uncover important information regarding potential vulnerabilities and the odds that they are exploited. It is also possible to measure levels of compliance in other areas such as patch management, password policy, and configuration management [11].

Even though testing offers a better view of an organization's security posture than what is obtained via examinations, it is more intrusive and some test cases can impact systems or networks in the target environment. In a production environment, tests which cause high impact (e.g., denial of service) are not allowed by some organizations and examination techniques should be used instead as a complement, helping to identify problems related to security policy and configuration.

3.3.4 Testing Viewpoints

The guide compares two different viewpoints that can be used during a test. One for an external attacker and another for a malicious insider. The knowledge of the organization's IT staff about the test is also considered as a parameter [11].

3.3.4.1 External and Internal

External security testing is conducted from outside the organization's security perimeter, allowing the environment's security posture to appear as seen, usually, from the Internet, with the goal of revealing vulnerabilities that could be exploited by an external attacker [11].

External testing often begins with reconnaissance techniques that search public available information to collect details (e.g., system names, IP addresses, operating systems) that may help the assessor to identify vulnerabilities. Next, enumeration begins by using network discovery and scanning techniques to determine external hosts and listening services. Assessors can use techniques to avoid perimeter defenses such as firewalls, and access control lists, in the same way an attacker would, to be able to invade the internal network. Externally accessible servers are tested for vulnerabilities that could allow such invasions. Commonly allowed protocols (e.g., HTTP, FTP, SMTP) are good initial targets for attacks [11].

For internal security testing, assessors work from the internal network and assume the identity of a trusted insider or an attacker who has penetrated the perimeter defenses. This approach reveals internal vulnerabilities that could be exploited, demonstrating the potential damage this type of attacker could cause. Internal security testing also focuses on system-level security and configuration, including application and service configuration, authentication, access control, and system hardening [11].

Internal testing is not as limited as external testing because it takes place behind perimeter defenses. If both tests are to be performed, the external testing usually takes place first. In the case where the tests are conducted by the same assessors, the advantage is that they do not acquire insider information not available to an attacker, what would affect the legitimacy of the test [11].

3.3.4.2 Overt and Covert

Overt security testing involves performing external and/or internal testing with the knowledge and consent of the organization's IT staff, enabling comprehensive evaluation of the network or system security posture. The staff

may be able to provide guidance to limit the test's impact, and also learn from the activities and methods used by assessors to evaluate and potentially circumvent implemented security measures [11].

Covert security testing takes an adversarial approach by performing testing without the knowledge of the organization's IT staff but with the full knowledge and permission of upper management. In this case, it is important to know if an attack is really happening, or what is being detected is due to the tests, so appropriate measures can be carried out. This type of test is useful for testing technical security controls, IT staff response to perceived security incidents, and staff knowledge and implementation of the organization's security policy [11].

Covert testing fails to identify many vulnerabilities. Since its objective is to examine what impact or damage an adversary can cause, it normally identifies and exploits the most rudimentary vulnerabilities to obtain network access. Boundaries are, usually, defined in order for the test to be considered over when some goal has been reached, such as obtaining a certain level of access [11].

Other disadvantages of covert testing are its time-consuming and cost aspects, caused by the need to slow down its actions to avoid being detected. Overt testing is less expensive, offers fewer risks, and is more frequently used, but does not provide an everyday security view of the target systems, since the staff is already expecting something to happen [11].

3.4 OWASP Testing Guide

The Open Web Application Security Project (OWASP) is "*a worldwide free and open community focused on improving the security of application software*" [80]. They have several projects related to application security, including the *OWASP Testing Guide* [81], that has the objective to address all aspects of testing a web application.

This guide covers the scope of testing, the principles for a successful testing, and the necessary testing techniques. The focus is the integration of testing in the software development life cycle [81]. The main part covers *penetration testing* as a way to find specific vulnerabilities, but other techniques are also mentioned, such as *manual inspections & reviews, threat modeling and code review*. The latter has its own guide, the *OWASP Code Review Guide* [82].

3.4.1 Method

The OWASP method is based on a black box penetration test, where the tester does not have (much) information about the application to be tested. The test is divided in two phases [81]:

- **Passive mode:** The tester gathers information about the application and tries to understand how it works, by playing with it. At the end, the tester should understand all the access points (gates) of the application (e.g., HTTP headers, parameters, and cookies), which represents points of testing.
- **Active mode:** The OWASP methodology is used by the tester to perform the security assessment. The active tests are grouped in nine categories: configuration management, business logic, authentication, authorization, session management, data validation, denial of service, web services, ajax. Each category has its own set of controls to be tested.

To understand how to test the security controls, the guide presents several explanations, examples and references, including tools used to perform the tests.

3.4.2 Web Services

The guide defines controls to be tested specific to web services. These are [81]:

- **WS Information Gathering:** The purpose is to determine the web service's entry points, as described in a WSDL file, such as its methods and inputs.
- **Testing WSDL:** The entry points discovered earlier should be tested, including possible hidden operations (operations not used in a standard SOAP request).
- **XML Structural Testing:** An XML parser needs to inspect the entire XML message to assure it is well-formed. This task is CPU intensive and can be exploited if very large or malformed XML messages are sent, creating a denial of service attack.
- **XML content-level Testing:** Content-level attacks target the server hosting the web service and other applications used by it. This control tests for SQL injection, XPath injection, buffer overflow, and command injection.
- **HTTP GET parameters/REST Testing:** Web services that can be in-

voked by passing parameters via HTTP GET string, can be attacked through use of malicious content in the request.

- Naughty SOAP attachments: This control deals with web services that accept attachments, where the danger lies in processing them on the server and redistribution to clients.
- Replay Testing: This man-in-the-middle attack occurs when an attacker assume the identity of a user, by intercepting a message and resending it, modified or not.

3.4.3 Prioritizing

The need for a focused approach to perform a security test is also tackled by OWASP. It is important to determine the most important security concerns for an organization, that should be prioritized during testing. This is usually done via threat modeling techniques, that should be created early in the SDLC [81]. A threat can be classified, for example, by using STRIDE [83], as Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

After vulnerabilities are found, it is important to estimate the risks associated to the business, in order to decide what is more critical to fix, and include them in the testing report. The basic risk model is: **Risk = Likelihood*Impact**, where likelihood indicates how likely a vulnerability can be discovered and exploited, and impact indicates how badly it affects the technical aspects of the application (e.g., loss of confidentiality, loss of integrity) or the business aspects of the organization (e.g., financial damage, reputation damage) [81]. The guide describes in detail how to achieve such estimates.

3.5 SIFT Web Services Security Testing Framework

SIFT is a consulting, intelligence and training firm [84]. They are specialized in information security, risk and audit services. Noticing that there were no methodology specifically designed for testing web services security, SIFT created an approach to address this niche, the *Web Services Security Testing Framework* [85], using the OWASP Testing Guide as a model.

The Framework is comprised of five standard phases: Threat Modeling, Scoping, Test Planning, Test Execution and Reporting. There are not many news in this process, except the focus on web services. The most relevant information is presented in the *Test Execution* phase; an extensive list of test cases,

many of them specific to web services, grouped in different categories.

3.6 Chosen Method

Our objective is to demonstrate the level of security that a SOA-based health-care system can achieve, but without the illusion of uncovering every possible flaw, as this would be unrealistic. By following some recommendations and guidelines from the methodologies and guides presented above, we defined a custom approach to perform the security testing on the MPOWER platform. Such method consists of the same three phases outlined by NIST, adapted in some ways, and simplified, to represent the reality of our tests:

- **Planning:** In order to plan what kind of tests should be part of our assessment, we look at the security requirements identified in the MPOWER Project Deliverable D5.2 [67], presented in section 2.3.2, associated to the security components, outlined in section 2.3.2.1. This, and knowing that the system is implemented via web services, makes the basis for selecting test cases associated to relevant threats. The tester may also play with the application to better understand it, and use this knowledge to perform incremental changes to the test cases. Chapter 4 presents these test cases and also describes the testing environment.

During this phase is important to characterize:

- Test target: The assessment will be performed on a POCA, that makes use of the MPOWER middleware services. The tests will be conducted in a controlled environment, with no restrictions regarding scope or techniques to be employed.
- Objective: The main objective is to verify if the MPOWER security components offer the desired security level or fall short of their intent. Security problems discovered in the POCA implementation may be considered as a consequence of the assessment.
- Techniques: A mix of techniques will be used to perform the tests, the main one being penetration testing and, to a lesser extent, code and configuration reviews. As McGraw [70] tells us, passing a penetration test does not assure that an application is free of vulnerabilities.

Important to point out that the tester has direct access to the necessary resources (i.e., source code, WSDL, application server, database).

- Scope and limitations: As already mentioned in section 1.3, the scope of the tests will not be as extensive as it could be, leaving out, operating system’s specific flaws, wireless transmissions, physical security and social engineering.

The focus will be on the functionalities provided by the MPOWER platform and the observed security when using them. The test cases define better the intended scope and, since the assessment is performed in a controlled academic environment, there are no restrictions in executing tests that may crash the application, provoking a denial of service.

- Constraints: The team responsible for the tests is small, with most of the work being done by just one person, whom does not have much experience in security testing. The time frame is also quite limited, so the focus will be on the most important test cases. Such factors may influence negatively the end result.
 - Remarks: The MPOWER project lacks documentation regarding the implementation, making it harder to understand the decisions taken during the development cycle. Another issue is related to the security design document, that was elaborated in parallel with the coding process.
- **Execution:** Having established the test cases, these should be put into practice in order to identify vulnerabilities. Chapter 4 explains the tools that are needed for this phase.
 - **Post-Execution:** The results generated during the execution phase are analyzed and presented in chapter 5 together with the respective countermeasures. General recommendations based on these findings are proposed in chapter 6.

The methodologies and guidelines presented before offer much more details, which are necessary when the objective is to conduct a professional security assessment for some organization, and therefore are not our focus. Since we work in an academic environment with limited scope, several concepts are not applied, such as Rules of Engagement, Overt and Covert testing, and different channels. The choice for the NIST structure was based on the flow already imagined by the tester and it outlined the necessary phases and characteristics of the test. Table 3.1 presents a summary of the methodologies studied and how they are related to our approach.

Methodology	Phases	Characteristics	How it relates to our chosen method
OSSTMM	Regulatory, definitions, information, interactive controls test.	Five channels are considered for testing, such as human, physical and data networks. A set of metrics is used to represent the security state. Several audit types are contemplated, including penetration testing. Different types of errors that may occur during the testing process are identified. Rules of engagement are described. The security tests are grouped by modules. Six types of testing practices are also presented, such as Blind and Tandem.	The only channel we consider is the data networks. Human error is considered as a possibility due to the tester's lack of experience.
NIST-SP800-115	There should be at least three: planning, execution, and post-execution.	NIST describes review and testing techniques. For the latter, it presents and compares different viewpoints that can be employed such as external and internal testing.	The three recommended phases were the ones adapted to the reality of our tests. Review techniques, such as code and configuration review, and testing techniques, such as penetration testing, are included in our approach.
OWASP Guide	Passive mode and active mode.	The guide contains a great number of test cases, including a section on web services. The integration of security testing in the SDLC is a recommended practice. Penetration testing is the main focus of this methodology, but other techniques are also indicated such as code review. Risk prioritization procedures are described and recommended to be included in the testing report.	The information gathering during the passive mode may be seen as part of the planning phase of the NIST-SP800-115, and therefore is included in our method. The guide was extensively used in order to prepare our test cases. Penetration testing is our chosen technique, the same one as in the guide.
SIFT Framework	Threat modeling, scoping, test planning, test execution, and reporting.	The framework presents an extensive list of test cases, many of them focused on web services, grouped in different categories.	The list of test cases influenced our selection, being an important reference throughout the planning of the assessment.

Table 3.1: A summary of methodologies, indicating how they are related to our chosen method

Chapter 4

Preparations

This chapter presents the testing environment, tools and the test cases elaborated for the security assessment. The test cases are based, mainly, on the work by the OWASP Testing Guide [81] and SIFT Framework [85], while paying attention to a number of other references [22, 24, 86, 87, 88, 89], which the reader should refer to for more details.

4.1 Testing Environment

This section describes the elements needed to prepare the testing environment and the Norwegian proof of concept application, which will be used for the security assessment.

In order to establish an environment that would be constantly attacked, and possibly suffer from DoS or be completely compromised, it was decided to make use of virtual machines. One of the main advantages of this approach is the possibility to take snapshots of a functional environment and, if something complicated or irreversible to fix happens, use these snapshots to restore the environment to a previous working state [90]. Another good point is the fact that a dedicated server is not needed for the tests.

A *VMware Server* [91] was installed on a windows host, and Ubuntu-Linux was chosen as a guest system. On the guest, the following applications were configured:

- GlassFish 2.1 [92]: This open source application server, for the Java EE 5 platform, contains the MPOWER services.
- Oracle Database 10g Express Edition [93]: This free alternative database from Oracle is responsible for storing the application's data.

Finally, the MPOWER middleware services were deployed on GlassFish, together with the Norwegian POCA. In this setting, we have the service requestors (represented by the POCA) in the same server as the service providers (i.e., the middleware services), which is not a desired web services scenario but serves our purposes. The figure 4.1 illustrates the expected environment.

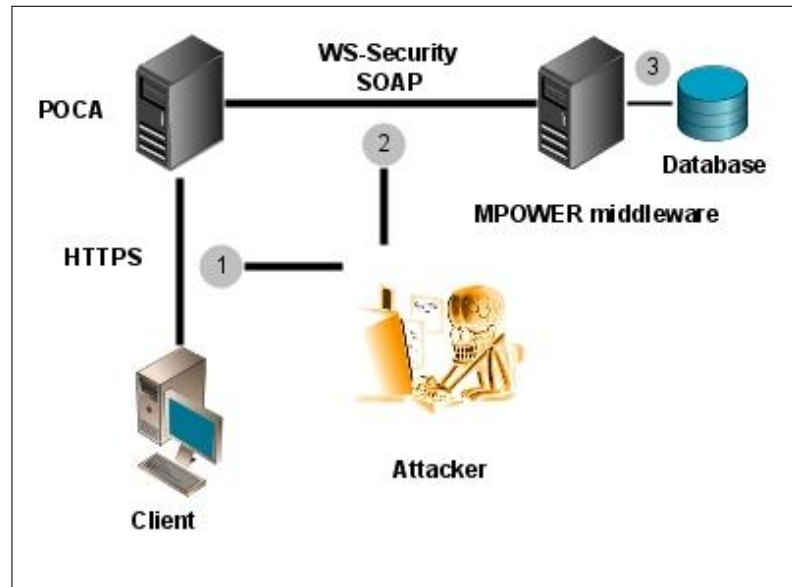


Figure 4.1: Expected testing environment

4.2 Tools

This section focuses on the description of the main tools that will be used for the security assessment. These are:

- **WebScarab** [94]: WebScarab is a framework for analyzing applications that communicate through HTTP and HTTPS. There are a number of plugins available, implementing several modes of operation. The proxy function is the most common used functionality, being able to intercept HTTP and HTTPS traffic, and allowing the operator to review and modify requests and responses that go through. Another possibility is the use of previous requests in replay attacks.
- **Burp Suite** [95]: Burp Suite is a platform for attacking web applications. It has some similarities with WebScarab, including a better proxy functionality, which keeps the history of both requests and responses for further analysis.

- **Wireshark** [96]: Wireshark is a network protocol analyzer, also called a packet sniffer, that can be used to inspect packets that pass by a specified network interface (including the loopback). The user can create filters to match the packets that need to be observed, and save the capture data for later analysis.
- **soapUI** [97]: This tool is specialized for the development and testing of web services. Its features include WSDL inspection, WSDL service invocation, web service functional and load testing. It has a good support for WS-Security, allowing signatures, encryption and security tokens. The main functionality for us is the ability to load a WSDL file and generate standard requests from it, being able to send them afterwards.
- **WSFuzzer** [98]: WSFuzzer has the objective to automate SOAP-based web services penetration testing by dynamically generating several attack vectors, based on the target, to be used for fuzzing¹ parameters.

4.3 Test Cases

Each test case is associated to a vulnerability/threat or security aspect, and is briefly explained in the *Motivation* section.

TC-1: SQL Injection

Objective

Verify SQL injection resistance.

Motivation

SQL injection is a threat that results from poor validation of user input, when performing a dynamic query, one formed by concatenating strings, in an application's database. Such attack can allow access to private data and modify the database in a number of ways, via select, insert, update, and delete operations [81]. The following query exemplifies one attack:

```
SELECT * FROM prescriptions WHERE patientID = ' + $patientID + ';
```

If \$patientID does not contain just a patient's "id" but something in this format:

```
$patientID = 3521'; DROP TABLE patients;
```

The resulting query will be:

¹Fuzz testing or fuzzing, is a testing technique that makes use of unexpected, random, data as input to an application, to verify the application's response to it [99].

```
SELECT * FROM prescriptions WHERE patientID = '3521';  
DROP TABLE patients;
```

In this case, the table *patients* would be deleted from the database².

The presence of characters such as single quote (') and semicolon (;), allied with SQL keywords, not properly handled, constitutes the means for the attack.

Test method

Test the application's queries for known attack vectors, i.e., malicious inputs. One of the basic tests is to use, as input to a query, a single quote, and check if an error is returned. The WSFuzzer tool provides a file with common injections (Appendix A contains an example).

Blind SQL injection is outside of our scope and will not be tested.

Tools

WSFuzzer, Burp Suite.

Requirements tested

Survivability, privacy, integrity, security auditing, authorization.

TC-2: WSDL Scanning

Objective

Identify service's entry points and test them.

Motivation

Since the WSDL contains information on a service's exposed methods and their parameters, by analyzing such a file, an attacker get to know about the inner workings of the service, using this knowledge to perform malicious requests [85]. It is also possible to guess unpublished methods based on the names of the public ones.

Test method

Check how WSDL files are published, obtain a few of them, and analyze their contents. Test the invocation of the operations, with or without the necessary access levels. Verify if there are any hidden operations that should not be available for external (public) use, and assess the outcome of their invocations. A common way to obtain these files is through the service's endpoint, by appending "?wsdl" to it. For example:

```
"http://somepage/service/endpoint?wsdl"
```

²Considering the authorization to do so.

Tools

Wireshark, soapUI.

Requirements tested

Survivability, authentication, authorization, security auditing, privacy, non-repudiation.

TC-3: Replay Attacks

Objective

Verify if replay attacks are effective against the application.

Motivation

By capturing a valid accepted message (e.g., an authentication message), and resending it at a later time, an attacker can pose as a legitimate user, gaining access to sensitive information and performing malicious actions [85].

Other use of replay attacks is to launch a DoS attack, by resending well formed, valid, messages over and over again, in an attempt to deplete the application's resources.

Test method

- Capture valid messages and replay them. Test for several time intervals, checking if the application accepts these messages. Test if freshness protection can be circumvented.
- Resend the same message multiple times, checking for a DoS situation.

Tools

Wireshark, WebScarab, soapUI.

Requirements tested

Survivability, authentication, authorization, privacy, integrity, security auditing.

TC-4: Parameter Tampering

Objective

Verify the application resilience regarding unexpected modification to requests.

Motivation

Improper protection of the communication channels allows an attacker to modify requests, bypassing access control and (client-side) input validation

mechanisms [22], resulting in unauthorized access to information and/or compromise of the application.

Test method

Capture requests, modify and relay them, checking for the application's response afterwards. Modifications include manipulating security tokens, inserting malicious code or just bogus information.

This test case can be used in conjunction with replay attacks.

Tools

WebScarab, Wireshark, soapUI.

Requirements tested

Survivability, integrity, authorization.

TC-5: Forced Browsing

Objective

Verify if the application allows path guessing.

Motivation

An attacker may try to bypass authentication and authorization procedures by typing specific locations in the URL bar of the browser, in order to access different areas of the application [81]. Knowing already some path to an interface, the attacker can also try to guess the location of other interfaces.

Test method

- Select some addresses available to a specific role and try to access them without authentication. Try to log in and log out and try them again.
- Try to access locations from other roles, being authenticated as a user without such roles.

Tools

A browser.

Requirements tested

Authentication, authorization, security auditing, non-repudiation.

TC-6: Buffer Overflow

Objective

Verify the resistance to buffer overflow attacks.

Motivation

Problems in memory management lead to this threat, since it may allow write

and read access to other parts of memory than the ones allocated for the application's input buffers. When these buffers get overloaded, an attacker can execute arbitrary data on the host, by overwriting specific memory locations, or cause a Denial of Service, by crashing the application [22].

Test method

- Input large strings in several forms, verifying if there is a limit to avoid the overflow or if an error is caused due to insufficient buffer size.
- Try to avoid input size validation on client side, by intercepting messages and altering the input's value.

In particular, the objective is to test the DoS scenario.

Tools

WebScarab, WSFuzzer, soapUI, a browser.

Requirements tested

Survivability.

TC-7: XDoS Attacks

Objective

Verify if the application falls victim of XDoS attacks.

Motivation

Besides other DoS attacks caused by buffer overflows and replay attacks, XML Denial of Service (XDoS) can also disrupt the service's availability. Common ways to perform such attacks deal with the exhaustion of system's resources:

1. Oversized payload: Due to high memory consumption to process XML documents [87, 100], it is easy to overload an XML parser by sending large SOAP messages to the server, locking all the resources to a single process. *Document Object Model (DOM)* parsers are more susceptible to this attack, since it loads the complete XML document representation into memory.
2. Recursive payload: By creating a deeply nested XML structure within a SOAP message, an attacker makes the XML parser deplete system's resources since it needs to process and validate each element in the structure [89].
3. SOAP array attacks: It may happen that the server pre-allocates memory for arrays declared in SOAP messages, which can cause an exhaustion of memory resources if the array size is too large [86].

4. Entity attacks: This category is explained in its own test case, TC-8.

Test method

1. Create a large SOAP request such as [81, 85]:

```
<soap:Body>
  <Item ... >large string...</Item>
  ... ..
  <Item ... >large string...</Item>
</soap:Body>
```

2. Generate a deeply nested XML structure inside a SOAP request [85, 87]:

```
<rcrsv >
  <rcrsv >
    <rcrsv >
      ...
```

3. Declare a large array inside a SOAP message:

```
<bigArray xmlns:xs="..."
          xmlns:enc="..."
          enc:arrayType="xs:int[1000000]" >
  <item>blabla</item>
  <item>blabla</item>
  ...
</bigArray>
```

Tools

soapUI.

Requirements tested

Survivability.

TC-8: Entity Attacks

Objective

Verify if the application allows DTD entities.

Motivation

XML allows for the definition of internal and external entities, via Document Type Definition (DTD). When the XML parser encounters them, these entities are replaced by the contents that they reference [86]. Internal entities can be defined as constants or referencing other entities recursively. The latter may cause large documents to arise when fully expanded, since linear inputs can produce exponential expansions [24], which can exhaust system's resources. This attack is also known as XML-Bomb [86].

External entities allows a DTD to include some of its contents from external sources, and can be dangerous if their referenced URIs are not validated by the XML parser. This makes it possible for an attacker to inject references to locations in the target system that contains sensitive information [22], to access limited resources multiple times [24], or to load malicious code from an unknown, untrusted, URL [89]. The last two scenarios can lead to a XDoS.

Test method

Internal entities can be of the form [86]:

```
<!DOCTYPE xmlBomb [
  <!ENTITY bomb "bombbombbombbomb">
  <!ENTITY bomb2 "&bomb; &bomb;" >
  . . . .
  <!ENTITY bomb1000 "&bomb999; &bomb999;">
]>
<node>&bomb1000;</node> <!--invocation of the entity-->
```

External entities can be defined as [81, 24]:

```
<!--private file-->
<!ENTITY accessFile SYSTEM "file:///etc/passwd">
<!--access to resource-->
<!ENTITY accessRsr SYSTEM "file:///dev/random">
<!ENTITY accessURL SYSTEM "http://malicious.com">
```

The SOAP definition [37] specifies that a SOAP message must not contain DTD entities. We should verify if that is the case by sending messages containing these entities.

Tools

soapUI.

Requirements tested

Survivability.

TC-9: Cross Site Scripting (XSS)

Objective

Test for XSS vulnerability.

Motivation

XSS attacks happen when malicious code (e.g., javascript) is injected into a web page hosted at the server side, and are executed at the client side via a web browser [86]. Once more, improper input validation leads to an XSS vulnerability. Cross site scripting is featured as the number one security issue in the *OWASP Top 10 web application vulnerabilities* [101].

Let us consider three types of XSS attacks:

1. Reflected: Also called non-persistent, this attack is possible when a user is convinced to click on a specially crafted URL, having the mischievous script as a GET parameter³. After clicking, the request is sent to the web server, which does not sanitize the parameter, the response is returned to the user, and the script is executed [81]. Ex:

```
http://url.edu/par.jsp?user=<script>alert (:P)</script>
```

2. Persistent: If the web application allows a user to store data, and sanitization is not done properly, it may be possible to inject a script that will be triggered when a user visits the infected page. Ex:

```
<script src="http://localhost/xss-alert.js"/>
```

3. DOM-based: DOM may be used to represent documents in a browser, and enables dynamic scripts to reference elements of such documents. A specially crafted request allied with the poor validation of user input, which is used on the client side for dynamic content under DOM elements, can result in an XSS vulnerability. The malicious code, in this case, does not need to be sent to the server in order to be reflected to the user [81, 102]. Ex:

```
<script>
  document.write(" Location: " +
    document.location.href + ".");
</script>

http://url.edu/par.html#user=
  <script>alert (:P)</script>
```

Everything after the '#' is considered as a fragment, i.e., not part of the query, and is not sent to the server.

Test method

Check for links that make use of GET parameters and choose some of the input forms available in the application, using them to test for harmless scripts (such as a pop up alert) that should be executed if there is an XSS vulnerability.

It is also important to test for character encoding, since an attacker can substitute the normal representation of characters such as '<' and '>' by '%3c' and '%3e', respectively, or by their entities representation '<' and '>'. These are two ways of circumventing input validation. The OWASP Testing Guide [81] explains in more details the problems regarding character

³Earlier we limited our scope to not include social engineering, but this is an important attack vector and a typical example of an attack that happens by tricking a user.

encoding, among other tests. For an extensive list of XSS attacks, please refer to the *XSS Cheat Sheet* [103].

Tools

WSFuzzer, soapUI, a browser.

Requirements tested

Survivability, privacy.

TC-10: XML Injection

Objective

Verify if the application allows injection of XML tags into SOAP requests.

Motivation

This is another flaw caused by improper input validation. The injection of XML elements and attributes effectively change the structure of a document [85], by adding or overwriting nodes, and can also allow XSS attacks.

Test method

Insert special characters into input fields, causing malformed documents. There are a number of metacharacters to try out such as [81]:

- Single quote ('). If the input will be part of an attribute, a possible result would be: `<node att='input'>`.
- Double quote ("). Same case as with single quote.
- Angular parenthesis (< and >). Ex: `<node>>input</node>`.
- Comment tag (<!-- / -->). Ex: `<node>input<!-- </node>`.
- Ampersand (& - used to represent XML entities). It may be used to test indirect XML injection, by using the entity representations of the special characters. Ex: `<` (<), `>` (>), `'` ('), `"` ("), `&` (&).
- CDATA tags (`<![CDATA[/]]>` - contents inside these tags are not parsed by the XML parser). These are usually used when metacharacters need to be considered as text values. Ex:

```
<node><![CDATA[ ]]>></node> /// ']]>' is not parsed.
```

```
<node>
<![CDATA[<script>alert('XSS')</script>]]>
</node>
```

The above code may launch an XSS attack.

If an error is returned that means that the input is not being properly sanitized.

Tools

WebScarab, soapUI, a browser.

Requirements tested

Survivability.

TC-11: Test the username/password authentication scheme**Objective**

Verify if this scheme provides an adequate level of security.

Motivation

Not having a secure authentication procedure, it is easier for an attacker to access users' accounts and private information.

Test method

It is usual to check if an application requires a strong password, one that makes use of upper case and lower case letters, special characters (e.g., !, \$), and numbers. Considering that the target group consists of elderly people, requiring such passwords is just a way to force the user to write it down, not offering much security. Instead, what should be verified is if the login is performed over a secure connection (via SSL/TLS), to avoid packet sniffing, and if the passwords are stored in an encrypted format, preventing unauthorized access to users' credentials.

Another defense mechanism to be tested is the *account lockout*, which prevents brute force password discovery by locking an account after a certain number of failed login attempts with a known username⁴ [81].

Tools

WebScarab, Wireshark, a browser.

Requirements tested

Authentication, privacy, security auditing.

TC-12: Test the transmission of security tokens**Objective**

Verify if the security tokens are being transmitted securely.

⁴We do not consider account lockout attacks, which have the objective to lock valid accounts [81].

Motivation

The security token is issued to a valid authenticated user, and provides information that should be used to prevent unauthorized access to the application's resources. As an example, we present listing 4.1:

```
<securityToken>
  <userID>1</userID>
  <sessionID>agzn4kdKtcDgQAB/AQFmzg==</sessionID>
  <primaryRoleName>Patient </primaryRoleName>
  <authenticationTime>
    2009-05-18T10:23:41.224
  </authenticationTime>
  <serviceIDs>
    CalendarQuery__retrieveExistingActivities_A
  </serviceIDs>
  ...
</securityToken>
```

Listing 4.1: Example of a security token

The tag `<serviceIDs>` identify the service name and the corresponding operation that can be performed by the user. Allowed operations are based on the user's *role* in the system, identified by the `<primaryRoleName>` tag. Each role has its own set of pairs service-operation.

An attacker that manages to acquire a valid security token can execute badly intentioned operations, even though they are legitimate in the application's point of view.

Test method

Sniff the connection between the service requestor (requesting the issuance of the token) and service provider (returning the token), and verify if the token can be distinguished.

Tools

Wireshark.

Requirements tested

Authentication, authorization, privacy.

TC-13: Test the effectiveness of the security token**Objective**

Verify if the security token prevents unauthorized access to the services.

Motivation

As said before, the security token contains the user's permissions on the system, and should be checked before allowing the user to execute an operation. A user that fails to present valid credentials should be denied access to the

application's services. A flawed verification of the token may provide the attacker with means to perform unauthorized operations.

Test method

Remove the security token from the requests, or just modify it, observing the responses received.

Tools

soapUI.

Requirements tested

Authentication, authorization, integrity, security auditing.

TC-14: Session Management

Objective

Verify if the session management is properly handled.

Motivation

The OWASP Testing Guide [81] tells us: *"In order to avoid continuous authentication for each page of a website or service, web applications implement various mechanisms to store and validate credentials for a pre-determined timespan."*

Such mechanisms are responsible for the session management and can be exploited by an attacker to gain access to a user account, even without the necessary credentials. We are interested in the cookie mechanism and how it is configured in the POCA server to interact with the clients. Cookies are briefly explained in the Guide: *"When a user accesses an application which needs to keep track of the actions and identity of that user across multiple requests, a cookie (or more than one) is generated by the server and sent to the client. The client will then send the cookie back to the server in all following connections until the cookie expires or is destroyed"*.

We see that an attacker that gets hold of a valid cookie can impersonate a user and act on his behalf, therefore cookies have to be well protected.

Test method

This test has the objective to verify how easy is to capture a cookie and impersonate a user. Traditional methods start with packet sniffing and XSS attacks. If a cookie is obtained, the next step is to use it (e.g., via a browser) in requests sent to the application, observing if access to the user's account is possible. Important cookie attributes have to be checked to validate if they meet a secure configuration:

- **Secure:** Cookies containing session IDs should be transmitted via secure channels (e.g., HTTPS). The flag ";secure" should be set in such cases, serving as a security advice for the browser, indicating that the cookie should be protected [104].
- **HttpOnly:** Even if some browsers do not support this attribute, it should be set through the flag ";HttpOnly". The purpose is to prevent a client side script from accessing the cookie.
- **Expires:** A cookie can be used until it expires or is destroyed. A session ID cookie should have a short timespan to avoid being used for long time if captured.

Tools

WebScarab, Wireshark, a browser.

Requirements tested

Authentication, authorization, privacy.

TC-15: SOAPAction Spoofing

Objective

Verify the workings of the SOAPAction header.

Motivation

First, let us present some definitions: A SOAPAction HTTP header indicates the intent of a SOAP HTTP request. As specified in the SOAP 1.1 definition [105] and better explained in the WS-I Basic Profile 1.1 [106], this header is required when a client issues a SOAP HTTP request. The WS-I Basic Profile 2.0 [107] clarifies that this header is not necessary for SOAP 1.2, and should not be expected in order to process the message. SOAP 1.2 [37] defines an optional *action* parameter, with similar purposes as the SOAPAction HTTP header, for the "application/soap+xml" content-type header.

Jensen et al. (2007) [87] shows that the HTTP header may be used as the trigger for a web service operation, regardless of the one requested in the SOAP body. The study argues that a SOAP request which tries to invoke a certain operation on a web service may end up invoking a different one, if the SOAPAction HTTP header says so. Furthermore, the header can be modified in any of the intermediary nodes.

Another problem can arise if an application-layer firewall is configured to decide if it accepts or not SOAP requests based on this HTTP header, and the server invokes the operation contained in the SOAP body, ignoring the header, even if it was not supposed to be allowed.

Test method

The message below exemplifies the above mentioned situations and can be adapted to meet our needs. It should be considered tests with and without firewall filtering on the HTTP header, checking if one of the problems occurs.

```
POST /TestService HTTP/1.1
...
SOAPAction: "editRecord"

<Envelope>
  <Body>
    <deleteRecord>
      ...
    </deleteRecord>
  </Body>
</Envelope>
```

Tools

soapUI.

Requirements tested

Authorization.

Remarks

The choice for these test cases was not arbitrary, it were included attacks that target standard web applications as well as SOA-based ones, and specific attacks (considering a web services realization of SOA). Some cases also deal specifically with the MPOWER middleware services, in order to evaluate the security level that they achieved.

The number of test cases is not as comprehensive as it could be; the literature, mentioned at the beginning of this chapter, defines many more cases that should be taken into account in a security assessment. Time constraints and project specifics, such as the ones that will be mentioned in section 5.1, limited the list of cases to a set that could be accomplished and were relevant for this study.

Other threats and test cases not dealt with in this assessment, but of interest nonetheless, include:

- UDDI Poisoning
- Routing Detours

- XPATH Injection
- Schema Poisoning
- Metadata Spoofing (e.g., WSDL Spoofing)
- Transmission of Certificates
- Validation of Audit Logs
- Secure Storage of Sensitive Data

Chapter 5

Assessment Evaluation

In this chapter we present the problems faced before and during the testing phase, relevant information to understand some scenarios, and the report detailing the results obtained after performing the tests, identified in the previous chapter, along with countermeasures.

5.1 Problems

It is very difficult to predict what is going to happen in a project for a person who has little knowledge about it, and that was the case for the researcher. The main issue was the delay in the preparation of the testing environment, which compromised the available window to conduct the assessment. Having less than one month to check the application and its functionalities, test them thoroughly for security flaws, and propose recommendations, was barely enough for the tester.

A second issue is related to the security configuration aspects. The WS-Security standards, presented in section 2.2.2.2, were not put into practice for the POCA and middleware services, preventing the tester from evaluating their effectiveness to deploy a secure SOA Web Services healthcare application. In fact, both links 1 and 2 from figure 4.1 were not secured in any way. No SSL/TLS channels were established for secure communication, allowing packet sniffing and, consequently, several attacks.

The PKI service was also not available for testing and the related test cases had to be dropped. The auditing module was not functional, so no logs were created during the tests, making it impossible to know about attack attempts. It was also not possible to test the administrator functionalities, since the application did not provide the necessary interfaces.

5.2 Roles and Interfaces

In order to understand some scenarios, it is convenient to inform the reader that the Norwegian POCA defines three roles, "patient", "doctor", and "nextOfKin". When a user logs in, he is redirected to the interface correspondent to his role. Each of them have certain permissions on the system, and should not be able to trigger operations not allowed for his role. Figure 5.1 presents the interface for a doctor before choosing a patient, while figure 5.2 shows the interface after a patient was selected. Figure 5.3 corresponds to the patient's interface. The nextOfKin's interface is similar to the doctor's and will not be shown.



Figure 5.1: Doctor's homepage interface (before a patient is selected)

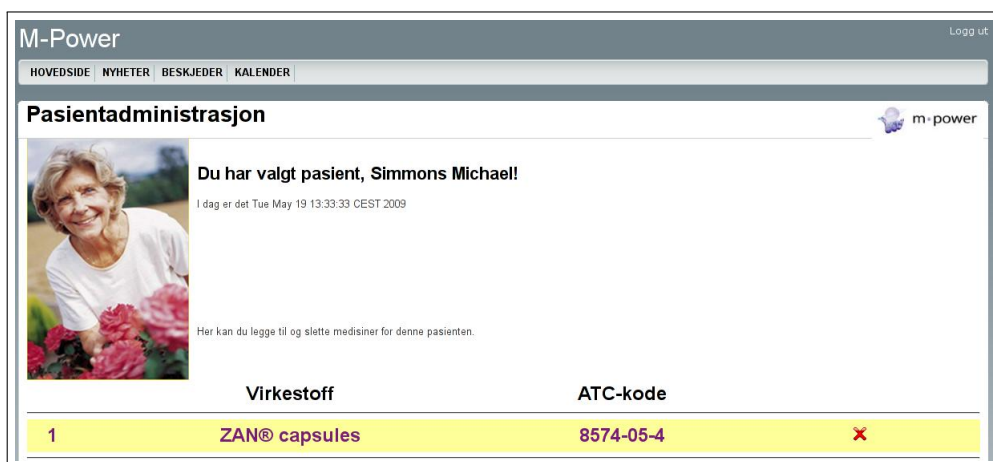


Figure 5.2: Doctor's interface after a patient is selected



Figure 5.3: Patient's interface

One of the main functionalities tested was the one used by doctors to send messages to patients. Figure 5.4 shows the associated interface.



Figure 5.4: Doctor's interface - send message

As the soapUI tool is constantly used, we present in figure 5.5 a screenshot of it, with a SOAP request on the left side and the response on the right.

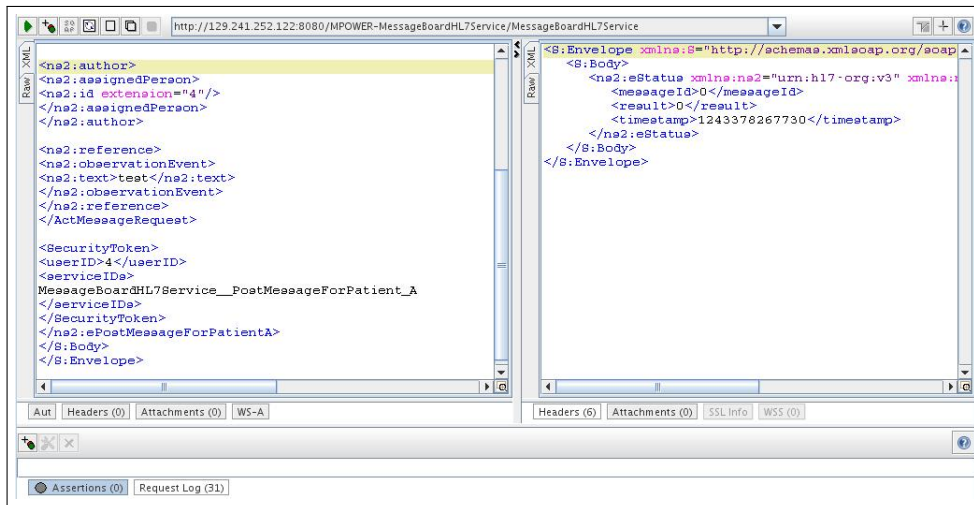


Figure 5.5: The soapUI tool

5.3 Results and Countermeasures

The ideal approach would be to validate the countermeasures in a practical implementation, but time constraints did not allow it, leaving it as a further work to be done.

TCR-1: SQL Injection

Status

Not vulnerable.

Results

This test was performed on requests which, when submitted, resulted in a database operation. The authentication and send message functionalities were the main subjects of test. The attack vectors used are the ones present in the WSFuzzer tool, covering different approaches to detect vulnerabilities. Some examples:

- ' or 1=1 or ''='
- ' or userid like '%
- ';shutdown-

To perform the fuzzing, the WSFuzzer did not behave as expected when loading a WSDL file to generate the attack vectors, and an alternative way had to be employed (appendix A contains relevant information on this method):

1. Create a file "F" with SQL injection inputs (from a WSFuzzer file).
2. Create a SOAP template (via soapUI) for generating the requests, indicating by a "?" marker the parameters that should be substituted by the attack vector.
3. Run a script responsible for parsing "F" and creating a SOAP request, based on the template, for each attack vector in the file.
4. Run another script to launch all the requests generated in the previous step, through a proxy (Burp Suite).
5. Analyze the results.

The analysis demonstrated that no vulnerabilities were found, indicating that the application is immune against the most common types of SQL injection, at least.

Countermeasures

It seemed that the database access was implemented securely, and a code review identified the use of parameterized SQL statements [108] as the method to avoid these attacks. This method makes use of "bound" parameters, which considers the whole user input as the parameter for the SQL command being triggered. In practice, the following occurs:

- `SELECT * FROM users WHERE username = ?; // '?' is the the bound parameter`
- `? => ' or userid like '% // the attacker's input`
- `"There is no user with username [' or userid like '%]" //message returned from the authentication operation`

Another popular way to prevent SQL injection is through input validation, which is discussed in the results for XSS (TCR-9).

TCR-2: WSDL Scanning

Status

Vulnerable.

Results

The WSDL files are not published in a UDDI directory, but can be retrieved from the server. The path to the files is not trivial if the attacker does not know the system, but if he is able to sniff on messages sent between service requestor and provider, he can easily obtain such information since he can

observe the service's endpoint. In figure 4.1, such scenario would happen on link 2.

Possessing the WSDL, it is possible to craft messages to be sent to the application. It was identified that some operations can be called in two different ways, depending on their name suffixes. For example, the operation "ePostMessageForPatient_A" indicates that authorization is needed for the request to succeed, while "ePostMessageForPatient" is an operation that can be executed without authorization. The normal flow can be seen below:

1. The request is triggered targeting the "_A" version;
2. The "_A" operation perform the authentication and requests the other version to continue the processing;
3. This second operation is responsible for the non-security related logic and effectively completes the request.

This implementation opens a big security hole in the system, letting a malicious user perform requests without any verification.

Countermeasures

First of all, the (internal) operations that do not require authentication should be removed from the WSDL. Even then, a smart attacker could still guess the name of these operation by just removing the "_A" suffix. The most appropriate alternative is to have just one version of each operation, the one that requires authorization. Another option is to restrict access to the internal operations via an XML firewall [87].

In our case, the architecture considered is not really complex, presenting only two web services nodes, one service requestor and one service provider, in a typical client/server configuration. This might permit the use of SSL/TLS communication alone, since there are no other intermediaries in the flow, and the SOAP messages are sent just between these two nodes. This approach would prevent the attacker from discovering the service's endpoint, since the HTTP headers would be encrypted. In a scenario where the middleware services are distributed in two or more servers, each dealing with different sensitive information, or are accessed by other services in different trust domains, data can go through untrusted intermediaries. In this situation, the added use of message-layer security is fundamental to safeguard the data transmitted, as already pointed out in section 2.2.1.1.

TCR-3: Replay Attacks

Status

Vulnerable.

Results

Considering the figure 4.1, without a confidentiality mechanism it was possible to capture clear text HTTP requests on link 1, via WebScarab, and SOAP requests on link 2, via Wireshark. For the replayed HTTP messages to be accepted, it was just necessary to keep the same cookie, which is set by the server and has a JSESSIONID value as the session identifier. Lacking integrity protection, GET and POST parameters could be modified without problems. After the user logged out, the replayed messages were not accepted anymore, but it was possible to replay the authentication message and get access to the system again.

The SOAP messages were easier accepted, since they access the web services directly. These messages could be replayed at any time along with parameter modification, via soapUI. Issues regarding the security token will be evaluated in its respective results sections (TCR-12 and TCR-13).

As expected, a DoS was possible by sending hundreds of messages to the server in a short interval.

Countermeasures

The use of SSL/TLS on link 1 would guarantee the confidentiality as well as integrity of the HTTP messages, and at the same time prevent replay attacks [27], while WS-Security mechanisms (Timestamp, XML Encryption, XML Signature) would protect the SOAP messages on link 2. An example of a timestamp is given next:

```
<wsse:Security>
  <wsu:Timestamp wsu:Id="timestamp">
    <wsu:Created>2009-06-11T08:55:00Z</wsu:Created>
    <wsu:Expires>2009-06-11T09:00:00Z</wsu:Expires>
  </wsu:Timestamp>
  ...
</wsse:Security>
```

Listing 5.1: A WS-Security timestamp element

The WS-Security specification [50] tells us that the obvious measure is to sign the timestamp element in order to prevent its modification, and to cache recent timestamps to be able to detect a replay; on top of that, messages with a timestamp older than a certain period of time should be rejected. However, the specification does not deal with the problem of clock synchronization, which is important to verify the expiration time set by the service requestor.

The security token in use by the MPOWER platform has an element that could be useful, to a lesser extent, in avoiding replay attacks, the "sessionID"

(created at the beginning of the authenticated session). It is natural that a session lasts longer than the transmission of a message, so the freshness control should not rely only on this element, since a message could be replayed during the whole time the session is active. It could be useful to avoid replay of messages that contain expired session IDs, though.

Another possibility to help prevent replay attacks is the use of message IDs, part of the WS-Addressing standard [109].

TCR-4: Parameter Tampering

Status

Vulnerable.

Results

Considering the figure 4.1, without confidentiality and integrity protection it was possible to use a proxy to intercept HTTP requests on link 1 and modify GET and POST parameters, before relaying them to the service provider.

It was not possible to do the same with SOAP messages, so the workaround was to sniff the requests on link 2, load them in soapUI, and send the modified messages, in a similar way as the approach for replay attacks. The lack of WS-Security (XML Signature) made it possible to tamper with parameters as there is no control for message integrity.

Countermeasures

Once more, the correct deployment of SSL/TLS and WS-Security would have prevented this type of attack, as previously discussed.

TCR-5: Forced Browsing

Status

Vulnerable.

Results

The first attempt to access the application assumed that the attacker did not have access to any valid account, but somehow knew some interfaces' addresses. The tests demonstrated that just one page was readily accessible, the one responsible for showing the patient's activities ("wholeday.jsp"), but no information was available to be seen. Furthermore, the virtual directories for resources such as javascripts, flash and images, could also be accessed.

The next step consisted in trying to access other parts of the system that the user logged in did not have permission to. It was possible, for a nextOfKin

and a patient, to access the doctor's interface and execute requests such as select patient, prescribe/delete drugs, send/delete messages, and post/delete activities on the calendar (as a matter of fact, figure 5.1 shows the patient Michael Simons browsing the doctor's homepage). A not so serious result showed us that a doctor could access the patient's interface and perform any operation as if he was himself a patient. Below we see the URL's of each interface:

```
http://129.241.252.121:8080/NorwegianPOCA/family/index.jsp //nextOfKin
http://129.241.252.121:8080/NorwegianPOCA/simple/index.jsp //patient
http://129.241.252.121:8080/NorwegianPOCA/doctor/index.jsp //doctor
```

One would assume that the RBAC scheme, mentioned in section 2.3.2.1, is not really effective, but performing an analysis of the roles' configuration in the database it was discovered that the users had indeed permission to execute all the operations mentioned. Still, the context based access control is not in action, since the doctor's homepage presents all the registered patients to any authenticated user when just the patients related to this particular doctor should be visible.

Reviewing the application's code, we found out that access to most of the web pages is controlled by verifying if a session attribute is set, as shown below:

```
if (session.getValue("MM_Username") != null &&
    !session.getValue("MM_Username").equals("")) {
//tester's comment: the next condition does not add
//anything to the access control
    if (true || (session.getValue("MM_UserAuthorization")==="") ||
        (MM_authorizedUsers.indexOf((String)session.
            getValue("MM_UserAuthorization")) >=0)) {
        MM_grantAccess = true;
    }
}
```

Since all users have this attribute ("MM_Username") set after they log in, they can access whichever pages they want, as long as the path is known.

Countermeasures

Obviously, the correct path to the desired interface has to be known for the attack to succeed, what may not be difficult to figure out for a skilled attacker if the alias¹ is too simple. For example, the alias for the doctor interface is "doctor", quite simple to guess.

It is extremely important to revise the system's configuration, specifically regarding roles' permissions, to prevent unauthorized access to interfaces

¹A name that an entity uses in place of its real name [13]. In our case, is the mapping to the interface.

and operations. The POCA should not deal with the access control, leaving this task for the middleware services. For example, the authorization service would check the security token of a patient and, therefore, not grant him access to the doctor's homepage, and neither authorization to prescribe drugs to other patients or to himself. To control access to interfaces, the roles should be related to a set of them and the security token might contain such a list. A simplified example is shown below:

```
<securityToken>
  ...
  <interfaces>userInterface</interfaces>
  <interfaces>userViewMessages</interfaces>
  <interfaces>userViewCalendar</interfaces>
  ...
</securityToken>
```

```
if ( authorizationService.isAuthorizedInterface(secToken ,
      "doctorInterface")){
  MM_grantAccess = true;
}
```

Another measure is to consider the context in which a user is requesting a service, to prevent that a nextOfKin "N", related only to a patient "PA", interacts with a patient "PB", or that a doctor manages another doctor's patients.

The virtual directories should also be protected from external access.

TCR-6: Buffer Overflow

Status

Not vulnerable.

Results

The several attempt to input large strings into the application, via the interface, parameter tampering or direct SOAP requests, did not cause any special problems. What happens is that the Oracle database returns an error indicating that the operation could not be completed, but no DoS or other out of ordinary events occur. The maximum input length tested was of 29164162 characters.

It is important to mention that this was a simple test, not involving tools and more refined methods as the ones specified in the OWASP Testing Guide [81].

Countermeasures

Good coding practice. Just as a reminder, languages such as Java and C#

offer protections against buffer overflows, since they tightly control data boundaries [110], while other languages like C or C++ have direct access to memory, which make them vulnerable. Even though a language is safer, the risk still exists and input validation should be considered [86].

TCR-7: XDoS Attacks

Status

Not vulnerable.

Results

All attack methods in this category failed and did not produce any XDoS situation. The following list details the inputs used for testing:

1. Oversized payload: This scenario was already tested in the buffer overflow test case.
2. Recursive payload: A nested structure of 100000 elements was sent, and the request was completed successfully.
3. SOAP array attacks: An array of size 1000000 was declared and sent in the message; the operation was executed without problems.

Countermeasures

It seems that the XML parser at the server responds well to these attacks, but if it was the opposite, *Schema Validation* could be a solution, by not allowing messages that do not comply with the defined schema [87]. Schema validation is also the basis for *Schema Hardening*, which is a method to limit the memory used to process messages. One drawback of using schema validation is that it is resource intensive and can lead to performance issues [22].

In this test case, it is important to point out that the use of encryption can obfuscate these attacks and, in order to avoid such scenario, it is necessary to decrypt the message before validation takes place [87].

TCR-8: Entity Attacks

Status

Not vulnerable.

Results

As simple as it may sound, the XML parser of GlassFish was configured to silently ignore DTD declarations, generating an error indicating that the entity present in the SOAP body of our request was not declared.

Countermeasures

The XML parser of the application server is configured correctly to prevent this attack.

TCR-9: Cross Site Scripting (XSS)

Status

Vulnerable.

Results

Of the three cases considered, the application is vulnerable to two of them:

1. Reflected: As seen in figure 5.6, the parameter "PatientName", that is used to print the name of the selected patient on the screen, was injected with a script.

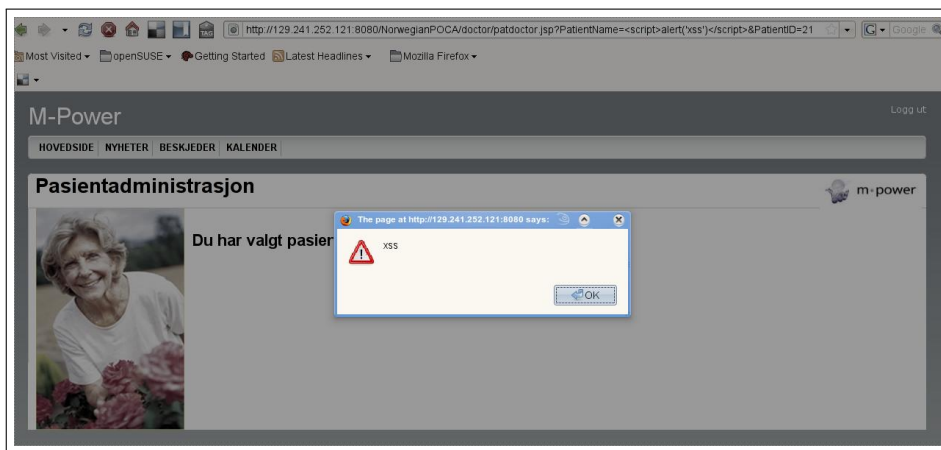


Figure 5.6: Doctor as a victim of XSS

Below we see a valid HTTP request (#1) turned into malicious ones (#2 and #3):

```
1) /doctor/patdoctor.jsp?PatientName=Simmons%20Michael&PatientID=21
2) /doctor/patdoctor.jsp?PatientName=<script>alert('xss')</script>
   &PatientID=21
3) /doctor/patdoctor.jsp?PatientName=<IMG SRC='javascript:
   alert(document.cookie)'>&PatientID=21
```

2. Stored: It was possible to insert into the database the same kinds of scripts used for the reflected attack, via the send message interface and its direct SOAP request. The latter was possible by substituting the character representations of "<" and ">" for their entities equivalent, and is better explained in the results for XML injection. Other forms of scripts can also be injected, but may depend on which browser is

being used to be successful, and we will not focus on this aspect.

Countermeasures

As we can see, input validation is not being properly handled. These special characters and tags should be filtered at the service provider side, before processing any business logic, to avoid being injected. The POCA (service requestor side) should also validate parameters that are just used locally (e.g., to show the name of the patient). Regular expressions are a common way to restrict the input received, rejecting anything that does not match pre-defined patterns, also paying attention to encoding and entities, as mentioned in the test case. As an example, consider the following regular expression (it rejects strings that starts with any number of white-space characters, followed by any number of letters and numbers, and ending with "<" or "<"):

```
" ^ ( ( ( \ s * \ w * ) ( < | & l t ; ) ) ) $ "
```

Shanmugam et al. [111] presents a data flow designed to prevent XSS attacks, based on a SOA environment, making use of schema validation and regular expressions. A more general approach is proposed by Steel et al. [112], called *Intercepting Validator*, also based on regular expressions. Both solutions define for each kind of request a set of rules to be used on the validations, since different requests deal with different data types.

There are several other references that deal with input validation, such as the *OWASP Validation Documentation* [113], and they should be evaluated to check which approach would bring the best results.

TCR-10: XML Injection

Status

Vulnerable.

Results

The system is vulnerable to three types of (stored) XML injection:

- Comment tag: Using the tag "<!--" as input, via the interface, it is possible to comment parts of the HTML page returned. After sending the tag at least twice², the page rendering such inputs have sections, which are contained between two occurrences of the tag, commented out.
- Ampersand: An input such as <!- - does not cause problems via the interface; since the browser encodes both "&" and ";" characters,

²That restriction was verified for Firefox (3.0.10), while for Opera (9.64) just one occurrence of the tag was enough.

the XML parser does not recognize anymore the entity "<". A SOAP direct request has the same effects as the comment tag injection above.

Let us now consider the next inputs:

```
&lt;script&gt;alert('xss')&lt;/script&gt;
&lt;script>alert('xss')&lt;/script>
```

In this case it was possible to perform an XSS injection, since the XML parser converts the entities to their character representations and the script is stored in the database. Testing such inputs through the interface was harmless, due to character encoding performed by the browser.

- CDATA tags: Another attack that just worked via a SOAP direct request, again causing an XSS injection. The input string for this case was:

```
<![CDATA[<]]>script <![CDATA[>]]>alert('xss') <![CDATA[<]]>/script
<![CDATA[>]]>
```

Countermeasures

The same type of input validation applied for XSS attacks should be employed to prevent XML injection, with the addition of comment and CDATA tags.

TCR-11: Test the username/password authentication scheme

Status

Insecure.

Results

The first problem is related to the non use of SSL/TLS on link 1 in figure 4.1. The login request can be easily intercepted and checked for the user's credentials. Even if this channel was secure, there should be another confidentiality protection on link 2, since an attacker can also eavesdrop on the SOAP messages being sent from service requestor to service provider. In this way, it is possible to impersonate users and get access to private information without much effort.

A second problem arises from the fact that the users' passwords are stored in cleartext in the database. Any person with access to the database can obtain this confidential information.

As expected, the authentication service allows very simple passwords, which

could be efficiently discovered via dictionary attacks³ if the username is known, but the account lockout functionality prevents this by locking the user's account after three wrong guesses.

An attacker can find out if a username is valid or not due to the message contained in the SOAP response to a login request:

- If the username exists and the password is wrong, an exception is returned.
- If the username does not exist, the message indicates so.

Countermeasures

As already mentioned, for the first issue, use of SSL/TLS allied with WS-Security should offer a reasonable level of confidentiality for the authentication procedure, preventing eavesdropping.

Regarding the second issue, a secure approach is to hash the passwords together with a random value (salt). Just by hashing the password, it is already not trivial to discover the original value by looking at its digest. Consider the attack where pre-computed hash tables (Rainbow Tables), based on large dictionaries, are used to compare their values with the target hashes in order to find matches that would reveal the passwords [15]. After appending the salt to the original password, the hash of the concatenated string should be completely different and the effectiveness of the attack will greatly drop. Another advantage of the salt is that it prevents duplicate passwords from being distinguished [14], since the hash of "*pass+salt1*" will be different from "*pass+salt2*". The table 5.1 illustrates the concept.

Password	Salt	Hash>Password)	Hash>Password+Salt)
lancaster	Gb?@	ea2f71710b0082b	b23b8477c901d0f
lancaster	H711	ea2f71710b0082b	d589296de65805a

Table 5.1: Use of salted hash on passwords

Finally, the message returned after a login attempt should be more generic, such as "There is no user X or the password does not match", to avoid giving too much information.

TCR-12: Test the transmission of security tokens

³Dictionary attacks are brute force attacks based on trying every word contained in a large list [13]. Such lists may contain any kind of strings, such as common names or birthday dates.

Status

Insecure.

Results

As already mentioned, the non use of security mechanisms such as SSL/TLS and WS-Security makes the application and middleware services really vulnerable and it is not different for the transmission of the security token. An attacker can easily sniff on the SOAP response carrying the token and use it in his own requests, impersonating an authenticated and authorized user. Even if the service provider just offers operations that need authentication (operations "_A", see TCR-2), with the security token the attacker will have access to all the resources allowed for the legitimate user.

Countermeasures

Use of SSL/TLS and WS-Security to protect the token's confidentiality.

TCR-13: Test the effectiveness of the security token

Status

Insecure.

Results

In an ideal and unrealistic situation, the security token works relatively well, it prevents unauthorized users from triggering operations of type "_A". Still, the tests showed that some of its elements were not checked to perform the authorization and could be removed, such as "sessionID", "authentication-Time", and "primaryRoleName", only being necessary the "userID" and the "serviceIDs" correspondent to the requested operation. The lack of integrity protection permits that an attacker tries to craft valid "serviceIDs" elements, using them to request other operations not available in the security token.

Countermeasures

Integrity protection is the appropriate way to guarantee that the security token delivers what it expected from it, access to authorized services only. Like in most cases, the mechanisms provided by SSL/TLS and WS-Security are essential here. Important to mention again that the security token is not checked for operations that do not need authorization (see TCR-2).

TCR-14: Session Management

Status

Insecure.

Results

The session management at the POCA server is poorly implemented, and we saw that basic security measures are not in practice, thus it is easy to perform a session hijacking in which the attacker "steals" the user session [81]⁴. The necessary steps for the attack are outlined below:

1. User 'U' requests the login page.
2. Server creates a session and sends back a cookie (JSESSIONID) to be set on the client browser.
3. 'U' logs in via username and password (browser sends the cookie to the server), is authenticated, the security token is created and it becomes a session attribute, stored at the POCA server.
4. The attacker 'A' could sniff the communication at any point, waiting for a cookie transmission, or even retrieve one through a successful XSS attack⁵. Having accomplished his objective, he sets the same cookie in his browser and requests one of the pages accessible by 'U'.
5. The attacker's cookie presents a valid session ID, which is used to retrieve the associated security token. In this way, he is identified and authenticated as 'U', being granted access to the requested resource.
6. Now, 'A' can perform any operation authorized for 'U'.

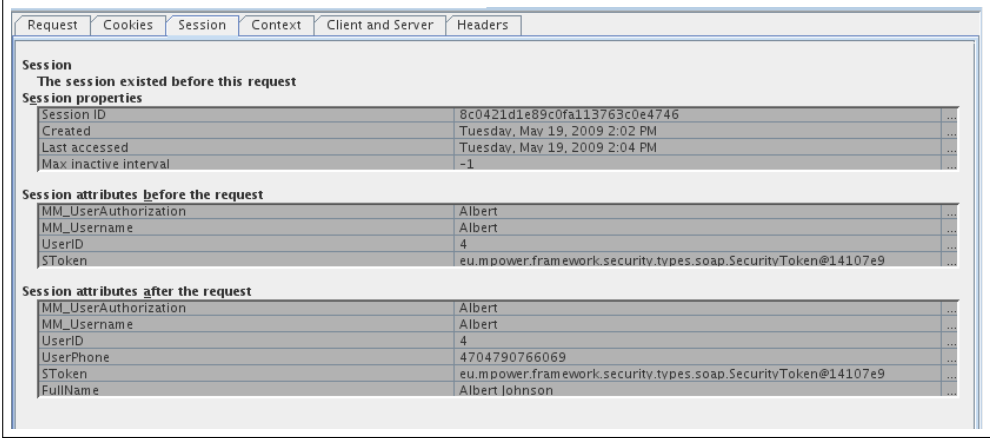
It was observed that the cookie does not have its expiration date set and therefore an attacker that obtains such a cookie could access the application at any time, as long as the user does not log out. The attacker cannot perform any other actions if the user logs out, indicating that the session is not valid anymore. In fact, just the session attribute responsible for the access control (see TCR-5) is reset, as we can see next:

```
if (request.getParameter("MMLogoutnow") != null &&
    request.getParameter("MMLogoutnow").equals("1")) {
    session.putValue("MM-Username", "");
    session.putValue("MM-UserAuthorization", "");
    ....
}
```

Figure 5.7 shows the session attribute "MaxInactiveInterval" set to "-1", meaning that the session does not expire, and could be accessed again if the user still has the associated cookie. In this case, the attacker could reuse the cookie already obtained.

⁴We say steal, but the legitimate user does not have his session interrupted, still experiencing normal access. Such an attack is called session "sidejacking" by Robert Graham of Errata Security [114], and happens when an attacker obtains a session cookie sent over insecure channels and is able to impersonate the user [115].

⁵In reality, the attacker could have gotten the user's credentials, as previously discussed, but the objective here is to demonstrate another kind of attack.



Session	
The session existed before this request	
Session properties	
Session ID	8c0421d1e89c0fa113763c0e4746
Created	Tuesday, May 19, 2009 2:02 PM
Last accessed	Tuesday, May 19, 2009 2:04 PM
Max inactive interval	-1
Session attributes before the request	
MM_UserAuthorization	Albert
MM_Username	Albert
UserID	4
SToken	eu.mpower.framework.security.types.soap.SecurityToken@14107e9
Session attributes after the request	
MM_UserAuthorization	Albert
MM_Username	Albert
UserID	4
UserPhone	4704790766069
SToken	eu.mpower.framework.security.types.soap.SecurityToken@14107e9
FullName	Albert Johnson

Figure 5.7: Session does not expire

The dangers of an attacker getting hold of a valid cookie are even higher since it is possible to access all the interfaces defined for the application (see TCR-5).

Countermeasures

SSL/TLS provides the means to prevent an attacker from stealing a session cookie, by encrypting the data transmitted on link 1 (figure 4.1). Furthermore, the session attributes identified in TC-14 should be set accordingly to avoid a cookie from not expiring, from being compromised via XSS attacks, and from being sent over insecure channels. If the risk of having the cookie stolen is acceptable (e.g., using HTTPS for the login and HTTP afterwards), the application server should sign it to guarantee its integrity.

To avoid the reuse of a cookie by an attacker, the session at the POCA server should be destroyed after a user logs out, besides expiring after a certain time of inactivity.

The tester is aware that users of the application may leave it connected for a long time and could be bothered if their sessions are interrupted periodically, so the best option would be to secure the transmission of the cookie, so it does not end up in malicious hands in the first place.

TCR-15: SOAPAction Spoofing

Status

N/A.

Results

It was observed that the operations triggered on the web services depend

only on the request in the SOAP body, no matter what the SOAPAction header indicates.

It was not possible to configure an application-layer firewall to perform the second part of the tests.

Countermeasures

As Jensen et al. (2007) [87] proposes, the SOAP body should be inspected to verify which operation is being requested and it should be compared to the one indicated in the SOAPAction header. Any discrepancy may indicate an attack and the request should be dropped.

Chapter 6

Recommendations

The aim of this chapter is to provide recommendations that the tester considers useful for tightening the security of the MPOWER platform and the POCA. These recommendations can also be applied to other SOA-based (healthcare) systems for improving their security.

6.1 Summary of Risks and Security Review

Table 6.1 presents a summary of the issues that need to be tackled to make sure the middleware is more secure and can be utilized in a secure manner by any healthcare application, based on the results obtained in section 5.3.

Test Case	Results
TC-2: WSDL Scanning	Operations that do not need authorization can be invoked directly via SOAP messages.
TC-3: Replay Attacks	Any kind of replay attack is possible, with or without modification, at any time.
TC-4: Parameter Tampering	Requests can be modified on the fly, without detection.
TC-5: Forced Browsing	Users can access interfaces and functionalities related to roles they do not have.
TC-9: Cross Site Scripting (XSS)	It is possible to recover session cookies and then hijack users' sessions.
TC-10: XML Injection	Can trigger XSS attacks.
TC-11: Test the username/password authentication scheme	Login credentials are transmitted over insecure channels and passwords are stored in cleartext in the database.
TC-12: Test the transmission of security tokens TC-13 Test the effectiveness of the security token	The tokens may be captured by an attacker wishing to send direct SOAP requests, and can also be modified in an attempt to trigger other operations not indicated.
TC-14: Session Management	It is possible to steal session cookies and impersonate users for long periods of time.

Table 6.1: Summary of risks

The initial idea was to prepare a risk prioritization list based on the OWASP Testing Guide's method, presented in section 3.4.3, but in the tester's perception all the security issues are easy to exploit and can cause great impact on the confidentiality and integrity of data.

Adoption of security schemes, such as SSL/TLS and WS-Security, should take into consideration what is it really expected from a SOA realization. Is it going to be used in a client/server approach or as a true interaction between distributed services? If the goal is to achieve communication across boundaries, as the paradigm proposes, WS-Security should be part of the solution design and should be configured for the services, enabling message-layer security (end-to-end). Otherwise, there is no need to worry about all the standards under WSS, being enough the use of SSL/TLS alone. For the remainder of the chapter, we consider the SOA standard view.

Considering the MPOWER components listed in table 2.3, we can say that most of them could not be evaluated because they were not active, such as the PKI service, Audit, Encryption, Secure Storage and Secure Communication. The Audit service could provide the means for detecting attack attempts, while the other four could help in the data protection, by safeguarding data in transit and data at rest. Administrator operations were not assessed as well, as mentioned in section 5.1. The following recommendations result from the observations and countermeasures in chapter 5.

As we have seen in section 2.2.2.2, the standards XML Encryption and XML Signature are the basis of the necessary protection for the SOAP messages, if applied correctly, preventing attacks such as parameter tampering and session hijacking. The PKI module of the MPOWER platform could be the central piece of this scheme, as it provides the means for public key cryptography, essential for session (symmetric) key exchange and digital signatures.

Regarding the Token Management service, the custom security token designed for the MPOWER platform could be seen as an SSO solution, in the same way as SAML assertions, as long as it is signed by a trusted party. The signature would guarantee that the token is trustworthy, as well as protect its integrity. We have seen in the TCR-12 that confidentiality is extremely important for the token as well, since an eavesdropper could use it for his particular requests. In a scenario where the middleware services are accessed from an external organization, WS-Trust could be implemented in order to generate the necessary security tokens.

The Access Control module would definitely be more successful if the configuration of roles were more restrictive and the context of the requests were considered. There is no point in allowing a patient to perform doctor's ac-

tivities, or a nextOfKin to observe and interact with unrelated patients. The POCA should take advantage of the middleware services and base its access control logic on them, avoiding such a weak verification of session attributes as shown in the TCR-5.

For every operation that deals with parameters, an input validator mechanism should be triggered before any other business logic, to avoid injection attacks. Such defense is important in both POCA and middleware services, as noted in the TCR-9.

Message freshness identifier is an element useful for preventing replay attacks, and can be ensured via WS-Security through its Timestamp node, as we have seen in the TCR-3.

The MPOWER Project Deliverable D5.2 [67] contains highly useful recommendations for the security of the platform, some of which were contemplated as countermeasures for the issues found in the assessment, and should be reviewed.

6.2 Configuration Aspects

When configuring SSL/TLS on the application servers (e.g., via the GlassFish Admin Console), the best alternative would be to consider the most recent standard, which is the TLS 1.2 [27] at the time of this writing. In this way, from the available cipher suites¹, on the servers, the ones that provide better security should be chosen².

Usually, just the server authenticates itself to the client when a secure channel is established, but in one-way authentications there is a risk of a man-in-the-middle attack, since the party providing its certificate cannot verify the identity of the other party. The most secure approach would be to require a mutual authentication, where both parties offer proof of their identity to each other, by means of public key certificates, signed by a trusted party (e.g., an MPOWER CA). This method would prevent the execution of requests coming from untrusted nodes. In this setting, the MPOWER users and servers should have access to certificates provided by the PKI service.

Configuration of WSS may be facilitated via the Netbeans IDE , which integrates well with the GlassFish application server and was the chosen platform for the development of the middleware services, being part of the MPOWER

¹Cipher suites are negotiated between client and server, and specify which cryptographic algorithms will be used in the secure connection.

²TLS 1.2 removed support for DES, for example, since it is considered an insecure cipher. The standard still supports MD5, but this hash function is also considered insecure.

Tool Chain³ [58]. Online references, such as the JAX-WS Reference Implementation Project [116], offer step-by-step tutorials to configure the security for the service requestor and service provider, using Netbeans.

6.3 Distribution Model

Figure 6.1 exemplifies a possible distribution of MPOWER servers over the internet, and how they are accessed, even by external organizations. We see that each server has a user interface (UIF) and a service interface (SIF). A UIF is necessary for the users to interact with the services, and may be represented by a web portal, while a SIF represents the machine-to-machine interface, used by services to communicate with each other.

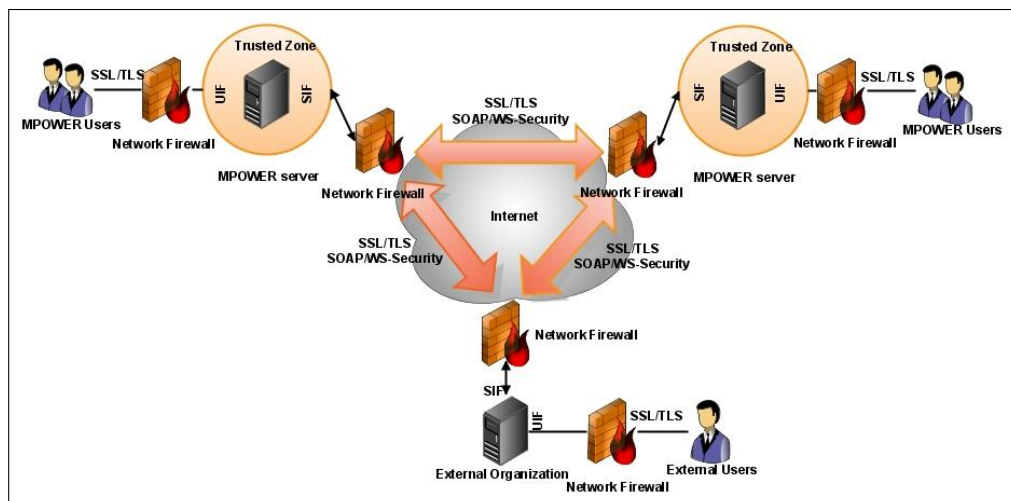


Figure 6.1: MPOWER distribution model. Figure adapted from the MPOWER Project Deliverable D5.2 [67].

We can see that requests can go through untrusted zones in the scenario shown in figure 6.1, justifying the use of message-layer security, while other issues discussed earlier corroborates the use of SSL/TLS. Such architecture may represent other SOA systems as well, which may benefit from following the advices presented.

Besides the already mentioned SSL/TLS and WS-Security, network firewalls are in place to prevent attacks on other server applications that should not be reached from external networks, such as ftp or smtp, reducing the attack surface of the system [67]. These firewalls could also restrict access to the

³The set of tools necessary for development, deployment and testing.

middleware services, via access control lists (ACL), only allowing requests from pre-defined network addresses or subnets. A remark though, the use of IP based authentication, alone, is vulnerable to spoofing attacks [14].

The deployment of XML firewalls (or application layer firewalls) could add another level of protection, by inspecting the contents of incoming and outgoing SOAP messages. A number of policies can be configured to prevent several kinds of threats [30, 88]. Execution of a service's internal operations can also be avoided, as already mentioned in the TCR-2.

6.4 Final Comments

By applying the recommendations contained in this chapter, the overall security for both the middleware services and the POCA is expected to increase, but it is important to remind the reader that security should be part of the whole software development life cycle and should not be overlooked. Testing for security during the development may reveal vulnerabilities earlier in the process and reduce costs related to mitigation. Documenting the security related decisions, including configuration details, is another way to check for issues and verify if the implementation and deployment satisfy the requirements.

To achieve the level of security required for healthcare applications, a lot of effort has to be put into the development of secure components and their integration. It does not matter if one component provides an acceptable level of security but it is misused. Several scenarios have to be thought of and tested, to guarantee that private data is protected at all times. It is normal to say that *"security is only as strong as its weakest link"*.

Chapter 7

Discussion

This chapter has the objective to reflect about some aspects of the work conducted in this thesis.

What have we achieved?

The assessment proved that the POCA and the MPOWER platform are vulnerable to common attacks targeting web applications. Considering the *OWASP Top 10 web application vulnerabilities* [101], seven of them are present in our case study system:

1. Cross Site Scripting (XSS)
2. Injection Flaws
3. Information Leakage and Improper Error Handling
4. Broken Authentication and Session Management
5. Insecure Cryptographic Storage
6. Insecure Communications
7. Failure to Restrict URL Access

Based on our observations, we can infer that SOA-based systems in general are expected to suffer from the same problems. While this is not surprising, the fact that an organization that is concerned with data confidentiality and integrity does not implement basic security mechanisms, makes us think how many other similar cases are there, completely vulnerable.

Even though we evaluated a healthcare system, we can extrapolate the results to other domains since the vulnerabilities found are not specific. Therefore, the findings in this report are relevant when considering the development of

secure applications, based on SOA or not. Problems related to disclosure of personal information will cause a loss of users' trust and make the organization(s) behind the applications liable to lawsuits, no matter if it happens in the healthcare domain or any other.

This report can be considered as an alert at the same time as a high level guide to test for security issues and fix them.

What could have been done differently?

In the beginning, it was thought that the system to be tested would be more secure, having implemented basic security measures that could prevent some kinds of attacks. The figure 4.1 presented the expected testing environment, indicating what the tester imagined it to be. If this were the reality faced during the tests, different test cases would provide us with a better idea about the security level achieved, e.g., testing for weak ciphers in the SSL/TLS suite, evaluating message-layer security, checking the validity of digital certificates and verifying if forgery would be possible.

Unfortunately, there was no time to configure the environment in the desired way, and such tests will have to be performed in a further occasion.

Was our method an appropriate one?

Naturally, the decision to exclude or include test cases could just be made after understanding better the system and its defenses, what bring us to the methodology aspect of our assessment. As shown in section 3.6, the planning phase of our approach contemplates the selection of test cases and incremental changes to the resulting list. In general, the adopted method for performing the tests was a tentative to offer a less complex solution based on complicated, and extensive, methodologies and guidelines. It is normal that, in this process, some details are left out and therefore may cause some sort of impact on the end result. We have to verify if such a simplification is possible and if is worth it or not.

Still, the method may be modified in any way a tester wants, or needs, in order to provide the desired results. This characteristic gives flexibility but is also a disadvantage, since there is a need to reevaluate the standard methodologies in order to include additional controls if the assessment changes focus. The need for a more comprehensive methodology, or the adoption of just certain parts of different methodologies, may depend on the conditions of the assessment. In our case we had access to a controlled environment specifically set up for the tests, but what if we were assessing a production environment? Then, additional measures might have to be considered, such as Rules of Engagement, increasing the complexity of the approach. The good part is

that after defining the new set of controls, the new version of the method is ready to be applied in assessments with similar requirements.

Up to what extent our results are valid?

Even though we had a dedicated testing environment and access to any necessary resource, fact that made it easier to execute the attacks (e.g., via proxying or sniffing), we have to bear in mind that a skilled attacker could perform the same actions on the system, only needing more preparation to do so. Just having the possibility for an attack is enough reason to be worried about the security of a system. In a production environment the system will be exposed and, if any small security flaw is active, we should expect some form of attack to be successful. In this way, the vulnerabilities found have to be mitigated; since we do not know about possible hidden problems, it is better to eliminate the ones we are sure about.

We also have to point out that both POCA and the MPOWER platform are prototypes of an ongoing research project, which may have relaxed on the security aspects in a first moment, and are still not ready for production. We expect that this report will motivate the ones involved in the project to properly integrate security in it.

Chapter 8

Conclusions and Further Work

This chapter summarizes the achievements of our work and discusses further possibilities related to this project.

8.1 Conclusions

Any project dealing with sensitive information should take the appropriate actions to protect this information from being disclosed or manipulated (e.g., deleted, altered), and such is the case for the MPOWER platform. Its objectives are noble and with a security focused approach it can fulfill the requirements, shown in table 2.2, needed to comply with the laws and regulations. Just in this way it will be usable by healthcare organizations.

The choice for using Service Oriented Architecture makes it easier for the middleware services to be utilized by different organizations that wish to share information to achieve better results in their operations. Sharing knowledge can help all involved parties to deliver better services to their patients, but if the means to provide such services are not secure, private information can end up in the hands of malicious entities. SOA introduces new challenges to safeguard data, and if these are not properly understood when designing and implementing a SOA-based system, the results will not be satisfactory.

Designing and implementing secure systems is not an easy task, especially when there are many entry points into the system as is the case in SOA. The fact that trust boundaries are crossed is just another important reason to pay attention to security, since we do not know how secure are the zones being traversed and what could happen to the data in transit. As seen during the assessment, transport-layer allied with message-layer security may help

providing the level of security required in a (healthcare) SOA environment, but other standard measures have to be taken as well, such as input validation and proper configuration.

We have seen that SOA-based applications may be target of attacks employed against the traditional client-server architecture approach, besides a new wave of attacks. In this way, when testing for security every possible test case have to be considered. Regardless of the initial cost increase, having such focus since the beginning of a project and during the whole development life cycle could yield good results and save on mitigation or, even worse, lawsuit costs.

8.2 Further Work

As we have seen, it was not possible for the tester to validate the proposed solutions and to effectively evaluate the impact on security of a WS-Security implementation for the MPOWER platform. The next steps to be taken are to deploy the necessary countermeasures, properly configure the WS-Security mechanisms, put into action all the security components that could not be tested, and conduct a second assessment, based on the experiences achieved in this report. With a hardened system, and enough time for testing, the real effectiveness of web services security in a healthcare environment will be obtained.

A different setting for the tests could also be insightful, such as the one presented in figure 6.1. Such an environment could better provide us with interesting results from a security assessment by considering distributed services and untrusted zones, which are expected in a SOA paradigm.

Bibliography

- [1] Computer Economics. SOA Adoption Surges (accessed 16-02-2009). URL: <http://www.computereconomics.com/article.cfm?id=1423&tag=rbspot>, January 2009.
- [2] Gartner, Inc. Gartner Says the Number of Organizations Planning to Adopt SOA for the First Time Is Falling Dramatically (accessed 16-02-2009). URL: <http://www.gartner.com/it/page.jsp?id=790717>, November 2008.
- [3] IBM. New to SOA and web Services (accessed 17-02-2009). URL: <http://www.ibm.com/developerworks/webservices/newto/>.
- [4] Girish Juneja, Blake Dournaee, Joe Natoli, and Steve Birkel. Improving Performance of Healthcare Systems with Service Oriented Architecture (accessed 26-02-2009). URL: <http://www.infoq.com/articles/soa-healthcare>.
- [5] Nasjonal IKT. Tjenesteorientert arkitektur i spesialhelsetjenesten (SOA for specialized health services). Available at http://www.nasjonalikt.no/Publikasjoner/Tjenesteorientert_arkitektur_i_spesialisthelsetjenesten_hovedrapport_full_v1_0e.pdf, 2008.
- [6] Jeremy Epstein, Scott Matsumoto, and Gary McGraw. Software security and SOA: danger, Will Robinson! *Security & Privacy, IEEE*, 4(1):80–83, 2006.
- [7] New Rowley Group, Inc. The Challenge of Securing SOA. Available at ftp://ftp.software.ibm.com/software/uk/flexible/wp/the_challenge_of_securing_soa.pdf, 2006.
- [8] Commission of the European Communities. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995: On the Protection of Individuals with Regard to the Processing of Personal

- Data and on the Free Movement of such Data. *Official Journal of the European Communities L 281*, 23 November 1995, p. 31.
- [9] Sasan Adibi and Gordon B. Agnew. On the diversity of ehealth security systems and mechanisms. *Engineering in Medicine and Biology Society(EMBS)*, 2008. 30th Annual International Conference of the IEEE, pages 1478–1481, 2008.
- [10] MPOWER Consortium. Middleware Platform for eMPowering cognitive disabled and elderly (accessed 21-03-2009). URL: <http://www.mpower-project.eu>.
- [11] Karen Scarfone, Murugiah Souppaya, Amanda Cody, and Angela Orebough. *NIST Special Publication 800-115: Technical Guide to Information Security Testing and Assessment*. NIST, 2008.
- [12] Nicolai Josuttis. *SOA in Practice: The Art of Distributed System Design*. O’Reilly Media, Inc., 2007.
- [13] R. Shirey. Internet Security Glossary. IETF RFC 2828, 2000.
- [14] William Stallings. *Cryptography and Network Security (4th Edition)*. Prentice-Hall, Inc., 2005.
- [15] Hossein Bidgoli. *Handbook of Information Security, Information Warfare, Social, Legal, and International Issues and Security Foundations (Handbook of Information Security)*. John Wiley & Sons, Inc., New York, NY, USA, 2006.
- [16] D. Richard Kuhn, Vincent C. Hu, W. Timothy Polk, and Shu-Jen Chang. *NIST Special Publication 800-32: Introduction to Public Key Technology and the Federal PKI Infrastructure*. NIST, 2001.
- [17] Elaine B. Barker, William C. Barker, and Annabelle Lee. *NIST Special Publication 800-21: Guideline for Implementing Cryptography In the Federal Government*. NIST, 2005.
- [18] S. Chokhani and W. Ford. Internet X.509 Public Key Infrastructure. IETF RFC 2527, 1999.
- [19] IBM. Service-Oriented Architecture expands the vision of Web services, Part 1 (accessed 21-02-2009). URL: <http://www.ibm.com/developerworks/webservices/library/ws-soaintro.html>.
- [20] Jeff A. Estefan, Ken Laskey, Francis G. McCabe, and Danny Thornton. *Reference Architecture for Service Oriented Architecture Version 1.0*. OASIS Open, 2008.

- [21] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F. Brown, and Rebekah Metz. *Reference Model for Service Oriented Architecture 1.0*. OASIS Open, 2006.
- [22] Anoop Singhal, Theodore Winograd, and Karen Scarfone. *NIST Special Publication 800-95: Guide to Secure Web Services*. NIST, 2007.
- [23] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, and Ryo Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams Publishing, 2001.
- [24] Ramarao Kanneganti and Prasad Chodavarapu. *SOA Security*. Manning Publications Co., 2008.
- [25] Jothy Rosenberg and David L. Remy. *Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Sams Publishing, 2004.
- [26] Judith Hurwitz, Robin Bloor, Carol Baroudi, and Marcia Kaufman. *Service Oriented Architecture For Dummies (For Dummies (Computer/Tech))*. Wiley Publishing, Inc, 2006.
- [27] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246, 2008.
- [28] E. Rescorla. HTTP Over TLS. IETF RFC 2818, 2000.
- [29] Eric Pulier and Hugh Taylor. *Understanding Enterprise SOA*. Manning Publications Co., 2005.
- [30] Mamoon Yunus and Rizwan Mallal. An empirical study of security threats and countermeasures in web services-based services oriented architectures. In *WISE*, volume 3806 of *Lecture Notes in Computer Science*, pages 653–659, 2005.
- [31] Artem Vorobiev and Jun Han. Security attack ontology for web services. In *SKG '06: Proceedings of the Second International Conference on Semantics, Knowledge, and Grid*, page 42. IEEE Computer Society, 2006.
- [32] Hugo Haas and Allen Brown. Web Services Glossary (accessed 02-03-2009). URL: <http://www.w3.org/TR/ws-gloss/>.
- [33] IBM. IBM DeveloperWorks: Standards and Web Services (accessed 02-03-2009). URL: <http://www.ibm.com/developerworks/webservices/standards/>.

- [34] David Hunter, Jeff Rafter, Joe Fawcett, Eric van der Vlist, Danny Ayers, Jon Duckett, Andrew Watt, and Linda McKinnon. *Beginning XML, 4th Edition*. Wrox Press Ltd., 2007.
- [35] Fielding, et al. HTTP/1.1, part 1: URIs, Connections, and Message Parsing. IETF RFC 2616, 1999.
- [36] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005.
- [37] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP Version 1.2 Part 1: Messaging Framework (accessed 03-03-2009). URL: <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>, 2007.
- [38] Michael McIntosh, Martin Gudgin, K. Scott Morrison, and Abbie Barbir. WS-I Basic Security Profile, Version 1.0 (accessed 11-03-2009). URL: <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html>, 2007.
- [39] Mark O'Neill, Phillip Hallam-Baker, Seán Mac Cann, Mike Shema, Ed Simon, Paul A. Watters, and Andrew White. *Web Services Security*. McGraw-Hill, 2003.
- [40] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia, and Ed Simon. XML Signature Syntax and Processing (Second Edition) (accessed 12-03-2009). URL: <http://www.w3.org/TR/xmlsig-core>, 2008.
- [41] Takeshi Imamura, Blair Dillaway, and Ed Simon. XML Encryption Syntax and Processing (accessed 16-03-2009). URL: <http://www.w3.org/TR/xmlenc-core/>, 2002.
- [42] Phillip Hallam-Baker and Shivaram H. Mysore. XML Key Management Specification (XKMS 2.0) (accessed 16-03-2009). URL: <http://www.w3.org/TR/xkms2/>, 2005.
- [43] Paul Madsen and Eve Maler. *SAML V2.0 Executive Overview*. OASIS Open, 2005.
- [44] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS Open, 2005.
- [45] John Hughes, Scott Cantor, Jeff Hodges, Frederick Hirsch, Prateek

-
- Mishra, Rob Philpott, and Eve Maler. *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS Open, 2005.
- [46] Nick Ragouzis, John Hughes, Rob Philpott, Eve Maler, Paul Madsen, and Tom Scavo. *Security Assertion Markup Language (SAML) V2.0 Technical Overview*. OASIS Open, 2008.
- [47] Scott Cantor, John Kemp, Prateek Mishra, Rob Philpott, and Eve Maler. *Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS Open, 2005.
- [48] OASIS. OASIS Security Services TC (accessed 17-03-2009). URL: <http://www.oasis-open.org/committees/security/faq.php/>.
- [49] Tim Moses. *eXtensible Access Control Markup Language (XACML) Version 2.0*. OASIS Open, 2005.
- [50] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004) - OASIS Standard incorporating Approved Errata*. OASIS Open, 2006.
- [51] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and Ümit Yalçinalp. Web Services Policy 1.5 - Framework (accessed 16-03-2009). URL: <http://www.w3.org/TR/ws-policy/>, 2007.
- [52] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. *WS-SecurityPolicy 1.3*. OASIS Open, 2009.
- [53] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. *WS-Trust 1.4*. OASIS Open, 2009.
- [54] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. *WS-SecureConversation 1.4*. OASIS Open, 2009.
- [55] OASIS. OASIS Members Form Committee to Advance WS-Federation Identity Management Specification (accessed 19-03-2009). URL: <http://www.oasis-open.org/news/oasis-news-2007-05-02.php>.
- [56] Hal Lockhart et al. *Web Services Federation Language (WS-Federation) - Version 1.1*. IBM, Microsoft, et al, 2006.
- [57] Marius Mikalsen and Ståle Walderhaug. EMPOWERING THE ELDERLY AND THE COGNITIVELY DISABLED. Works in Progress: Healthcare Systems and Other Applications. *IEEE Pervasive Computing*, 6(1):59–60, 2007.

- [58] Ståle Walderhaug and Erlend Stav. *Overall Architecture - MPOWER Project Deliverable D1.1*. MPOWER Consortium, 2007.
- [59] Jostein Jensen, Inger Anne Tøndel, Martin Gilje Jaatun, Per Håkon Meland, and Herbjørn Andresen. Reusable Security Requirements for Healthcare Applications. In *ARES 2009: Proceedings of the Fourth International Conference on Availability, Security, and Reliability*. IEEE Computer Society, 2009.
- [60] Lovdata. LOV-1999-07-02-64/Helsepersonelloven. URL: <http://www.lovdata.no/cgi-wift/wiftldles?doc=/usr/www/lovdata/all/nl-19990702-064.html>, 1999.
- [61] Lovdata. FOR-2000-12-21-1385/Forskrift om pasientjournal. URL: <http://www.lovdata.no/cgi-wift/ldles?doc=/sf/sf/sf-20001221-1385.html>, 2000.
- [62] Lovdata. LOV-1999-07-02-6/Pasientrettighetsloven. URL: <http://www.lovdata.no/cgi-wift/wiftldles?doc=/usr/www/lovdata/all/nl-19990702-063.html>, 1999.
- [63] Lovdata. FOR-2000-12-15-1265/Personopplysningsforskriften. URL: <http://www.lovdata.no/cgi-wift/ldles?doc=/sf/sf/sf-20001215-1265.html>, 2000.
- [64] Lovdata. FOR-2001-07-01-744/Forskrift om informasjonssikkerhet. URL: <http://www.lovdata.no/cgi-wift/ldles?doc=/sf/sf/sf-20010701-0744.html>, 2001.
- [65] Lovdata. LOV-2001-05-18-24/Helseregisterloven. URL: <http://www.lovdata.no/cgi-wift/wiftldles?doc=/usr/www/lovdata/all/nl-20010518-024.html>, 2001.
- [66] Lovdata. LOV-2000-04-14-31/Personopplysningsloven. URL: <http://www.lovdata.no/cgi-wift/wiftldles?doc=/usr/www/lovdata/all/nl-20000414-031.html>, 2000.
- [67] Jostein Jensen. *Security Middleware Design - MPOWER Project Deliverable D5.2*. MPOWER Consortium, 2008.
- [68] Arun Kumar, Neeran Karnik, and Girish Chaffle. Context sensitivity in role-based access control. *SIGOPS Oper. Syst. Rev.*, 36(3):53–66, 2002.
- [69] Douglas A. Ashbaugh. *Security Software Development: Assessing and Managing Security Risks*. Auerbach Publications, 2008.

- [70] Brad Arkin, Scott Stender, and Gary McGraw. Software Penetration Testing. *IEEE Security & Privacy*, 3(1):84–87, 2005.
- [71] Mimi Herrmann. Security strategy: From soup to nuts. *Information Security Journal: A Global Perspective*, 18(1):26–32, 2009.
- [72] Warwick Ashford. Heartland data breach triggers class action suit (accessed 01-04-2009). URL: <http://www.computerweekly.com/Articles/2009/03/17/235295/heartland-data-breach-triggers-class-action-suit.htm>, 2009.
- [73] Robert Lemos. Data-breach lawsuit follows \$9 million heist (accessed 01-04-2009). URL: <http://www.securityfocus.com/brief/903>, 2009.
- [74] Brian Krebs. Sprint: Employee Stole Customer Data (accessed 01-04-2009). URL: http://voices.washingtonpost.com/securityfix/2009/03/sprint_employee_stole_customer.html, 2009.
- [75] Pete Herzog. *OSSTMM 3 LITE - Introduction and Sample to the Open Source Security Testing Methodology Manual*. ISECOM, 2008.
- [76] Pete Herzog. *OSSTMM 2.2 - Open Source Security Testing Methodology Manual*. ISECOM, 2006.
- [77] University of Oulu. Glossary of Vulnerability Testing Terminology. URL: <http://www.ee.oulu.fi/research/ouspg/sage/glossary/>, 2008.
- [78] Andrew Whitaker and Daniel Newman. *Penetration Testing and Cisco Network Defense*. Cisco Press, 2005.
- [79] James S. Tiller. *The Ethical Hack: A Framework for Business Value Penetration Testing*. Auerbach Publications, 2003.
- [80] OWASP. OWASP Main Page (accessed 09-04-2009). URL: http://www.owasp.org/index.php/Main_Page.
- [81] OWASP. *OWASP Testing Guide v3.0*. The OWASP Foundation, 2008.
- [82] OWASP. *OWASP Code Review Guide v1.1*. The OWASP Foundation, 2008.
- [83] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla, and Anandha Murukan. *Improving Web Application Security: Threats and Countermeasures*. Microsoft Corporation, 2003.

- [84] SIFT Pty Limited. SIFT - Information Security Services (accessed 09-04-2009). URL: <http://www.sift.com.au/index.asp>.
- [85] Colin Wong and Daniel Grzelak. *A Web Services Security Testing Framework - Version 1.0*. SIFT Pty Limited, 2006.
- [86] Tom Gallagher, Lawrence Landauer, and Bryan Jeffries. *Hunting Security Bugs*. Microsoft Press, 2006.
- [87] Meiko Jensen, Nils Gruschka, Ralph Herkenhoner, and Norbert Luttenberger. SOA and Web Services: New Technologies, New Standards - New Attacks. In *ECOWS '07: Proceedings of the Fifth European Conference on Web Services, 2007.*, pages 35–44, Nov. 2007.
- [88] Don Patterson. XML Firewall Architecture and Best Practices for Configuration and Auditing. SANS Institute, 2007.
- [89] Walid Negm. *Anatomy of a Web Service Attack*. Forum Systems, Inc., 2004.
- [90] David Davis. What is a VMware Snapshot? (accessed 11-05-2009). URL: http://www.petri.co.il/virtual_vmware_snapshot.htm, 2009.
- [91] VMware, Inc. VMware Server (accessed 11-05-2009). URL: <http://www.vmware.com/products/server/>.
- [92] Sun Microsystems, Inc. GlassFish - Open Source Application Server (accessed 11-05-2009). URL: <https://glassfish.dev.java.net/>.
- [93] Oracle. Oracle Database 10g Express Edition (accessed 11-05-2009). URL: <http://www.oracle.com/technology/products/database/xe/index.html>.
- [94] OWASP. OWASP WebScarab Project (accessed 12-04-2009). URL: http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project.
- [95] PortSwigger Ltd. Burp Suite (accessed 12-05-2009). URL: <http://portswigger.net/suite/>.
- [96] Wireshark Foundation. Wireshark: Go deep (accessed 14-05-2009). URL: <http://www.wireshark.org/>.
- [97] Eviware. The Web Service, SOA and SOAP Testing Tool - soapUI (accessed 13-04-2009). URL: <http://www.soapui.org>.

- [98] OWASP. OWASP WSFuzzer Project (accessed 13-04-2009). URL: http://www.owasp.org/index.php/Category:OWASP_WSFuzzer_Project.
- [99] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: Brute Force Vulnerability Discovery*. Addison-Wesley Professional, 2007.
- [100] Eui nam Huh, Jong-Youl Jeong, Young-Shin Kim, and Ki-Young Mun. Secure XML Aware Network Design and Performance Analysis. In *Computational Science and Its Applications - ICCSA 2005, International Conference, Singapore, May 9-12, 2005, Proceedings, Part I*, pages 311–319, 2005.
- [101] OWASP. OWASP Top 10 2007 (accessed 03-05-2009). URL: http://www.owasp.org/index.php/Top_10_2007.
- [102] Amit Klein. DOM Based Cross Site Scripting or XSS of the Third Kind (accessed 05-05-2009). URL: <http://www.webappsec.org/projects/articles/071105.shtml>, 2005.
- [103] RSnake. XSS (Cross Site Scripting) Cheat Sheet (accessed 12-05-2009). URL: <http://ha.ckers.org/xss.html>.
- [104] D. Kristol and L. Montulli. HTTP State Management Mechanism. IETF RFC 2965, 2000.
- [105] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.1 (accessed 03-05-2009). URL: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>, 2000.
- [106] Keith Ballinger, David Ehnebuske, Christopher Ferris, Martin Gudgin, Canyang Kevin Liu, Mark Nottingham, and Prasad Yendluri. WS-I Basic Profile, Version 1.1 (accessed 03-05-2009). URL: <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>, 2006.
- [107] Christopher Ferris, Anish Karmarkar, and Prasad Yendluri. WS-I Basic Profile, Version 2.0, Working Group Draft (accessed 03-05-2009). URL: [http://www.ws-i.org/Profiles/BasicProfile-2_0\(WGD\).html](http://www.ws-i.org/Profiles/BasicProfile-2_0(WGD).html), 2007.
- [108] OWASP. Guide to SQL Injection (accessed 18-05-2009). URL: http://www.owasp.org/index.php/Guide_to_SQL_Injection.
- [109] Martin Gudgin, Marc Hadley, and Tony Rogers. Web Services Ad-

- dressing 1.0 - Core (accessed 20-05-2009). URL: <http://www.w3.org/TR/ws-addr-core/>, 2006.
- [110] Gary McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.
- [111] Jayamsakthi Shanmugam and M. Ponnaivaikko. A solution to block cross site scripting vulnerabilities based on service oriented architecture. *Computer and Information Science, ACIS International Conference*, 0:861–866, 2007.
- [112] Christopher Steel, Ramesh Nagappan, and Ray Lai. *Core Security Patterns: Best Practices and Strategies for J2EETM, Web Services, and Identity Management*. Prentice Hall PTR, 2005.
- [113] OWASP. OWASP Validation Documentation (accessed 23-05-2009). URL: http://www.owasp.org/index.php/OWASP_Validation_Documentation.
- [114] Robert Lemos. Session hijacking now point and click (accessed 25-05-2009). URL: <http://www.securityfocus.com/brief/562>.
- [115] Nicholas Weaver. Sidejacking, Forced Sidejacking, and Gmail (accessed 25-05-2009). URL: <http://blog.icir.org/2008/02/sidejacking-forced-sidejacking-and.html>.
- [116] JAX-WS Reference Implementation Project. Configuring Security Using NetBeans IDE (accessed 29-05-2009). URL: https://jax-ws.dev.java.net/guide/Configuring_Security_Using_NetBeans_IDE.html, 2009.
- [117] Marcin Wielgoszewski. Writing a web services fuzzer in 5 minutes to SQL injection (accessed 10-05-2009). URL: <http://www.tssci-security.com/archives/2008/12/14/writing-a-web-services-fuzzer-in-5-minutes-to-sql-injection/>.
- [118] Haxx. cURL (accessed 10-05-2009). URL: <http://curl.haxx.se/>.

Appendix A

Fuzzing

This appendix presents files that were necessary during the process of fuzzing the application for vulnerabilities related to SQL injection and XSS.

A.1 Attack Vectors' File

The following file contains a few of the attack vectors elaborated for the WSFuzzer tool, and that were used to generate our own specially crafted SOAP requests:

```
<script>alert("XSS")</script >:::XSS
<script>alert(document.cookie)</script >:::XSS
&lt ; script&gt ; alert ( ' xss ' ) & lt ; / script&gt ; :::XSS
<IMG%20SRC='javascript : alert ( document . cookie ) ' >:::XSS
'%3Ciframe%20src=javascript : alert (%2527XSS%2527)%3E%3C/iframe%3E:::XSS
<![CDATA[<]]>SCRIPT<![CDATA[>]]> alert ( ' XSS ' ) ; \
  <![CDATA[<]]>/SCRIPT<![CDATA[>]] >:::XSS
':::SQL Injection (SQLi)
":::SQL Injection (SQLi)
--';:::SQL Injection (SQLi)
0 or 1=1:::SQL Injection (SQLi)
') or ('a'='a):::SQL Injection (SQLi)
' or username like '%:::SQL Injection (SQLi)
```

The reader should note that the blank lines were added for better readability and the "\ " character indicates that the line continues, and should be


```
done
echo "_done"

# Cleanup
echo "Removing temporary files no longer needed."
rm $1~

exit 0
```

A snippet of the script execution:

```
./fuzzerParser-soapUI.sh sql_attack.txt
Generating attacks ..... done
Removing temporary files no longer needed.
```

A.4 Script for Sending SOAP Requests

This is another script inspired by the work of Marcin. It uses the tool cURL [118] to send the requests to the service provider:

```
#!/usr/bin/env bash
#using proxy at localhost:8010
for i in `ls *.xml.*`;
do
    curl -A "rsassoon" -s -k -x localhost:8010 -d @$i -H \
    "Content-Type: _text/xml; charset=UTF-8" -H \
    "SOAPAction: _\"authenticateUserPass\"" \
    http://129.241.252.121:8080/MPOWER-Security/ \
    AuthenticationWServiceService;
done
```