إقـــــرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

An Ontology-Based Approach for Detecting SOAP Message Attacks

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وإن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

DECLARATION

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification

Student's name:	اسم الطالب/ة: محمود اكرم حمودة
Signature:	التوقيع: محمود
Date:	التاريخ: 6 / 03 / 2016

بسم الله الرحمن الرحيم

Islamic University – Gaza
Deanery of Graduate Studies
Faculty of Information Technology

الجامعة الإسلامية – غزة
عمادة الدراسات العليا
كلية تكنولوجيا المعلومات

An Ontology-Based Approach for Detecting SOAP Message Attacks

Submitted By
Mahmoud A.K. Hamouda

Dr. Rebhi S. Baraka
Supervisor

January 2016 - Rabi' Thani 1437

A Thesis Submitted as Partial Fulfillment of the Requirements for the Degree of Master in Information Technology

المنارة للاستشارات

www.manaraa.com

بسم الله الرحمن الرحيم

الجامعة الإسلامية – غزة
The Islamic University - Gaza

مكتب نائب الرئيس للبحث العلمي والدراسات العليا   هاتف داخلي: 1150

الرقم ج س غ/35/ Ref

التاريخ 2016/02/14 Date

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحث/ **محمود اكرم سعيد حمودة** لنيل درجة الماجستير في **كلية *تكنولوجيا المعلومات*** **برنامج تكنولوجيا المعلومات** وموضوعها:

طريقة تعتمد على الأنتولوجيا لكشف هجمات رسائل SOAP

An ontology-Based Approach for Detecting SOAP Message Attacks

وبعد المناقشة التي تمت اليوم الأحد 05 جمادى الأولى 1437هـ، الموافق 2016/02/14م الساعة الحادية عشرة صباحاً، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

د. ربحي سليمان بركة	**مشرفاً و رئيساً**	....
د. توفيق سليمان برهوم	**مناقشاً داخلياً**	....
أ.د. سامي سليم أبو ناصر	**مناقشاً خارجياً**	....

وبعد المداولة أوصت اللجنة بمنح الباحث درجة الماجستير في **كلية *تكنولوجيا المعلومات*/ برنامج تكنولوجيا المعلومات.**

واللجنة إذ تمنحه هذه الدرجة فإنها توصيه بتقوى الله ولزوم طاعته وأن يسخر علمه في خدمة دينه ووطنه.

والله ولي التوفيق ،،،

نائب الرئيس لشئون البحث العلمي والدراسات العليا

أ.د. عبدالرؤوف علي المناعمة

ص.ب. 108 الرمال، غزة، فلسطين هاتف Tel: +970 (8) 286 0700 فاكس fax: +970 (8) 286 0800 P.O. Box 108, Rimal, Gaza, Palestine
public@iugaza.edu.ps www.iugaza.edu.ps

Acknowledgements

First, I thank Allah for giving me strength and ability to complete this study.

I also would like to express deepest gratitude to my advisor Dr. Rebhi S. Baraka for his full support, expert guidance, understanding and encouragement throughout my study and research. Without his incredible patience and timely wisdom and counsel, my thesis work would have been frustrating and overwhelming pursuit. In addition, I would like to express my appreciation to Deanery of Graduate Studies staff of the Faculty of Information Technology in the Islamic University of Gaza.

I also thank my colleagues at the college of Information Technology for their inspiring and fruitful discussion and suggestion. A special thanks goes to my college Mr. Mahmoud A. El-Askary for his valuable technical helps.

Finally, I should extends a special thank for my parent, my brother, and my wife for their unconditional love and support during my study. I would not have been able to complete this thesis without their continuous support, love, and encouragement.

المنارة للاستشارات
www.manaraa.com

Abstract

Web Services is a distributed communication technology that can implement Service Oriented Architecture (SOA) to support the requirement of business process integration. Simple Object Access Protocol (SOAP) is a lightweight protocol that standardized a framework for web service communications. Messages written in SOAP are susceptible to different kinds of security attacks negatively affecting the integrity, confidentiality and efficiency of the SOAP message and the related Web Services.

In this thesis, we intend to build an ontology-based detection approach aiming to check SOAP message Web Services attacks and in certain cases detect the tampered messages. The approach consists of two main parts: the predefined ontology SOAP message part, which aim to preserve the SOAP message structure, by capturing the SOAP elements, and defines the relationship between these elements, this part of the approach is responsible for detecting and preventing XML rewriting attacks. The second part consist of the policy filters, these policy filters check if the SOAP Message comply to some requirements, which determine if the SOAP message violates some restriction rules that might lead to considering the SOAP message as vulnerable to Denial of Services attack.

In our approach we preserve integrity using ontology checker which checks that the SOAP message has not been modified during the transmission process. The confidentiality is preserved by encrypting a copy of the SOAP message in a log file. Finally we achieve time efficiency by executing the policy filters in a concurrent manner. The results show a33 times speed up of the concurrent execution of the policy filters over the sequential execution.

Keywords: *Web Services, SOAP message, XML Rewriting attacks, WS-Security, Replay Attack, Coercive-parsing attack, oversized attack, and parameter tampering attack.*

للاستشارات

www.manaraa.com

Acknowledgements

First, I thank Allah for giving me strength and ability to complete this study.

I also would like to express deepest gratitude to my advisor Dr. Rebhi S. Baraka for his full support, expert guidance, understanding and encouragement throughout my study and research. Without his incredible patience and timely wisdom and counsel, my thesis work would have been frustrating and overwhelming pursuit. In addition, I would like to express my appreciation to Deanery of Graduate Studies staff of the Faculty of Information Technology in the Islamic University of Gaza.

I also thank my colleagues at the college of Information Technology for their inspiring and fruitful discussion and suggestion. A special thanks goes to my college Mr. Mahmoud A. El-Askary for his valuable technical helps.

Finally, I should extends a special thank for my parent, my brother, and my wife for their unconditional love and support during my study. I would not have been able to complete this thesis without their continuous support, love, and encouragement.

المنارة للاستشارات
www.manaraa.com

عنوان البحث

طريقة تعتمد على الانتولوجيا في كشف هجمات رسائل SOAP

الملخص:

تعتبر خدمات الويب المنظومة الاكثر شهرة للاتصالات الموزعة والتي بدورها تحقق مفهوم البنية الموجهة للخدمات (Service Oriented Architecture)، حيث تدعم المتطلبات الخاصة بالمؤسسات من حيث تحقيق التكاملية بين اجزائها واجزاء منظمات اخرى. بروتوكول SOAP عبارة عن بروتوكول اتصالات خفيف يستخدم في ارسال الرسائل بين خدمات الويب. رسائل SOAP عادة تكون عرضة لأنواع مختلفة من الهجمات الامنية التي تستهدف سلامة وسرية وكفاءة خدمات الويب ذات الصلة.

هذه الدراسة تهدف الى بناء طريقة تعتمد على الانتولوجيا (ontology) لكشف الهجمات الامنية التي تتعرض لها رسائل SOAP. الطريقة تتكون من جزئين رئيسيين هما الانتولوجيا المعرفة مسبقا بناءا على تركيب رسالة SOAP، والمرشحات (filters) التي تتأكد من تحقيق رسالة SOAP للمتطلبات والقيود التي تعمل على حمايتها من هجمات حرمان الخدمة (denial of service).

الجزء الأول يقوم بتحديد العناصر الاساسية في رسالة SOAP وتصنيفها كعناصر للانتولوجيا وتحديد العلاقات فيما بينها بهدف كشف أي تعديل على الرسالة، فيما يعرف باسم اعادة ترتيب او تركيب رسالة SOAP (XML rewriting attacks)، من خلال مايعرف بفاحص الانتولوجيا (ontology checker) بحيث يتم التأكد من تركيب رسالة SOAP وانه لم يتم أي تعديل بهدف تخريبي لها خلال عملية الاتصال. الجزء الثاني وهو المرشحات (filters) حيث تقوم مجموعة متوازية من المرشحات بفحص خلو الرسالة من الهجمات التي تتعلق بالحرمان الخدمة (denial of service).

من خلال طريقتنا يتم التحقق من حفظ سلامة الرسالة باستخدام فاحص الانتولوجيا الذي يفحص الرسالة ضد اي تعديل خلال نقلها وان المتغيرات التي حدثت للرسالة ليست معادية. اما سرية الرسالة فيتم التحقق منها من خلال الاعتماد على تخزين نسخة مشفرة في سجل (log file). أما سرعة الفحص فتم التحسين عليها من خلال تنفيذ فحوصات المرشحات بشكل متوازي وذلك لانعدام الاعتمادية فيما بينهم. أثبت النتائج أن تنفيذ فحص المرشحات على التوازي أسرع ب 33 مرة من تنفيذ الفحص على التوالي.

المنارة للاستشارات
www.manaraa.com

Contents

Abstract 2
Chapter 1 Introduction 11
1.1. Statement of the Problem 12
1.2. The Objectives 12
1.2.1. Main Objective 12
1.2.2. Specific Objectives 12
1.3. Importance of the Thesis 12
1.4. Scope and Limitations of Thesis 13
1.5. Methodology 13
1.6. Thesis Format 14
Chapter 2 Technical Foundation 15
2.1. Overview of Web Services 15
2.2. SOAP message 16
2.3. Web Services attacks 18
2.4. Ontology 19
2.1.1. Reasoning 21
2.5. Ontology Development Processes 21
2.6. Policies 24
2.7. Document Object Model (DOM) 25
2.8. SOAP with Attachment API for Java (SAAJ) 26
2.9. XML Path Language (XPath) 27
2.10. Apache Jena (Jena Framework) 28
2.10.1 Jena Architecture: 28
2.11. Log File 31
2.11.1 Log4j has three main components: 31
2.11.2 Log4j Features 31
2.11.3 Pros and Cons of Logging 32
2.12. Summary 32
Chapter 3 Related Works 34
3.1. XML Rewriting Approaches 34

المنارة للاستشارات
www.manaraa.com

3.1.1. Inline Approaches ... 34
3.1.2. Policy Approaches ... 35
3.1.3. Other Approaches ... 36
3.2. Denial of Services Approaches ... 37
3.2.1. Oversized Payload Approaches ... 37
3.2.2. Parameter Approaches ... 38
3.2.3. Coercive-Parsing Approaches ... 38
3.2.4. Replay Approaches ... 39
3.3. Overcoming Limitations ... 39
3.3.1. First XML Rewriting attacks Overcoming limitations ... 39
3.3.2. Second Denial of Services (DOS) Overcoming Limitations ... 40
3.4. Summary ... 40
Chapter 4 The Proposed Approach for Detecting SOAP Message Attacks ... 42
4.1. Overall Structure of the Approach ... 42
.4.2 The Ontology based on Predefined SOAP message Structure ... 46
4.3. Policies determination ... 53
4.3.1. Oversized filter ... 54
4.3.2. Message Replay Filter ... 54
4.3.3. Parameter Tampering Filter ... 55
4.3.4. Coercive Parsing Filter ... 55
4.4. Summary and Discussion ... 56
Chapter 5 Implementation ... 57
5.1. SOAP message Creation ... 57
5.2. Building Log File ... 60
5.3. SOAP-Message Ontology Structure Implementation ... 62
5.3.1. Define Important Terms (Class Hierarchy) ... 62
5.3.2. Define Slots and Facets ... 64
5.4. Detecting XML Rewriting Attacks in SOAP Message Using Ontology Checker ... 67
5.5. Policy Filters Implementation ... 69
5.5.1. Oversized Filter ... 70
5.5.2. Replay Attack Filter ... 70
5.5.3. Parameter Tampering Filter ... 70

المنارة للاستشارات
www.manaraa.com

5.5.4. Coercive Parsing Filter ... 71
5.6. Summary ... 72
Chapter 6 Experimentation and Evaluation ... 73
6.1. Experimentation and testing cases ... 73
6.1.1. Checking Non-modified SOAP message ... 73
6.1.2. Checking the approach against XML Rewriting attack ... 74
6.1.3. Checking the Approach against Replay Attack ... 75
6.1.4. Checking the approach against Oversized attack ... 75
6.1.5. Checking the approach against Parameter Tampering attack ... 76
6.1.6. Checking the approach against Coercive-Parsing attack ... 76
6.2. Evaluation ... 77
6.2.1. Integrity ... 77
6.2.2. Confidentiality ... 77
6.2.3. Time efficiency ... 79
6.3. Summary ... 80
Chapter 7 Conclusion and Recommendations ... 81
7.1. Conclusion ... 81
7.2. Recommendation ... 81
Bibliography ... 82
Appendixes ... 86
Appendix 1: The SOAP Structure Ontology in OWL Format: ... 86
Appendix 2: SOAP Creation Code ... 90
Appendix 3: Policy Filters Code ... 93
Appendix 4: The Ontology Checker Code ... 97

المنارة للاستشارات
www.manaraa.com

List of Figures

Figure 2.1Web Services Architecture [5] 16
Figure 2.2 XML tree representation for the SOAP message [5]..................... 17
Figure 2.3 SOAP message Structure..................... 17
Figure 2.4 Class Definition in OWL..................... 20
Figure 2.5 Ontology Development Processes [16] 23
Figure 2.6 An example of WS-Policy..................... 24
Figure 2.7 Create SOAPConnection using SAAJ API 26
Figure 2.8 An example of XPath Language..................... 27
Figure 2.9 An Example of Jena Apache 28
Figure 2.10 Jena Architecture 29
Figure 2.11 An Example of creating a triple in Graph Layer 29
Figure 2.12 Creating a Triple in Model Layer 30
Figure 4.1 Structure of the adaptable ontology-based approach..................... 43
Figure 4.2 Structure of the SOAP message using Ontology..................... 44
Figure 4.3 Ontology Checker procedure (flow Chart)..................... 45
Figure 4.4 Filters procedure 46
Figure 4.5 Ontology Individuals 51
Figure 4.6 SPARQL for Calling Envelope element child..................... 52
Figure 4.7 Envelope SPARQL..................... 52
Figure 4.8 the SPARQL of Return the Timestamp child elements..................... 53
Figure 4.9 SPARQL of the Timestamp child elements 53
Figure 4.10 Oversized filter Algorithm 54
Figure 4.11 The Algorithm of Message Replay Filter..................... 54
Figure 4.12 The Algorithm of Parameter Tampering Filter 55
Figure 4.13 The Algorithm of the Coercive Parsing Filter..................... 55
Figure 5.1 SOAP Message Element in SAAJ 58
Figure 5.2 SOAP message Creation and Connection 58
Figure 5.3create SOAP Part elements..................... 59
Figure 5.4 SOAP Header Elements create 59
Figure 5.5 Create the Attachment Part..................... 60
Figure 5.6 file of properties 60
Figure 5.7 The layout of the Log file..................... 60
Figure 5.8 Capture SOAP Structure..................... 61
Figure 5.9 An Example of Log File 62
Figure 5.10 Log File Logging Event..................... 62
Figure 5.11 The SOAP message Ontology Structure 62
Figure 5.12 SOAP message Ontology Structure Properties 64
Figure 5.13 parentOf Properties Example 65
Figure 5.14 childOf Properties Example 65
Figure 5.15 hasNode Property 66
Figure 5.16 unique property..................... 66
Figure 5.17 isExist Property 67

المنارة للاستشارات
www.manaraa.com

Figure 5.18 Prepare SOAP Message 67
Figure 5.19 Return All Elements based on Ontology 68
Figure 5.20 Return Envelope Sub-elements 68
Figure 5.21 Return Header sub-Elements 68
Figure 5.22 Return Timestamps sub-elements 68
Figure 5.23 Return received SOAP message elements 69
Figure 5.24 Return Envelope sub-elements received SOAP message 69
Figure 5.25 The Oversized Filter 70
Figure 5.26 The Message Replay Attack 70
Figure 5.27 The Parameter Tampering Filters 71
Figure 5.28 Coercive Parsing Filter 71
Figure 6.1 SOAP message without any Modification 73
Figure 6.2 Normal Case Result 74
Figure 6.3 Malicious SOAP message modification structure 74
Figure 6.4 Time-Stamp Modification Recognize 74
Figure 6.5 An Old SOAP message 75
Figure 6.6 Replay attack 75
Figure 6.7 Oversized Attack 76
Figure 6.8 Parameter Type Test 76
Figure 6.9 Coercive Parsing Attack 77
Figure 6.10 Coercive Parsing checking Result 77
Figure 6.11 concurrent procedure for filters checking process 80
Figure 6.12 non-concurrent procedure for filters checking process 80

المنارة للاستشارات
www.manaraa.com

List of Tables

Table 4-1 Ontology Main Terms 48
Table 4-2 Ontology Object Properties 49
Table 4-3 Ontology Data Properties 50

المنارة للاستشارات
www.manaraa.com

Chapter 1 Introduction

Web Services is a distributed systems where devices in the network exchange specific form of XML document called Simple Object Access Protocol (SOAP) message. SOAP message allows the existence of network intermediaries. An intermediary can be a routers or a firewall. SOAP message allows these intermediaries to process the SOAP message, by adding or modifying headers. Because of its simplicity, standardization and platform independent nature a lot of business organizations have embraced Web Services technology for the integration of data, system and application inside and outside their organization boundary such as Amazon and Yahoo are using Web Services to integrate various applications. Due to this growing adoption by different organization, security of Web Services became as a vital issue that slowing the deployment, according to a new survey of senior IT executives sponsored by Computer Associates [1]. One key finding in this survey was that 43% of the respondents felt that security was the most significant issue related to the deployment of Web Services applications.

The communication between Web Services occurs via SOAP messages. Thus, making Web Services secure means making SOAP messages secure during the exchange of messages. Several security standards such as WS-Security [2] are used to secure SOAP messages exchange in Web Services environment. However, [3] demonstrated that the content of a SOAP messages, secured with XML digital signature, could be changed without invalidating the digital signature. This is so called XML rewriting attacks or XML Signature Wrapping attacks.

In this research, we intend to build an approach for detecting SOAP message attacks, which depends on the ontology and the policy filters. The sender captures the SOAP message and attaches it to the log file, encrypt the log file, attach the log file to the SOAP message, then send the SOAP message to the Web Services provider.

First the predefined ontology checker checks the SOAP message and compare the element position with the predefine ontology structure. If there is any modification, the ontology checker clarify it, and see if it is a malicious intent or not. If the modification is not malicious the ontology checker passed the SOAP message to the policy filters. Otherwise if the modification is malicious, the ontology checker decrypt the SOAP message captured in the log file and compares it also with the predefined SOAP ontology structure. If the log file SOAP has not any structuring problem, the ontology checker modify the receiving message to become like the log file SOAP message. Then sends it to the policy filters. Else, if the log file SOAP message version suffering from malicious vulnerability the ontology checker rejects the SOAP message.

In addition the log file play an important role in registering each logging event that happens to the SOAP message during the transmission process and displays it in a manner where machine and human can read it, and specify which intermediate node modifies the SOAP message and when the modification has happened.

المنارة للاستشارات
www.manaraa.com

The policy filters focus in four SOAP message attacks namely, Replay attack, Oversized attack, Parameter Tampering attack, and Coercive Parsing attack. Each filter has to handle one kind of attack; also, they are working in parallel as multiple thread.

The rest of this chapter consists of the following sections. Section 1.2 presents the statement of the problem. Section 1.3 presents the objectives of the thesis. Section 1.4 states the importance of the research. Section 1.5 addresses the scope and limitations of the research. Section 1.5 illustrates the methodology. Finally, Section 1.6 outlines the research format.

1.1. Statement of the Problem

Protecting SOAP message from Web Services attacks is a crucial issue for the security of Web Services. Existing approaches to this issue suffer from handling one kind of attack and ignore other kinds and they tend to exhaust the resources of the services in the detection process.

The problem of this research is how to design an efficient approach for the detection of SOAP message attacks in Web Services that preserves integrity and confidentiality.

1.2. The Objectives

1.2.1. Main Objective

To design a time efficient ontology-based approach for detecting SOAP message against Web Services attacks which can detect multiple kinds of attacks while preserving integrity and confidentiality.

1.2.2. Specific Objectives

The specific objectives of the thesis are:

- To analyze the different approaches used to protect SOAP messages and investigate the limitations of each solution.
- Building the corresponding ontology for SOAP message structure and the appropriate filters.
- To design the approach for detecting SOAP message attacks based on the ontology and the filters.
- To implement the approach including the ontology checker and the concurrent policy filters using suitable tools.
- To conduct the required experiments and evaluate the approach for preserving integrity and confidentiality and ensuring time efficiency.

1.3. Importance of the Thesis

- The proposed approach depends on the ontology to guarantee integrity and confidentiality for the SOAP message.
- It attempts to decrease the use of resources on the detection process by handling attacks before it reaches the service.

المنارة للاستشارات
www.manaraa.com

- The approach makes sure that the SOAP message is non-changeable (preserves integrity) by capturing its structure using the ontology.
- It makes Web Services communication under SOAP messages more confidential.

1.4. Scope and Limitations of Thesis

- Our solution depends on ontology and filter polices.
- The approach focuses on detecting SOAP message attacks such as rewriting attack, replay attack, coercive-parsing attack, oversized attack and parameter-tampering attack.
- Time efficiency is preserved using multiple thread technique in order to let the filters run concurrently and reduce filtering time.
- Our approach will be at the middle between the service requester and the service provider.
- The approach consists of filters, and these filters are following specific policies such as the message size and the timestamp element.
- Validation process is done before the SOAP message reaches the service.
- Cross-site scripting attack cannot be handled by our approach.
- Our approach depends on the ontology checker but the other policy filters do not depends on ontology.

1.5. Methodology

We intend to accomplish this thesis by using the following methodology:

Phase 1: First, we investigate the Web Services attacks and there approaches, and try to conduct each approach and infer its limitations.

Phase 2: Analyze the requirements of the approach.

This phase is separated into four parts:

First, study the kinds of malicious attacks that manipulate the SOAP message.

Second, capture the structure of the SOAP message and add it as encrypted to the log file.

Third, develop the ontology that contain the predefined SOAP message structure, and the XML rewriting attacks case.

Fourth, we create strict policy roles to control the attack attempt.

Phase 3: Design the detection SOAP message attacks approach.

In this phase, we mainly design the ontology and policy filters:

Design the ontology to define the main elements that must exist in the SOAP message. The XML rewriting attacks kind are also registered as cases, and the log file for capturing the SOAP message structure and the log file for registering every modification that happened to the SOAP message through the transmitting process.

المنارة للاستشارات
www.manaraa.com

Design oversized filter, this filter aims to validate the SOAP message and make sure that it follows the policies to avoid Web Services attacks such as buffer over-flow. Design message replay filter, this filter compare the message timestamp to its current clock time value for synchronization. Design parameter tampering filter, this filter checks the SOAP message parameter for invalid data. Design coercive parsing filter, this filter verifies the received message for wrong format.

Phase 4: Implement the approach based on Phase 2 and Phase 3 using suitable tools and platform.

Phase 5: Conduct the necessary experiments on the detection of SOAP message attacks approach to make sure that it solves the problem.

Phase 6: Evaluate the approach to ensure that it preserves time consuming, integrity and confidentiality of the SOAP messages.

1.6. Thesis Format

The rest of the thesis is organized as follows: Chapter 2 presents technical foundation, Chapter 3 reviews related works; Chapter 4 presents the proposed approach, Chapter 5 details the implementation of the proposed approach, Chapter 6 explains the experiments, testing and evaluation the approach, and finally Chapter 7 present conclusion and future work.

المنارة للاستشارات
www.manaraa.com

Chapter 2 Technical Foundation

In this Chapter, we aim to present the technical foundation the proposed approach depending on to achieve the security goals we aim to. Firstly, we define the terminology such as Web Services, SOAP message.

Then we identify Web Services attacks. Which our approach, detect and handle. In addition, we discuss the ontology terminology because the approach depends heavily on ontology to check the SOAP message structure. Therefore, we identify the development steps of the ontology build. Moreover we review uses policies and finally we give an overview about it.

Also we present the main APIs the proposed approach used in the implementation process. At the first we introduce the Document Object Model (DOM) API. Then we present the SOAP with Attachment API for Java (SAAJ), which is an API for creating and manipulating SOAP message. After that we give a brief introduction about the XML Path Language (XPath), and how to use it. Then we discuss Apache Jena or Jena API for extracting data from and write to RDF graphs, also we present its architecture. Finally we present the Logging strategy, and take the Log4j Apache as an example.

2.1. Overview of Web Services

The World Wide Web consortium (W3C) defines Web Services as follows [4]: "a Web Services is a software system identified by a Uniform Resource Identifier (URI), whose public interfaces and bindings are defined and described using Extensible Markup Language (XML)".

In particular the use of Web Services depends on components. Services provider which offer the services operation, using Web Service Description Language (WSDL). This language describe the functionality requirement to use this services. Then the services provider publish this WSDL in the Universal Description Discovery and Integration (UDDI). The services requester discover the UDDI and consuming the appreciate services using Simple Object Access Protocol (SOAP) message. This process is shown in **Figure 2.1**.

A Web Services is a service that has the following properties [4]:

- The service can be accessed over the Internet
- The service uses a standard XML messaging system. SOAP is a communication protocol used for XML messaging.
- The service is not dependent on any particular operating system or programming language

Although not absolutely necessary, Web Services can have the following two properties as well

- A Web Services have a description written in XML. WSDL (Web Services Description Language) used for this.

المنارة للاستشارات
www.manaraa.com

- A Web Services can be located using a find mechanism. UDDI is a standard to locate Web Services.

Web Services provide a mechanism for machine-to-machine interaction. It is a way for programmatic access to web sites. Unlike traditional client/server models, such as a Web Server/Web page system, Web Services do not provide the user with a GUI. Web Services instead access business logic, data and processes through a programmatic interface across a network.

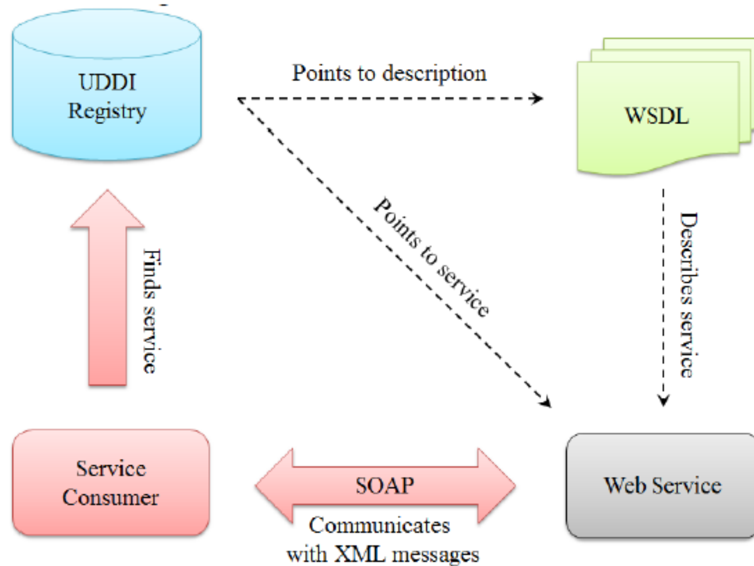


Figure 2.1Web Services Architecture [5]

2.2. SOAP message

The communication between different Web Services occurs via the SOAP messages. SOAP 1.2 specification defines SOAP messages as follows [6]: "SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment".

Simple Object Access Protocol (SOAP) depends as a base on the XML Technology, because of that SOAP can exchange over variety of protocols. This advantages allow the SOAP to be independent of any particular programming model, and other implementation specific semantics.

Figure 2.2 illustrates a sample XML tree representation for the SOAP message.

المنارة للاستشارات
www.manaraa.com

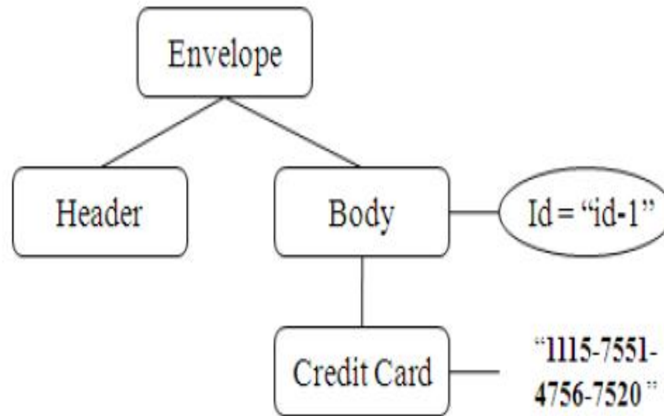


Figure 2.2 XML tree representation for the SOAP message [5]

The structure of a SOAP message in a pictorial representation illustrated in **Figure 2.3**. A SOAP message is encoded as an XML document. So SOAP message must has a root element, which is called <Envelope> element. The <Envelope> element will contain the following sub elements as its children.

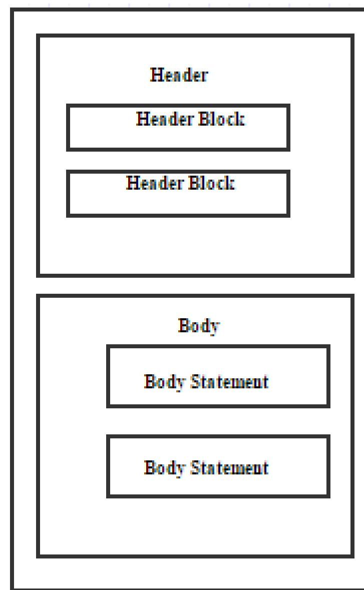


Figure 2.3 SOAP message Structure [5]

- An optional <Header> Element
- A mandatory <Body> Element

A <Header> element will contain data that is not the application payload. This element is intended to be processed by zero or more intermediaries along the path of the SOAP message from sender to the receiver. <Header> element will contain zero or more <HeaderBlock> as its child element. Each HeaderBlock within the <Header> element may realize zero or more features. For instance, to realize the security feature, which is not specified by the core SOAP

المنارة للاستشارات
www.manaraa.com

messaging framework, a <Security> header block will be used as the sub element of the <Header>.

The <Body> element is mandatory and contains the application payload. The <Body> element is always intended to be processed by the Ultimate Receiver of the SOAP message.

2.3. Web Services attacks

Web Services such any other environment suffer from vulnerability, in the following section we discuss and describe the main five Web Services attacks that influent the security availability, confidentiality and integrity.

- *XML rewriting attacks*

SOAP message is an XML-based document. One particular vulnerable case is that of a XML rewriting attacks – XML wrapping attack – which is a general name for a distinct type of attacks based on the malicious interception, manipulation, and transmission of SOAP messages in a network of communication system. The main limitation in WS-Security [2], WS-Policy [7] and other standards that they can`t exactly specify the real location of the element.

However, in practice, incorrect deployment of these standards especially by human being is very likely, leading to significant vulnerable cases [4].

- *Buffer overflow attack (over-sized attack)*

"The attacker inserts malicious content with well-formed message in SOAP request, which is beyond the allowable size of the buffer and causes Denial of Services Attack (DOS). It called buffer overflow attack" [8].

- *Message replay attack*

"A message replay attack is one in which the attacker eavesdrops and obtains a copy of an encrypted message and then reuses the message later in an attempt to reveal the secret message or to provide a fake identify. For example, when a legitimate client transfers money from this account in a bank to the receiver, the attacker steals the password and uses it multiple times by sending it to the Web Services in order to cause money lose" [8].

- *Parameter tampering attack*

"The WSDL document has parameters to receive inputs from the client. The parameters are visible in a WSDL structure to all users. Here, the attacker tries to send different data types of parameters several times. Then, the Web Services may crash" [8].

- *Coercive parsing attack*

"The attacker sends a SOAP message with an ultimate amount of opening tags in the SOAP body. It means the attacker sends a very deeply nested XML document into the targeted Web Services. If the parser receives a peculiar format of SOAP message, it reduces the processing capability and this may result in distributed denial of services attack" [8].

المنارة للاستشارات
www.manaraa.com

2.4. Ontology

Ontology has become one of the main ways to represent data. It is used to support a great variety of tasks. Nowadays there are applications of ontologies in commercial, industrial, academicals or research focus depend on the ontological principle and structure.

In addition, ontology is used to connect people, organizations and software programs together. Although if they used different viewpoints or environment. This divergence is natural and valuable, but it leads to problems in communication, interaction and understanding. Ontologies are an alternative to address these kind of problems. The main objective of this section is to give a brief introduction about the ontology.

Here we introduce the specific objectives to make the reader have a general understanding of the ontology power and ontology concept [9].

- To identify main characteristics of ontologies, their relevance and some of their applications.
- To review tools and languages to support ontologies.
- To look over relevant research projects.

Any ontology design for a specific purpose must answer main and specific question, such as what are ontologies? What are their characteristics?, what are they for?, how they address semantically and interoperability problems?, what tools or languages support their management?, how they have been used?.

To come over the ontology understand we want to introduce the main concept and explain it, and focus on the main language that has been used in knowledge sharing and data integration in conjunction with the semantic web activity [10]. So we will give a briefly discuss of the DAML+OIL [9] and the present one OWL [11], these language are based on RDF triples and support reasoning capability.

Darpa Agent Markup Language (DAML) is an ontology language developed by the RDF Core Working Group [12] in order to represent the ontology in a way more explicitly than XML, RDF, and RDF Schema, the DAML+OIL is an extensional of the DAML, which combines DAML with the Ontology Interface Layer (OIL) [9]. The DAML+OIL is consists of class elements, property elements, and instances. Also, it can imports statement to reference another DAML+OIL ontology. DAML+OIL divides the domain into data types and objects. This ontology language supported the field at the time it was recommending, but could not keep up with the growing need for more expressive ontologies because of the limited restriction and concept support. Thus, OWL took the place of DAML + OIL as the semantic web standard.

The Ontology Web Language (OWL) developed after the DAML+OIL and it is the current W3C standard for ontology languages and has been extended to provide more explicit description logics. [13]. The OWL has added three increasing level of expressivity it is OWL Lite, OWL DL, and OWL Full respectively. These advantages allow the user to define their own needs for

المنارة للاستشارات
www.manaraa.com

expressivity and chose the best version that support their needs. The OWL syntax employs URIs for naming and implements the description framework for the Web provided by RDF to add the following capabilities to ontologies: the ability to be distributed across many systems, scalability to Web needs, compatibility with Web standards for accessibility and internationalization, and openness and extensibility [14].

Here we want to make a small compare between the DAML+OIL and the OWL. The OWL has various updates to RDF and RDF Schema from the RDF Core Group [12]. The DAML+OIL restrictions has removed, and various properties and classes has renamed in OWL syntax. Examples of some of the differences in syntax can view in the sequence-analysis class definition examples is in **Figure 2.4** below. Note the difference in RDF tags and labels. In addition, Owl:SymmetricProperty was added and DAML+OIL synonyms for RDF and RDF Schema classes and properties were removed, as well as added properties and classes to support versioning and unique names assumptions. The Ontology Web Language OWL employs the most recent version of RDF Semantics, which thus replaces some semantic terms identified in DAML+OIL. RDF and RDF Schema updates include: allowing cyclic subclasses, handling multiple domain and range properties as intersections, changing namespaces, and implementing XML Schema data types and new syntax for list functions [13].

The updates and changes that was added have made the OWL more expressive ontology language standard.

```
- <owl:Class rdf:about="#SEQUENCE_ANALYSIS">
- <rdfs:subClassOf>
  <owl:Class rdf:about="#ANALYSIS" />
  </rdfs:subClassOf>
  </owl:Class>
```

Figure 2.4 Class Definition in OWL

OWL also support the distribution of the ontologies, which allow the semantics web to create and share ontologies in distributed manner across the web. When creating an ontology for a given use, it is most efficient and effective to rely on the expertise of others and previous models in order to provide a more robust representation of a domain. Thus, the integration of distributed ontologies becomes an important design implication [15]. Also, as the breadth and depth of the individual ontology increases, the ability to manage the information contained within the knowledge base also increases. Thus, the support of a distributed ontology system where specialized ontologies can be maintained as separate entities becomes an attractive option [10].

One advantage of a distributed ontology is that it can be collaboratively created and easily maintained over time. Specialists in their field of expertise can gain access to a particular part of the ontology in order to update and revise it as they see appropriate without interrupting the integrity of the top-level system ontology [10]. The ability to collaborate with many different

المنارة للاستشارات
www.manaraa.com

professionals adds to the depth and breadth of any ontology and will result in better reasoning and query capabilities.

2.1.1. Reasoning

When we talk about the expressive representation of an ontology we must talk about the tool available to infer information from the ontology. Many available Reasoners today exploit the capabilities of Description Logics. According to Lambrix, description logics are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies [14]. Ontology reasoning performed at two levels. On one level, a Reasoner provides the basic core usability of ontology by testing for concept satisfiability, class subsumption by concept hierarchy, class consistency, and instance checking [14]. Reasoners also support first order logic whereby users can create rules and query expressions in order to deduce answers from the knowledge base. The first order logic reasoning in description logics is based on concepts, roles, and, individuals. Concepts relate to classes in ontology language, roles are equivalent to relationships, and individuals found in both cases. As described, reasoners allow the information contained within an ontology to be utilized to its fullest potential to maintain and infer information. RacerPro is the description logics reasoner employed in this project to ensure concept satisfiability and as a tool for advanced query formulation and inference implementation.

2.5. Ontology Development Processes

Ontology is a formal specification of a shared conceptualization [18]. Ontology is a formal explicit representation of concepts in a domain, properties of each concept describes characteristics and attributes of the concept known as slots and constrains on these slots. Ontology is a shared conceptualization with a clear hierarchy and a strong support for logical consequences [18].

Ontology contains a set of specific and clearly described classes or concepts, property of the concepts, slot, restriction, facet and a series of instance related to one class, which combines to form the knowledge storage [19]. Class is the core of ontology, which describes the concepts in some domain. Slot describes the property of the class and the instance.

In the process of development an ontology, there is some rule must be emphasize in the design of the ontology. According to [16] the other things that need to considered are:

- There is exactly or right way to model a domain – there is always a suitable alternative the best solution depends on the application to be developed and throughout and extension that you anticipate and develop.
- Ontology development is a repetitive process.
- The concepts in the ontology should be close to the object (physical or logical) and relationships in the domain that you study. Normally nouns (objects) or verbs (relationships) in sentences will clarify and explain our domain.

المنارة للاستشارات
www.manaraa.com

The general steps in the design and development of ontology are summarized in **Figure 2.5** and as follows:

Step 1: Determine the Domain and Scope of the Ontology

This step defines the purpose and boundaries of the ontology. That is, answer several basic questions: what domain will the ontology cover? What is the purpose of the ontology? For what sorts of questions should the information in the ontology be able to provide answers? [16].

The answers to these questions may change during the ontology-design process, but at any given time, they help limit the scope of the model.

Step 2: Consider Reusing Existing Ontologies

This step is to ascertain if ontology has developed previously in the same subject area. If such ontology exists, it is easier to modify the existing ontology to suit ones needs than to create a new ontology [16]. Reusing existing ontology may be required if our system needs to interact with other applications that have been committed to a specific ontology or controlled vocabulary.

المنارة للاستشارات
www.manaraa.com

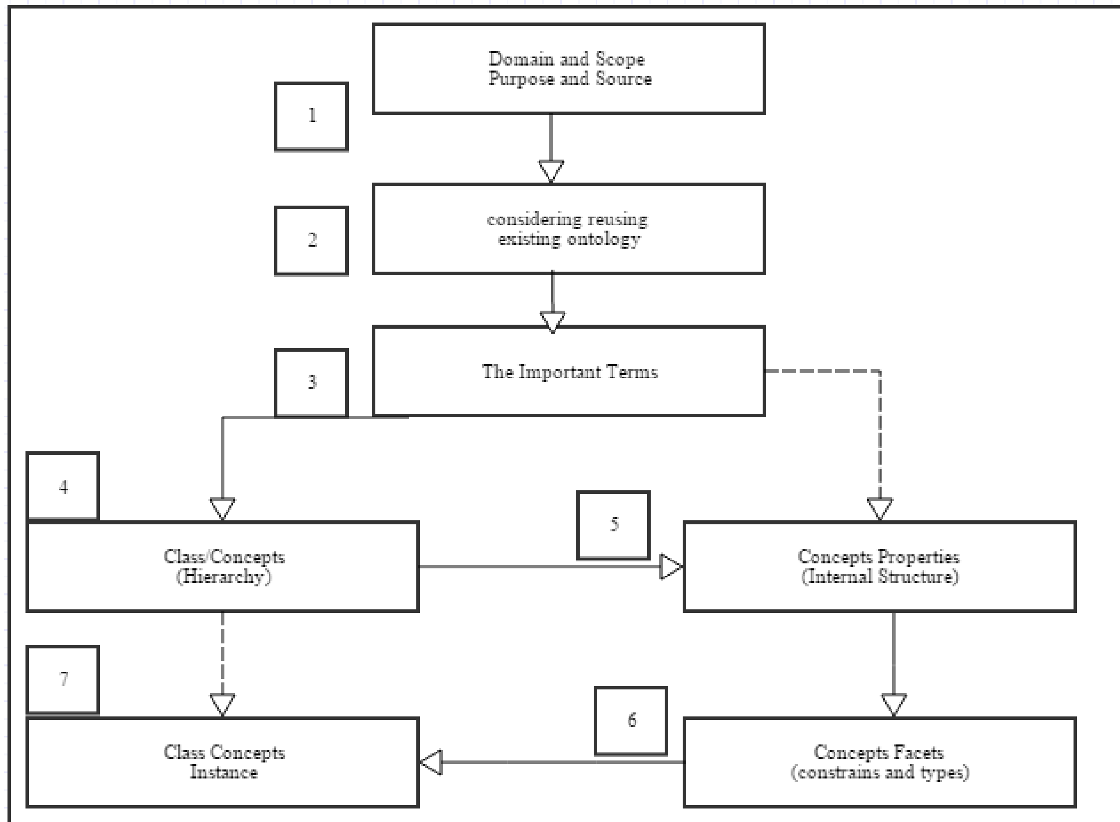


Figure 2.5 Ontology Development Processes [16]

Step 3: Enumerate the Important Terms in the Ontology

This step can viewed as a brainstorming activity [17] and it is useful if we could list all the words that we want to use, either in the form of a statement or explanation to the user. The following questions guide this brainstorming: What the terms would like to talk about? What properties do those terms have? What needed to say about those terms?

Step 4: Define the Classes and the Class Hierarchy

There are several possible approaches in developing a class hierarchy [18]: a top-down development process, which starts with the most general concepts and subsequent specialization of the concepts. Bottom-up starts with the most specific concepts or classes, the leaves of the hierarchy with subsequent grouping of these classes into more general concepts. Middle-out is a combination of the top-down and bottom-up approaches starts with the salient concepts first and then generalize and specialize them appropriately.

المنارة للاستشارات
www.manaraa.com

Step 5: Define the Properties of Classes (Slots)

This step used to describe the attributes or properties of the classes. These properties defined as the slots of the models [17]. Once the classes have been defined, the next step is to describe the internal structures (properties) of the concepts. Again, these should be readily available from the list produced because of Step 3 [16].

Step 6: Define the Facets of the Slots.

This step involves attaching facets to the properties, i.e., describing the value type, allowed values, the number of allowed values (cardinality) and other features that are necessary. In this way, constraints placed on the types of data that allowed.

Step 7: Create Instances

The last step allows the data to be entered and displayed. An instance (or individual) is the information that entered into the knowledge base. To create an instance the following method needs to be carried out:

- Choose a class.
- Create an instance and name instance after the source.
- Fill the slot values.

2.6. Policies

In this section we will talk about the policy concepts that we need in our study, we will talk briefly about policy represented as a constrain on the SOAP message brief creation, these

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
  <wsse:SecurityToken wsp:Usage="wsp:Required">
<wsse:TokenType>wsse:Kerberosv5ST</wsse:TokenType>
  </wsse:SecurityToken>
  <wsse:Integrity wsp:Usage="wsp:Required">
    <wsse:Algorithm Type="wsse:AlgSignature“
      URI="http://www.w3.org/2000/09/xmlenc#aes" />
  </wsse:Integrity>
</wsp:Policy>
```

Figure 2.6 An example of WS-Policy

constrains presented in our case as elements. Which help in the detection of multiple kind of Web Services attack. As an example case, we will introduce the WS-Policy. WS-Policy is a mechanisms needed to enable Web Services applications to specify their policies. It is an XML-based structure to express the policy information. In addition, it contain Grammar elements to indicate how the contained policy assertions apply. **Figure 2.6** present an example of WS-Policy, and how it`s look [7].

المنارة للاستشارات
www.manaraa.com

To make it clearer we must classify the policy terminology in a manner to make it clearer. In the following section, we will introduce the main terminology of the policy.

Policy: A collection of policy alternatives.

Policy Alternative: A collection of policy assertions.

Policy Assertion: An individual requirement, capability, other property, or a behavior.

Initiator: The role sending the initial message in a message exchange.

Recipient: The targeted role to process the initial message in a message exchange.

Security Binding: A set of properties that together provide enough information to secure a given message exchange.

Security Binding Property: A particular aspect of securing an exchange of messages.

Security Binding Assertion: A policy assertion that identifies the type of security binding being used to secure an exchange of messages.

Security Binding Property Assertion: A policy assertion that specifies a particular value for a particular aspect of securing an exchange of message.

Assertion Parameter: An element of variability within a policy assertion.

Token Assertion: Describes a token requirement. Token assertions defined within a security binding are used to satisfy protection requirements.

Supporting Token: A token used to provide additional claims.

Policy means that some required rule must be followed. In our approach we define some restrictions on the SOAP message. Therefore some attacks can be avoided.

The main restrictions in our approach is the existing of creation time to enable the Replay filter to check the time of the SOAP message, also the size of the message, one of the main important element of the checking is the input parameter type the Web Services expected. Finally we check the namespace of Envelope element to clearly specify the SOAP version.

Clearly we cannot use the WS-Policy. Because the Policy focus in the way to sign or decrypt/encrypt the XML file, this kind of restrictions or assertions not needed in our approach to save the processing time.

2.7. Document Object Model (DOM)

The Document Object Model (DOM) is a programing interface for HTML, XML and SVG documents. It provides a structured representation of the document (a tree) and it defines a way that the structure can be accessed and updated using programs and scripts. So that they can

المنارة للاستشارات
www.manaraa.com

change the document structure, style and content. The DOM provides a representation of the document as a structured group of nodes and objects that have properties and methods.

The DOM is a standard of the World Wide Web Consortium (W3C), it is used basically in navigating the document as a tree manner. The DOM has its own interfaces for handling documents. Many released has been published for the Document Object Model (DOM) the latest version has released in April 2004.

The DOM support many technique such as event model, XML namespaces, CSS, XPath, jQuery, JDOM, and AJAX.

2.8. SOAP with Attachment API for Java (SAAJ)

SAAJ message API follow SOAP Standard, which present a predefined some specification that are required, optional, and not allowed in SOAP specification. Using the SAAJ you can create and XML format conforming the SOAP 1.1 or SOAP 1.2 specification and the WS-I basic profile 1.1 simply by making a Java calls.

SAAJ API have its build in classes and methods which allow us to create and manipulate the SOAP message. These classes and methods handle the SOAP message object, for example the SOAPEnvelope class used to create the Envelope in the SOAP message. Then we can instance the SOAPHeader, and SOAPBody form the SOAPEnvelope object.

The SAAJ allow us to create two main kinds of SOAP message: SOAP message with attachment, and SOAP message without an attachment. The difference between this two kinds is creating the Attachment-part node. Also the SAAJ inherit the Node class from the org.w3c.dom package. in real it is not the node class alone, it also inherit the SOAPElement interface from the org.w3c.dom.Element interface, in addition the SAAJ implement the SOAPPart class form org.w3c.dom.Document interface, furthermore it extends the Text interface from org.w3c.dom.Text. This inheritances allow the SAAJ to implement the advantages of the DOM (Document Object Model) advantages.

In the other hand the SAAJ make the creating of the SOAP connection easier depending on SOAPConnection object, which is responsible for the sending and receiving of SOAP messages using the call method. **Figure 2.7** represented the way to create point-to-point connection using the SOAPConnection. The connection using the SAAJ API also called request-response, because the call method specify the endpoint for the SOAP message.

```
SOAPConnectionFactory factory = SOAPConnectionFactory.newInstance();
SOAPConnection connection = factory.createConnection();
//create a request message and give it content
java.net.URL endpoint = new URL("http://fabulous.com/gizmo/order");
SOAPMessage response = connection.call(request, endpoint);
```

Figure 2.7 Create SOAPConnection using SAAJ API

المنارة للاستشارات
www.manaraa.com

2.9. XML Path Language (XPath)

XPath is a language used for selecting parts of an XML document for subsequent processing. The main important thing in XPath is an XPath expression which allow the user to describe, in a formal way that a computer can process easily, certain parts of a document. In addition to defining specific parts of a document, XPath can also manipulate the data it returns. Although XPath used as a helper language, or ancillary technology, to identify parts of a document that will be manipulated by other languages.

The use of XPath components (path expressions, predicates, and functions) depends on two key concepts, node and sequences. A node is a piece of information in the XML tree, XPath define seven types of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes. XPath define relationship of node under five relationships: parent, children, sibling, ancestors, and descendants. The sequence is where this piece of information exist in the XML tree.

The w3school define the XPath as:

- XPath is a syntax for defining parts of an XML document or tree.
- XPath uses path expression to navigate XML document.
- XPath contains a library of standard functions.
- XPath is a major element in XSLT.
- XPath is a W3C recommendation.

Path expression present the most important component of XPath language, XPath uses path expressions to selects nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when work with a traditional computer file system. **Figure 2.8** illustrate an example of path expression.

```
XPathFactory xPathfactory = XPathFactory.newInstance();
XPath xpath = xPathfactory.newXPath();
XPathExpression expr =
xpath.compile("/Envelope/Header/TimeStamp/expire/text()");
Object result = expr.evaluate(doc,XPathConstants.NODESET);
```

Figure 2.8 An example of XPath Language

Figure 2.8 show the way of using XPath language embedded to java code. First we create an object of the class XPathFactory, and use this object to create new XPath, which use in XPathExpression. Finally store the value in an object after process the XPathExpression using the evaluate() method.

المنارة للاستشارات
www.manaraa.com

Functions in XPath language. XPath includes over 100 built- in functions. There are functions for string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values, and more.

2.10. Apache Jena (Jena Framework)

Apache Jena (or Jena in short) is a free and open source Java framework for building semantic web and Linked Data applications. The framework is composed of different APIs interacting together to process RDF data. It provides an API to extract data from and write to RDF graphs. The graphs are represented as an abstract "model". A model can be sourced with data from files, databases, URLs or a combination of these. A Model can also be queried through SPARQL. **Figure 2.9** demonstrated an example of using the Apache Jena.

```
FileManager.get().addLocatorClassLoader(Main.class.getClassLoader());
Model model = FileManager.get().loadModel("RDF/OWL file directory");
Reasoner reasoner = RDFSRuleReasonerFactory.theInstance().create(null);
InfModel inf = ModelFactory.createInfModel(reasoner, model);
```

Figure 2.9 An Example of Jena Apache

2.10.1 Jena Architecture:

According to the W3C recommendations, Jena API fully implements the RDF specification. It does so in a well-designed manner depending on three main layers [19]: **Figure 2.10** show the structure of these layers.

المنارة للاستشارات
www.manaraa.com

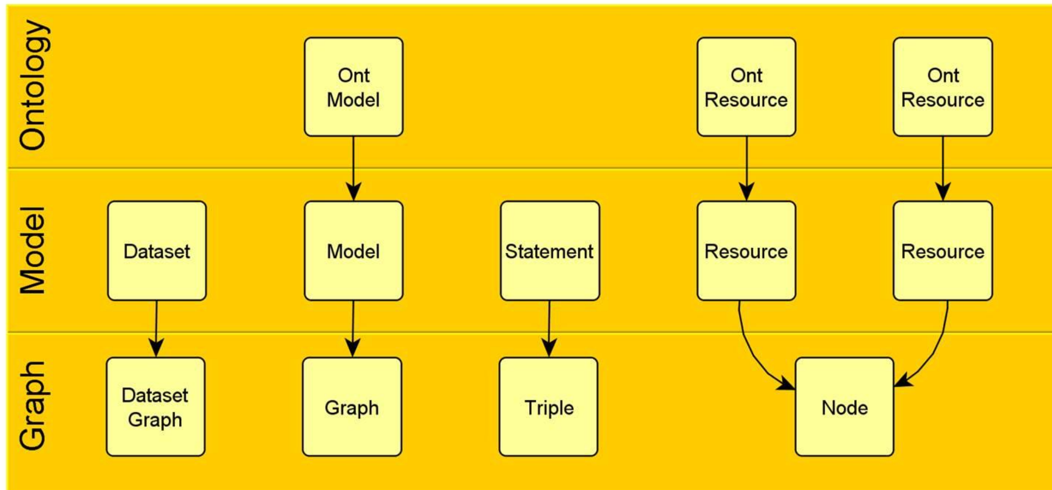


Figure 2.10 Jena Architecture [52]

1. Graph Layer

The Graph layer is very granular and is a very minimal implementation of the RDF specification. There are less methods available, and certain techniques such as reification, are not possible.

The Graph layer is really an SPI (Service Provider Interface); a description of classes/interfaces/methods to implement and extend. The heart of the Jena2 architecture is the Graph layer containing the RDF graph. This layer is minimal by design, possible functionality is done in other layers. This permits a wide range of implementations of this layer, such as in-memory or persistent triple stores. The EnhGraph layer is the extension point on which to build APIs. Within Jena2, the functionality offered by the EnhGraph layer is used to implement the Jena Model API and the new Ontology functionality for OWL and RDFS.

By separating the Graph layer from the Model layer, the Jena development team has made it very easy to extend the base Jena functionality. So, if you want to extend the Jena API, or use the Jena API in your own product, then you extend the Graph interfaces. For example, DB2-RDF support in DB2 Galileo extends the Jena Graph layer. And once you do that, you'll end up with something resembling the Model layer. The Model layer is simply the official Jena extension of the Graph layer.

Now if you're a developer, and you're working with the Jena, using either in-memory datasets or a TDB-backed dataset, you'll typically be working at the Model layer. There's really no reason to work with the Graph layer on a day-to-day basis. No benefits in terms of performance or convenience.

Figure 2.11 illustrates creating process of a triple in the graph layer:

```
Node s = Node.createURI("Rational Software Architect");
Node p = Node.createURI("hasAcronym");
Node o = Node.createURI("RSA");
Triple triple = new Triple(s, p, o);

graph.add(triple);
```

Figure 2.11 An Example of creating a triple in Graph Layer

للاستشارات
www.manaraa.com

This code creates an instance of a Triple object, populated with a subject, predicate and object. Each of these latter three are instances of the Node object. The triple is then added to the RDF graph.

2. Model Layer

Figure 2.12 shows creating a triple in the model layer:

```
Resource s = model.createResource("Rational Software Architect");
Property p = model.createProperty("hasAcronym");
Resource o = model.createResource("RSA");
model.add(subject, predicate, object);

Statement statement = model.createStatement(subject, predicate, object);
```

Figure 2.12 Creating a Triple in Model Layer

So what's different about these two techniques? At a glance, there's roughly the same amount of code, and neither the Model nor the Graph layer seem to offer any more or any less in terms of either complexity or simplicity. However, the Model layer has extended the core functionality in the Graph layer in such a way, that by allowing developers to work with objects of type "Resource" or "Property" or "Statement", instead of "Node" or "Triple", there are simply more methods available in the Model layer objects that are convenient.

3. OntModel Layer

The third and final layer in the Jena API is the OntModel (Ontology Model) layer. This gives you inference capabilities. That is, the ability to work with triples that are implied, in addition to the triples that have been explicitly defined. So, if you have a predicate in your RDF model, and it's defined as transitive, given these explicit relationships:

A -> B and B -> C

This relationship would be implied:

A -> C

The OntModel would be one way to arrive at this deduction. And it would do that by relying on an automated reasoner responsible for finding implicit triples from explicit information.

The Jena Ontology API gives an overview of Ontologies and reasoning support in the Jena API.

An enhanced view of a Jena model that is known to contain ontology data, under a given ontology vocabulary (such as OWL). This class does not by itself compute the deductive extension of the graph under the semantic rules of the language. Instead, we wrap an underlying model with this ontology interface that presents a convenience syntax for accessing the language

المنارة للاستشارات
www.manaraa.com

elements. Depending on the inference capability of the underlying model, the OntModel will appear to contain more or less triples. For example, if this class is used to wrap a plain memory or database model, only the relationships asserted by the document will be reported through this convenience API. Alternatively, if the OntModel wraps an OWL inference model, the inferred triples from the extension will be reported as well.

For example, assume the following ontology fragment:

:A rdf:type owl:Class .
:B rdf:type owl:Class ; rdfs:subClassOf :A .
:widget rdf:type :B

In a non-inference model, the rdf:type of the widget will be reported as class :B only. In a model that can process the OWL semantics, the widget's types will include :B, :A, and owl:Thing. Note that OntModel is an extension to the InfModel interface. This is to support the case where an ontology model wraps an inference graph, and we want to make the special capabilities of the InfModel, for example global consistency checking, accessible to client programs. Since not all OntModels use a reasoner, using these methods may result in a runtime exception, though the typical behaviour is that such calls will be silently ignored.

2.11. Log File

The Logging package is an ultra-thin bridge between different logging implementations. A library that uses the commons-logging API can be used with any logging implementation at runtime. Commons-logging comes with support for a number of popular logging implementations, and writing adapters for others is a reasonably simple task. As an example of the logging API we want to present log4j as an example.

Log4j is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License.

Log4j has been ported to the C, C++, C#, Perl, Python, Ruby, and Eiffel languages.

Log4j is highly configurable through external configuration files at runtime. It views the logging process in terms of levels of priorities and offers mechanisms to direct logging information to a great variety of destinations, such as a database, file, console, UNIX Syslog, etc.

2.11.1 Log4j has three main components:

Loggers: Responsible for capturing logging information.

Appenders: Responsible for publishing logging information to various preferred destinations.

Layouts: Responsible for formatting logging information in different styles.

2.11.2 Log4j Features

- It is thread-safe.

المنارة للاستشارات
www.manaraa.com

- It is optimized for speed.
- It is based on a named logger hierarchy.
- It supports multiple output appenders per logger.
- It supports internationalization.
- It is not restricted to a predefined set of facilities.
- Logging behavior can be set at runtime using a configuration file.
- It is designed to handle Java Exceptions from the start.
- It uses multiple levels, namely ALL, TRACE, DEBUG, INFO, WARN, ERROR and FATAL.
- The format of the log output can be easily changed by extending the Layout class.
- The target of the log output as well as the writing strategy can be altered by implementations of the Appender interface.
- It is fail-stop. However, although it certainly strives to ensure delivery, log4j does not guarantee that each log statement will be delivered to its destination.

2.11.3 Pros and Cons of Logging

Logging is an important component of the software development. A well-written logging code offers quick debugging, easy maintenance, and structured storage of an application's runtime information.

Logging does have its drawbacks also. It can slow down an application. If too verbose, it can cause scrolling blindness. To alleviate these concerns, log4j is designed to be reliable, fast and extensible.

Since logging is rarely the main focus of an application, the log4j API strives to be simple to understand and to use.

2.12. Summary

In this chapter, we discussed the main technical foundations that we used in our proposed approach. First, we defined the main technical terminology that our approach depend on. The Web Services and their components the relation between them recognized and classified. Then we defined the SOAP message as a lightweight transmission protocol and it`s structure. In addition, we give a brief defined of Web Services attack that our approach handle. Also, we presented the Ontology concepts and the main languages used for describe the RDF and the RDF Schema and focus in the main steps needed to develop an ontology. Moreover, how to recognized the domain and range of the system through asking specific question, in addition how to identify the classes and properties of the ontology. Additionally, we introduced the WS-Policy as an example of the policy restriction and the main terminology of the WS-Policy, and why this standard cannot take place in the approach.

Also we present the main APIs that our proposed approach used in the implementation process. Firstly we introduce the Document Object Model (DOM) API, and how it present the XML

المنارة للاستشارات
www.manaraa.com

document in a tree manner. Then we present the SOAP with Attachment API for Java (SAAJ), which is an API for creating and manipulating SOAP message and SOAP connection. After that we give a brief introduction about the XML Path Language (XPath), and how do we use it to navigate the XML document and get information about piece of XML document. Then we discuss Apache Jena or Jena API for extracting data from and write to RDF graphs, and introduce an example of calling the RDF graph, also we present it architecture. Finally we present the Logging strategy, and take the Log4j Apache as an example.

المنارة للاستشارات
www.manaraa.com

Chapter 3 Related Works

Security of Web Services is an important issue. According to a survey of senior IT executives sponsored by computer Associates [1], 43% of the respondents answered that security is the most significant issue for the deployment of Web Services. This result classifies the security of Web Services as a hot study area. Many approaches have been proposed to handle the security in Web Services. XML rewriting attacks are used [3] as a baseline for the attacks targeting SOAP messages. According to [20], the threats can be categorized into four main classes according to their consequences: (i) Disclosure: unauthorized access to data, (ii) Deception: the provision of false data which is believed to be true, (iii) Disruption: preventing system from correct operation, and (iv) Usurpation: leads to losing control of the assets to an unauthorized entity. Threats to SOAP massages can be in any of these classes.

In reviewing related works, we review the main approaches for handling XML rewriting attacks followed by Denial of Services attacks (DOSs) since we aim to handle them in our approach. We classify XML rewriting attacks into three main approaches. Then we introduce Denial of Service attacks.

3.1. XML Rewriting Approaches

Multi detection techniques and approaches are proposed to solve XML rewriting attacks, this attack is a general name for a distinct type of attacks usually they depend on the malicious interception, manipulation, and transmission of SOAP message. Where the attacker aim to modify SOAP message. This modification classify as XML rewriting attacks. The XML digital signature did not handle these kind of attacks according to [3]. Because the attacker can modify the real position of the element without violating.

In order to take a deep look in XML rewriting detection approaches we classify they into three categories as follow:

3.1.1. Inline Approaches

SOAP Account, [21] [22] the authors proposed an inline approach that takes into account information about the structure of a SOAP message and adding it to new header element calls SOAP Account element. SOAP Account contains number of envelop child, number of header element, and number of reference of signing elements, predecessor, successor, and sibling of the signing elements. The SOAP Account signing using X.509 certificate. Each node sign its own SOAP Account and concatenated to the previous node. Although this approach detect and prevent some kind of XML rewriting. But it is also vulnerable to other kind of XML rewriting attacks.

This according to [23] the SOAP Account approach cannot address all kind of XML rewriting attacks, in addition they present some ideas to fix the approach but they did not present a

المنارة للاستشارات
www.manaraa.com

complete solution. Also it is [22] vulnerable. Because it handles the Message ID and Timestamp elements as optional elements, so replay attack can take place, in addition it does not include any mechanism that can uniquely identify the parent of the signed element.

RewritingHealer used in [6] the authors attach a new element to the header containing information about the structure of the SOAP message. They added new characteristics such as; depth, parent name and parent ID. Although this new approach handle multiple kind of XML rewriting attacks. But it still vulnerable to other kind. Where the main limitation of this solution, that the attacker can move the signed elements together with its parent without violate the depth.

Position of signed element [24] they used two methods, one to setting the element position and the other to check the signed element position, this depending on post-order traversal by visiting left node first, then right node, and finally root node, this mechanism in specifying the signing element position suffer from the absence of uniquely identifier to the parent of the sign element [21].

3.1.2. Policy Approaches

In general, usage a Web Services provides services to clients. On the other end, a client requests a service via a particular application communicating directly to the Web Services or a web browser connecting to the Web Services via AJAX (Asynchronous JavaScript and XML). In order to protect SOAP messages from unauthorized parties to view or alter the messages, Web Services has made the security a key element in open architectures.

However, basic Web Services specifications themselves do not address any security topics. Several additional specifications as WS-Security [2], WS-SecurityPolicy [7], WS-Trust [25] and WS-SecureConversation [26] are proposed.

WS-Security [2] is the security mechanism for Web Services working in message level. It relies on digital signature and encryption techniques to ensure that messages secured during transmission. Digital signature provides data integrity or to proof authenticity to the communications by using hash algorithm whereas encryption process offers data confidentiality to the messages. In WS-Security case, encryption method can be implemented by either symmetric or asymmetric method.

The main limitation of the WS-Security [2] according to [3] is handling the signed element real position, where the XML signature locate the element by references. And this reference cannot locate the real position of the signed element, we must here explain the idea of the WS-Security specification in specifying the position of element it depend on references, this references can be changed and the verification process can’t detect this change. This mean that the validation of the signature did not violate after the real position changed.

Web Services Security Policy [7], if used correctly, can prevent such attacks. However, it is quite difficult to specify all possible security requirements in the WS-Security policy file. The author and the implementer of the security policy needs to be very careful in writing and implementing

المنارة للاستشارات
www.manaraa.com

the policy, also we must say that the specification file that associated with the policy is hand writing, here is the problem may the receiver misunderstand the requirement of the sender. In addition, it is vulnerable to the same limitation as WS-Security [2].

Despite all the Web Services policy standards, SOAP message still suffering from vulnerable to class of attacks. In [3] the author demonstrated the ability of changing the position of signed element by XML digital signature without violating the validation. In this paper, author present many example of XML rewriting attacks, also they explain how to use security policies correctly in order to prevent attacks. Security policies are configured by showing that more sophisticated XML rewriting attacks may avoid them, and by then improving the policy.

3.1.3. Other Approaches

A new model have been proposed in [27], their proposed model avoid XML rewriting attacks and ensures secure conversation. The model highlighted three possible recommendations namely. Using share key for encrypting timestamp in the message body for generating corresponding signature. Secondly, using value referencing for both signature validation and message processing. And finally encrypting the whole SOAP body instead of sending an open SOAP message in the network to prevent unauthorized access, the main limitation in this solution that it Consume time and resource in encrypt and decrypt the SOAP message beside they rely on shared key, how this key will be shared without the knowledge of the attacker.

FastXpath [28], used subset of Xpath instead of ID attributes to point to the signed sub tree. This solution is not flexible it limits the abilities of defining signature references.

Context-sensitive signature [29], the formal solution proposed here is a context-sensitive XML signature. To address the additional requirements of SOAP extensibility model, where a SOAP message can pass through several intermediaries before reaching the final receiver, an adaptive variant of context-sensitive signature is proposed in their solution. The tradeoff is that the context is to be generated and stored in the reference element of the signature in header section before signing the message.

A schema solution introduced in [30]. The author tries to define the Web Services attacks and proposed schema that should reflect structure for all possible genuine requests. Hence, they proposed a new self-adaptive schema-hardening algorithm to obtain fine-tuned schema that can used to validate SOAP message more effectively.

An ontology based approach has been presented in [31] [32] [33]. They proposed new approach using ontology, by build the SOAP message using ontology then attach it to the header of the message, also all modifications are written to log file. In [5] the same author makes a survey on the rewriting attack. The main limitation on this study is it Exhaust the memory and the CPU.

An adaptable framework [34] is proposed for applying agents, data mining, and fuzzy logic techniques to compensate for differences between anomaly and signature based detection for

المنارة للاستشارات
www.manaraa.com

Web Services. According to [7], these techniques allow for decision making in uncertain and inaccurate environments, but no concrete results provided.

An issued of [35] cloud security problems has been introduced, they classify it to four main classes, which are XML signature element wrapping, browser security, cloud malware injection attack and flooding attacks. In addition, they gives the possible countermeasures for each attacks, the limitation is that each countermeasure separated from other and there is not a complete system or model contains all the countermeasures.

A survey has been presented in [36] the author presents and classify all Web Services potential threats and vulnerabilities and gives an idea about there after effects on architecture behind. Also in [37] the author has presented a critically analyzed few server SOA security threats and their implications on SOAP Web Services based on literature study and their real-time experience.

A common classification and description depending in ontology has been introduced in [38]. They try to classify and describe a common vocabulary for firewall/intrusion detection systems vendors, this vocabulary based on ontologies, which can be define in OWL/OWL-S. The advantages of this solution is that the security issues become a public and common terminology and vulnerable.

3.2. Denial of Services Approaches

Denial of Service attack is the attack which targets the server behind the web services and aimed at the request processing capability of the processor [39].

The idea of DOS is to manipulate the request in a way increasing the processing resources, which cause the server of the web service to halt or denial legal user of the service. The next section represent the main DOS types handling in our proposed approach and the detection techniques used to detect each type.

3.2.1. Oversized Payload Approaches

Oversize payload attack aims to exhausting the resources of the web server. Against web services, this is attack is easy to mount due to high computation cost of xml processing. The reason behind this is that Web Service used tree in processing XML, such as DOM (Document Object model).

Using this model mean read the XML document, parsing it, and converted it into in-memory object representations. In case that the attacker replicate a single tag thousand time. In this case the server halt or more resources exhausting.

A new strategy-based detection system implemented in [40]. They describe an XML injection strategy-based detection system, XHDS, to mitigate the time gap for 0-day attacks resulting from an ontology`s attack variations. Because many new and unknown attacks derived from known strategies, they present XHDS as a hybrid approach that support knowledge-based detection

المنارة للاستشارات
www.manaraa.com

derived from a signature-based approach. Then they apply ontology to design the knowledge database for XML injection attacks against Web Services.

The approach limitations are focusing only on one kind of SOAP message attacks and ignores the other kind, this solution cannot handle new injection attacks with new strategies.

The most perfect countermeasure for oversized payload attack is to put a checker checking the size of the SOAP message [8].

3.2.2. Parameter Approaches

Parameter Tampering is a kind of XML injection attack in which the attacker modifies the parameters in a SOAP message in attempt to bypass the input validation in order to access the unauthorized information [41]. An attacker sends the SOAP message where the field values are different what the server expects. In SOAP the parameter can take the form of XML Elements. The server has XML Schema defined with a set of rules and restrictions regarding the input from the client. In parameter tampering attack the attacker violates this schema or provide oversized payload also known as XML Bombs. The Result is severe like Denial of Service is most common, and also information Disclosure, etc.

One of the main popular solution for this type of attack is [42] Input Filtration. This solution test the input of the users to remove characters like ‘,",/,--,<script>. This solution ignore the input type which is the most influent in causing this attack.

Also [8] proposed a solution which filter the input type, and check its type with the expected type.

The only way to test such kind of attacks is simulation of attacks by automation tools. The preventive measures are strict validation of SOAP message for input parameters and other attachments.

3.2.3. Coercive-Parsing Approaches

Any web services need to parse the SOAP message to get the parameter out of the request and transforming the request to make it accessible to the other application programs and this kind of processing leads a another kind of possibility for the Denial of Service attack known as Coercive Parsing [43].

In this attack a sequence of opening tags belonging to the different namespaces is created and this reduces the response efficiency to the web server to the considerable amount and further leads to the Denial of Service.

Such kinds of attack are very difficult to counter since XML does not provide any restriction for the declaration of the namespaces used in the XML Document and also no limit to the number of the External references used in the XML document, and if the referenced documents are also deeply nested then it becomes a lot more difficult to counter. But, still such kind of attacks can

المنارة للاستشارات
www.manaraa.com

be
mitigated to some extent by using Schema validation by providing the restriction to the number of
elements regarding particular namespaces in the XML Schema Definition (XSD) file [42].

In [44] proposed intrusion tolerant approach to avoid deep-nested XML. This approach consist of monitor to monitor both the attack and intrusion symptoms. If an attack is detected. The approach verified whether the intrusion symptoms appear on the target system or not. By other word, only if the attack succeeds, an altar is triggered and reaction is performed.

3.2.4. Replay Approaches

In this attack the attacker repeatedly sends SOAP Message request (the same request) to the web services in order to crash the system or restart. The target behind such type of attacks is to overload the web services or to gain access to system by posing as a legitimate user.

Many solution has been proposed to detect and prevent such type of attacks. Author in [8] proposed a filter that check the signature of the MessageID and Timestamps element of each receiving message, if the filter find any violation to the signature SOAP message reject, other way it pass. This solution is vulnerable to the XML rewriting attacks. Where the attacker can add fake MessageID and Timestamp element without violating the signature validation.

So to avoid the replay attack you need to add timestamps and random session IDs, and make sure that the attacker will not be able to manipulate.

3.3. Overcoming Limitations

3.3.1. First XML Rewriting attacks Overcoming limitations

To determine the limitations of each approach we decide to categorize them into three main groups; such as the policy-based approaches, the inline approaches, and others. In addition, specify each group's limitations. If we use policy-based approach correctly, we can prevent XML rewriting attacks by enforcing the location of the signed element in policy files in addition the other attack can be detected also. However, the policy-based approach is not efficient.

In order to detect attacks of element deletion with policy-based approach, every element should be declared as mandatory. This reduces the flexibility of the XML document and causes the performance degradation in the validation phase.

Inline approach can overcome pitfall of policy-based approach, however, is not efficient too. This is due to a high complexity of this approach, which needs a long calculation time in order to determine the structural information.

The other approaches we have to introduce each approach and it limitations.

In order to mitigate XML rewriting attacks, we can analyze detection techniques according to the following abilities.

المنارة للاستشارات
www.manaraa.com

- Detection techniques should have detection ability that indicates their ability to catch various kinds of Web Services attacks.
- Efficiency of these detection techniques is important as well. Because they have not only detect attacks, but do so in a timely manner.
- Recovery ability means to catch XML rewriting attacks and recover from them.
- Detection technique should be adaptable to various XML rewriting attacks. It has to have learning ability, which learns attack patterns and adapt to newly introduced attacks.
- The main reason why SOAP messages used in Web Services is their flexibility. This provided by the XML-syntax. Thus, detection techniques should be able to conserve this flexibility.

3.3.2. Second Denial of Services (DOS) Overcoming Limitations

In order to mitigate DOS attacks, we can classify the best countermeasure for each attacks depending on the best practice which preserves integrity, confidentiality and time consuming.

- Oversized Payload countermeasure is to define a size to check the size of SOAP message.
- Parameter-Tampering best preventive measures are strict validation of SOAP message for input parameters and other attachments.
- Coercive-Parsing such kind of attack can be mitigate using schema validation by providing the restriction to the number of elements regarding particular namespace in the XML Schema definition (XSD) file.
- Replay the perfect prevention techniques against this attack is to use the timestamps and message-id.

3.4. Summary

The efforts reviewed in this chapter related to Web Services attacks are classified into two main types: SOAP attacks which depend on XML rewriting, and SOAP attacks which target the Web Services. The works related to XML rewriting attacks are classified into: inline approaches which take into account information about the SOAP and store it in element, policy-based approaches which define all the standard used to protect the SOAP message, and approaches which depends on other techniques such as FastXPath and ontology-based. **Table 1** summarizes the limitations of the works related to XML rewriting attacks which is the main focus of our thesis.

Table 1 Summary of XML Rewriting Attacks Approaches

Attacks	Work Description	Limitations
XML Rewriting	**SOAP Account** [21]	Cannot address all kind of XML rewriting attack. Also this approach handle timestamps and message ID as an optional. In addition it does not include any mechanism that can uniquely identify the parent of signed element

المنارة للاستشارات
www.manaraa.com

	RewritingHealer [6]	The attacker can move the signed element without violate the depth. Also it vulnerable to SOAP account approach limitation.
	Position of signed element: [24]	The main limitation of this approach the ability to move signed element without violate the validation. Also time consuming.
	WS-Security [2]	Modified the signed elements without violate the signature validation.
	WS-Policy [7]	The specification is hand writing, a problem of misunderstand of the requirements.
	Recommended FastXpath [28]	FastXpath is not flexible as it limits the abilities of defining signature references.
	Suggest using Context-Sensitive Signature (CSS) [29]	The tradeoff is that the context is to be generated and stored in the reference element of signature in header section before signing the message.
	Proposed model [27]	Time consuming in encrypt and decrypt the whole message, also the share key exchange without attacker knowledge.
	Schema Solution [30]	Handling kind of XML Rewriting attack and ignoring some.
	An Ontology Based Approach [31]	Exhaust resources in the checking using the four module.
	An Adaptable Framework [34]	Make decision in uncertain environments.

المنارة للاستشارات
www.manaraa.com

Chapter 4 The Proposed Approach for Detecting SOAP Message Attacks

In this chapter, we present the approach for detecting SOAP message attacks based on a SOAP structure ontology we designed for this purpose. We start by presenting the main components and their roles in detecting SOAP message attacks. The approach depends on the ontology to check the structure of the SOAP message and makes sure that during the transmission process malicious modification has not taken place. Additionally, there are the policy filters which are responsible for checking the SOAP message coming out of the ontology checker against Web Services attacks.

First we present an overall structure of the approach, followed by building the ontology and how it is used by the ontology checker, then we introduce the policy filters and their roles in the approach.

4.1. Overall Structure of the Approach

The proposed approach consists mainly of ontology checker and the four policy filters as shown in **Figure 4.1**. It consists of the following:

- **Sender Side**: in the sender-side, after creating the SOAP message, must capture the structure of the SOAP message and add it to the log file in an encrypted manner.
- **Log File:** This log file also contains the logging event of the modification in the SOAP message during the transmission between the intermediary nodes. Then this log file is attached to the SOAP message as an attachment.
- **Ontology Checker:** The ontology checker compares the sent SOAP message structure with the predefined and standard SOAP message structure captured and defined in the ontology. Figure 4.2 is an example of the predefined SOAP message structure defined in the ontology. If the checker finds any malicious modification or missing in an important element, therefore the checker, retrieve the captured SOAP message existing in the log file and decrypt it. Also compare the log file SOAP structure with the predefined SOAP ontology. If there is a malicious missing or modification. So the SOAP message is rejected. Else, if the log file SOAP achieve the condition of the predefined, and look like it. The ontology checker send the log file SOAP to the filters as the received SOAP message.
- In the ontology, there is cases that represent all the kinds of modification or missing parts. If the SOAP message look like any of these cases so the ontology checker rejects the message.

المنارة للاستشارات
www.manaraa.com

Figure 4.3 represents the steps the ontology checker follows in checking the SOAP message structure.

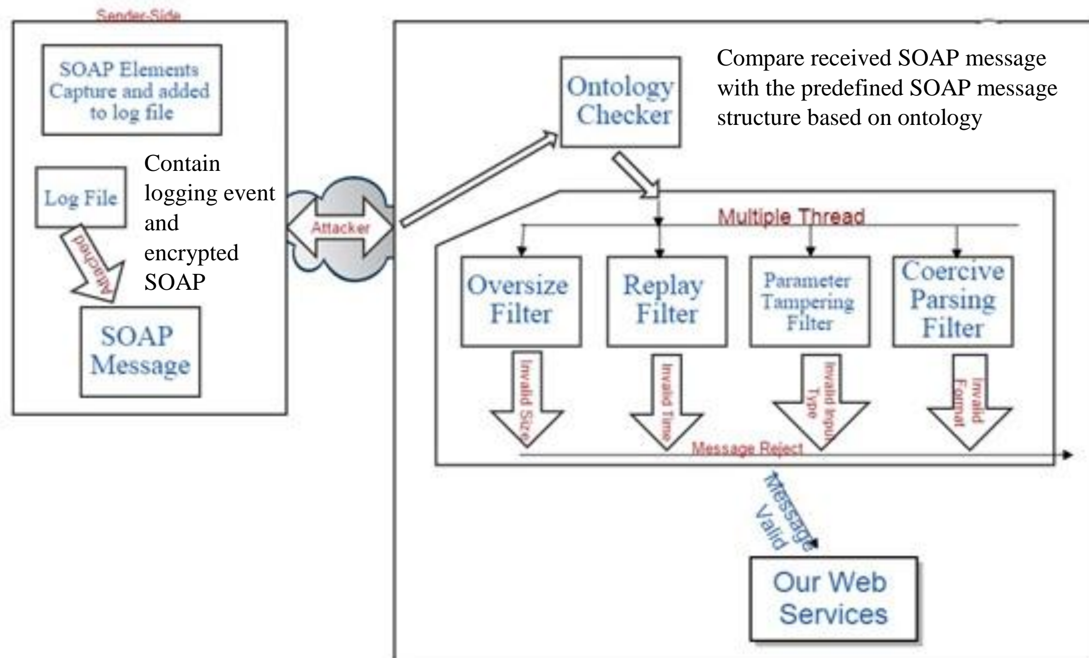


Figure 4.1 Structure of the adaptable ontology-based approach

- After the SOAP message passes the ontology checker, the filters start their checking procedure in a parallel manner. Each filter specialized in check one kind of the SOAP message attacks. **Figure 4.4** present the procedure of the filters checking. In addition, it show the condition and restriction used by the filter to check the SOAP message against the SOAP message attacks.
- **The Oversized Filter:** this filter focuses on the size of the SOAP message to avoid over flow attack and the denial of services. Therefore, this filter checks the size of the SOAP message and compares it with maximum allowed size for a SOAP message. If the SOAP message sized is more than the maximum size then the SOAP message must be rejected. Else, the SOAP message is less than the maximum size, so the filter will pass the message.
- **The Replay Filter:** this filter focuses on the element value (creation element and expire element), because the creation element consists of the time when the SOAP message has been created. Consequently the SOAP message has limited live period. This period equals the differences between expire time and the creation time, the result of the subtract process is

المنارة للاستشارات
www.manaraa.com

the living time of the SOAP message. The replay filter compare the current time with expire time. If current time is less than expire time the message must be rejected. Else, if current time is more than expire time the SOAP message is passed.

Figure 4.2 shows the structure of SOAP message captured in the ontology. Any attempt to modify the SOAP message is caught by the ontology checker.

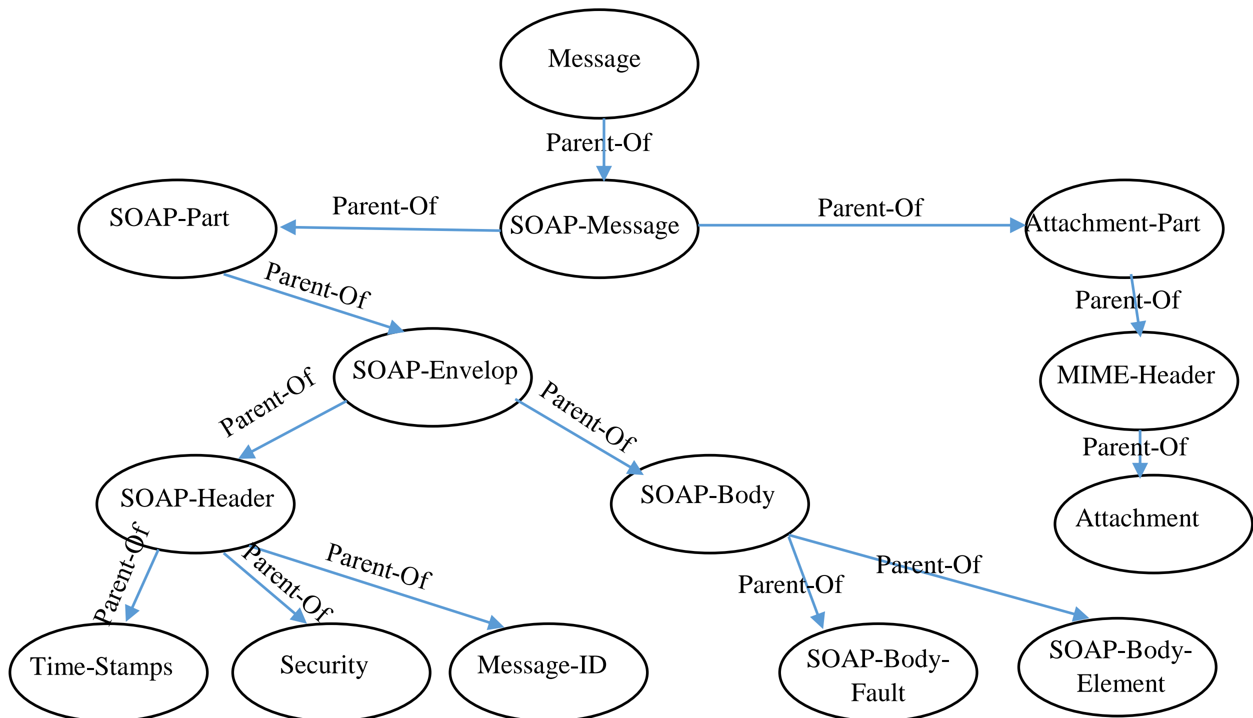


Figure 4.2 Structure of the SOAP message using Ontology

- **Parameter Tampering Filter:** this filter focuses in checking the input type of the SOAP message (Focus on the type not the value). The matter of sending input type different from the Web Services expected type. This kind of situation cause a Daniel of Services attacks (DOSs). To avoid this attack the Parameter tampering Filter check and make sure that the input type is the same as the Web Services expected as an input type. Else, the SOAP message must be rejected.

المنارة للاستشارات
www.manaraa.com

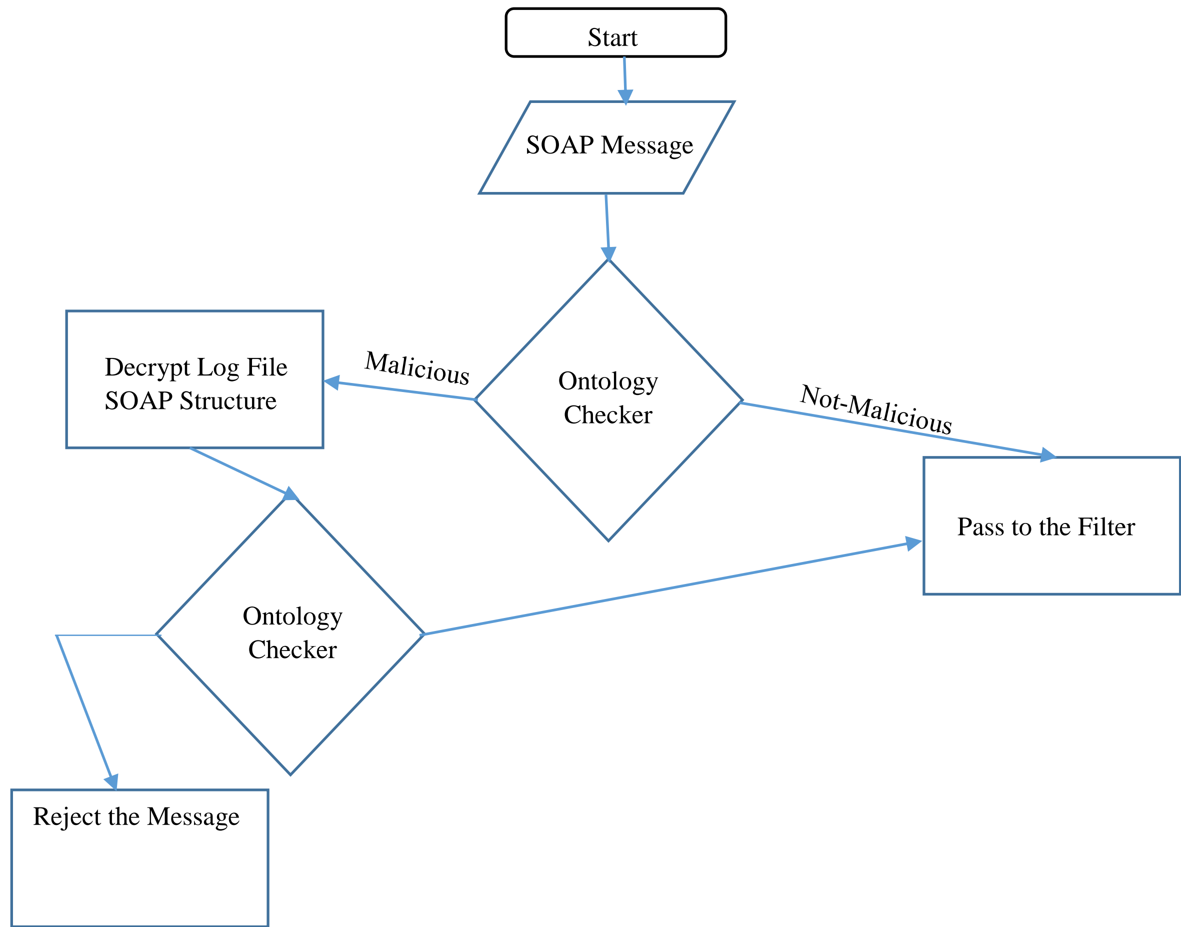


Figure 4.3 Ontology Checker procedure (flow Chart)

- **Coercive Parsing Filter:** this filter focus in checking the SOAP message format, if the format of the SOAP message is fine, the message passed to the Web Services. Else, the SOAP message must be rejected as shown in **Figure 4.4**.

المنارة للاستشارات
www.manaraa.com

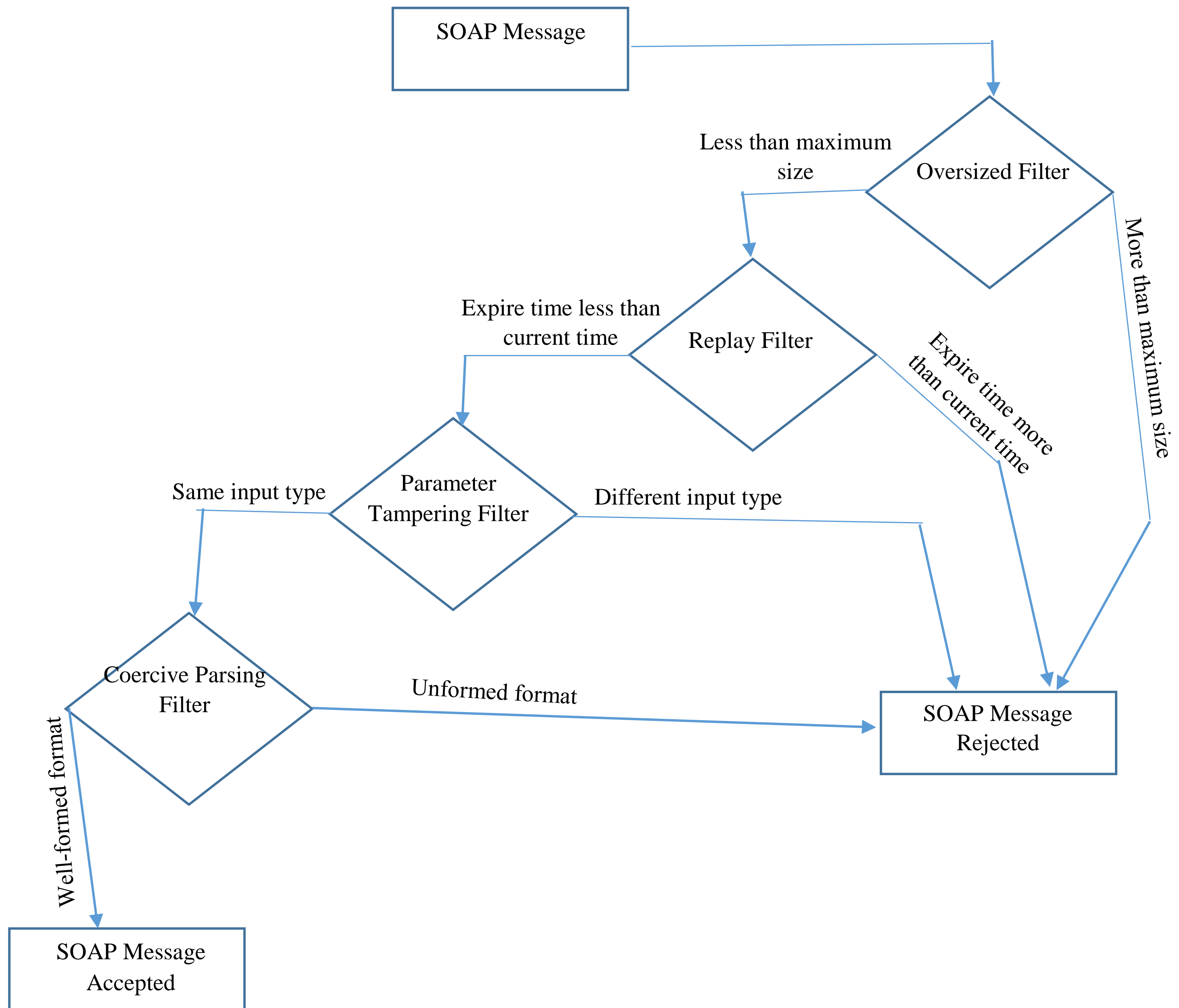


Figure 4.4 Filters procedure

4.2. The Ontology based on Predefined SOAP message Structure

In this section, we present the steps of developing the SOAP message ontology structure; this ontology must capture the structure of the SOAP message and present the main important elements of the SOAP message.

The steps of developing the SOAP message ontology structure is following the steps discussed and demonstrated in Section 2.5 Ontology Development Process.

المنارة للاستشارات
www.manaraa.com

Step 1: Determine the Domain and Scope of the Ontology

The first step in ontology development is defining ontology domain and scope, in which the ontology will be developed in order to answer some basic questions:

1) *What is the domain that the ontology will cover?*

The ontology covers and captures the structure of the SOAP message, which is a specific and limited domain serving the purpose of using the ontology.

2) *What is the use of the ontology?*

To detect any manipulation that happens to the SOAP message during the transmission process.

3) *What types of questions would be answered by the information contained in the ontology?*
   The ontology will provide the structure of the SOAP message, and detect if there any malicious modifications happens during transmission process and like

- What is the structure of the message?
- How many manipulations has happened?
- What kind of manipulation has happened?
- Is the element position re-located or re-named?
- Where the message has been manipulated, and by whom?
- What is the kind of XML attack, and how to handle it?
- Which element of the SOAP message deleted or missing?

Step2: Reuse Existing Ontologies

Existing security ontology such as [33], [32] and [31], do not separately cover the SOAP structure. Therefore, after reviewing such ontologies, we decided to design our own ontology that both covers the SOAP structures and suites the purpose of checking the validity and detection of the SOAP message within the proposed approach.

Step3: Enumerate the Important Terms in SOAP message Ontology Structure

This step represents a brainstorming activity (see Section 2.5 Ontology Development). We use the SOAP message`s important elements as the basis of the terms that we want to use. We specify the properties for a term depending on the relationship between elements and the XML rewriting attacks kind. The following questions guide out brainstorm activity to determine the terms:

1. *What are the main terms in SOAP message need to cover?*

As shown in **Figure 2.3** the main important terms is the main SOAP message elements, so the terms we cover is a SOAP-Message, SOAP Envelope, SOAP Header, SOAP Body, Message ID, Time Stamp, Security, creation, expire.

2. *What are the properties of these terms?*

- **SOAP-Message** term has the following properties: *ParentOf. hasNode*

المنارة للاستشارات
www.manaraa.com

- **Envelop** term has the following properties: *ParentOf, isExist, hasNode, and ChildOf.*
- **Header** term has the following properties: *Unique*, *isExist, ParentOf, hasNode, and ChildOf.*
- **Body** term has the following properties *Unique*, *isExist and ChildOf.*
- **MessageID** term has the following properties: *Unique*, *isExist, and ChildOf.*
- **Time-Stamp** term has the following properties: *Unique*, *isExist, ParentOf, hasNode, and ChildOf.*
- **Security** term has the following properties: *Unique*, *ChildOf*, and *isExist.*
- **Creation** term has the following properties: *Unique*, *ChildOf, and isExist.*
- **Expire** term has the following properties: *Unique*, *ChildOf, and isExist.*

Step4: Define Classes and Class Hierarchy of the Ontology

This step starts by defining classes. From the list, which created in Step 3, terms selected whether they describe objects having independent existence or terms that describe these objects. The terms in **Table 4.1** are classes in the ontology and will become anchors in the class hierarchy.

Table 4-1 Ontology Main Terms

NO.	Class	Description
1	SOAP-Message	Represent the SOAP message
4	Envelop	Represent the SOAP message Envelop
5	Header	Represent the SOAP message Header
6	Body	Represent the SOAP message Body
7	Creation	Represent the SOAP message creation time
8	Expire	Represent the SOAP message Expire time
9	Message-ID	Represent the message Id of the SOAP message
10	Time-Stamp	Represent the time of the SOAP message
11	Security	Represent the Security element
12	SOAP-Case	Represent the famous XML rewriting attacks

There are three possible ways to develop the class hierarchy [27]: top-down approach, bottom-up approach, or combination of both approaches. In our approach, we choose the top-down approach. Therefore, level concept such as SOAP- Message, Envelope, Header, and Body.

Step5: Define the Properties of classes (Slots)

Once we define the classes, we clarify and reflect the internal structure concepts. This is considered the property of the developed classes. These properties are related to the classes illustrated in **Table 4-1**. These properties are illustrated in **Table 4-2.** Each property is based on a

المنارة للاستشارات
www.manaraa.com

required and existing relationship between classes based on the standard SOAP structure. Every property has its domain and range.

Table 4-2 Ontology Object Properties

Object properties	Domain	Range	Description
isExist	Header	Envelop	Header element isExist in SOAP-Envelop
	MessageID	Header	Message-ID element isExist in Header
	TimeStamp	Header	Time-Stamp element isExist in Header
	Body	Envelop	Body element isExist in Envelop
	Creation	TimeStamp	Creation element isExist in TimeStamp
	Expire	TimeStamp	Expire element isExist in TimeStamp
	Security	Header	Security element isExist in Header
ParentOf	SOAP-Message	Envelope	SOAP-Message ParentOf (Envelope)
	Envelop	Header Body	Envelop ParentOf (Header and Body)
	Header	Message-ID Security Time-Stamp	Header ParentOf (Message-ID, Security and Time-Stamp)
	TimeStamp	Creation Expire	TimeStamp ParentOf (Creation, Expire)
ChildOf	Envelope	SOAP-Message	Envelope ChildOf SOAP-Message
	Body	Envelop	Body ChildOf Envelop
	Header	Envelop	Header ChildOf Envelop
	MessageID	Header	MessageID ChildOf Header
	Security	Header	Security ChildOf Header
	TimeStamp	Header	Timestamp ChildOf Header
	Creation	TimeStamp	Creation ChildOf TimeStamp
	Expire	TimeStamp	Expire ChildOf TimeStamp
Unique	Header	SOAP-Message	Unique in SOAP-Message
	Body	SOAP-Message	Unique in SOAP-Message
	Message-ID	SOAP-Message	Unique in SOAP-Message
	TimeStamp	SOAP-Message	Unique in SOAP-Message
	Security	SOAP-Message	Unique in SOAP-Message
	Creation	SOAP-Message	Unique in SOAP-Message
	Expire	SOAP-Message	Unique in SOAP-Message
hasNode	Envelop	Header Body	Envelop hasNode (Header, Body)
	SOAP-Message	Envelop	SOAP-Message hasNode Envelop
	TimeStamp	Creation Expire	TimeStamp hasNode (Creation, Expire)
	Header	TimeStamp MessageID Security	Header hasNode (TimeStamp, MessageID, Security)

المنارة للاستشارات
www.manaraa.com

Table 4-3 illustrates the data properties of the proposed ontology such as the data type properties of the ontology and the relations between them and the classes in the class hierarchy.

Table 4-3 Ontology Data Properties

Data Property	Domain	Range	Description
hasValue	MessageID	Integer	MessageID element must hasValue work as identifier for the SOAP message.
	Creation	Date	Creation element must hasValue presents the creation time of the SOAP message.
	Expire	Date	Expire element must hasValue present the time expire of SOAP message.
hasDepth	SOAP-Message	Integer	SOAP-Message must hasDepth integer value to identify the depth of the element.
	Envelop	Integer	Envelop must hasDepth integer value to identify the depth of the element.
	Header	Integer	Header must hasDepth integer value to identify the depth of the element.
	Body	Integer	Body must hasDepth integer value to identify the depth of the element.
	Security	Integer	Security must hasDepth integer value to identify the depth of the element.
	TimeStamp	Integer	TimeStamp must hasDepth integer value to identify the depth of the element.
	MessageID	Integer	MessageID must hasDepth integer value to identify the depth of the element.
	Creation	Integer	Creation must hasDepth integer value to identify the depth of the element.
	Expire	Integer	Expire must hasDepth integer value to identify the depth of the element.

Step6: Define the Facets of the Slots

Slots have different facets that describe the value type, allowed values, the number of the values (cardinality), and other features of the values the slot can take. In our case most of the slot values are integer. For example, the value type of hasDepth Property is integer and value type of hasValue in Time-Stamp is Date.

المنارة للاستشارات
www.manaraa.com

Step7: Create Instances

As usual after, defining the ontology classes and properties (Slots, and Facets) we need to create individuals of the ontology. In our ontology, we present the individuals as the XML rewriting attacks cases to allow the properties of the classes to be record.

The main role of the ontology SOAP message structure is to capture the SOAP message structure, we have define cases present the types of modification or missing elements in the SOAP message, these cases define in each case on the XML rewriting attacks kind.

Figure 4.5 present the SOAP-Case class, which has six individuals. The individuals SOAP-Attack from one to five present the main kind of XML rewriting attacks, and the individual SOAP-Normal demonstrate the SOAP message structure without any malicious intent.

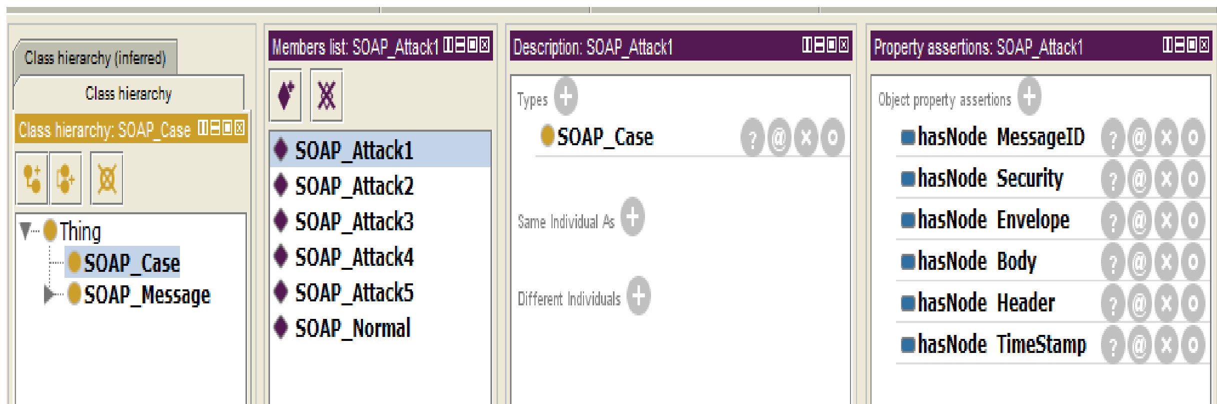


Figure 4.5 Ontology Individuals

Step8: Evaluation process

In the Evaluation Process, we aim to demonstrate that the Ontology SOAP message Structure achieving the purpose of building it. Therefor we present two main SPARQL using in the process of testing the SOAP message structure, these two SPARQL work perfectly and achieving the expectation of building the ontology SOAP message structure.

The result of using the SPARQL in **Figure 4.6**, return the child of envelope element, in the usual status the envelope element has two element. The body element as a mandatory part and the header element as an optional part, in our approach the header element is also mandatory, because it consist of element used in the detection process such as the TimeStamp element.

المنارة للاستشارات
www.manaraa.com

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>
SELECT ?a
WHERE { SOAP:Envelope SOAP:ParentOf ?a.}
```

Figure 4.6 SPARQL for Calling Envelope element child

The result of using the above SPARQL is shown in **Figure 4.6** illustrated in the **Figure 4.7**, so the body element and the header element is a mandatory child element of the envelope element in

```
SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>
SELECT ?a
WHERE { SOAP:Envelope SOAP:ParentOf ?a.}

Header
Body
```

Figure 4.7 Envelope SPARQL

our approach.

المنارة للاستشارات
www.manaraa.com

Figure 4.8 the SPARQL of Return the Timestamp child elements

The result of using the SPARQL in **Figure 4.8**, return the child of the TimeStamp element, and to make sure that our ontology is working right the child of the TimeStamp element is expire and creation. If the ontology return them then the ontology is working right.

Figure 4.9 show the result of the SPARQL in the TimeStamp element. As it demonstrate in the **Figure 4.9** the child element of the TimeStamp class is the creation class and expire class. In this case the ontology SOAP message structure implement the constrain and condition of the SOAP message structure.

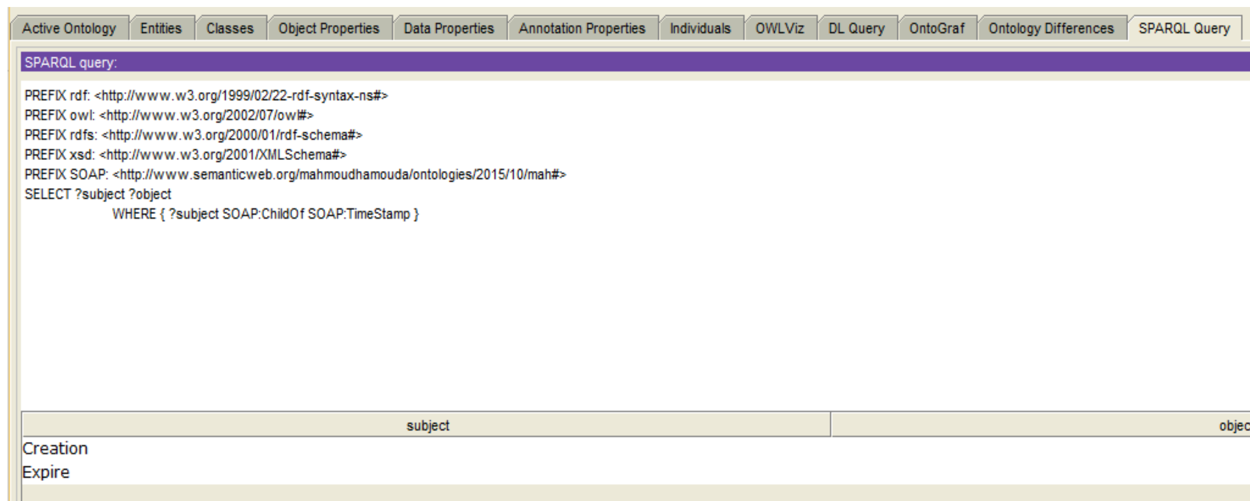


Figure 4.9 SPARQL of the Timestamp child elements

4.3. Policies determination

Policies Filter start it work after the Ontology Checker finished checking the SOAP message structure against XML rewrite attack as shown in **Figure 4.1**. It defines a framework for allowing Web Services to express their constraints and requirements. Such constraints and requirements expressed as policy assertions. This policy enforce the sender to comply these constraints and requirements. To make sure that SOAP message is free from any malicious intent. If any of these constraints and requirements missing or modified in malicious way the SOAP message will be rejected.

In our research, we intend to use the policy assertion as a policy filters to detect the Web Services attacks and reject the malicious SOAP messages we illustrate in **Figure 4.4**, if there is any fail in one filter test the SOAP message will be rejected. Our policy filter focus in four main attacks (oversized or buffer overflow attack, message replay attack, parameter-tampering attack, and coercive parsing attack). We intend to test the SOAP message against this attack in a parallel.

www.manaraa.com

The goal of handle the policies filter as a multiple thread is save the time consuming. We provide the algorithm that use to handle each filter then in the implementation chapter we illustrate how each filter works.

4.3.1. Oversized filter

In this filter, rather attack the attacker aim to insert malicious content with well-formed message in SOAP request, which is beyond the allowable size of the buffer and causes Denial of Services attack (DOS). Which cause system halt or restart. Therefore, to avoid this attack the oversized filter must measure the size of the incoming message.

The Web Services developer must estimate the maximum expected size of the SOAP message. If the incoming SOAP message is more than that expected size, message must be rejected. If the services provider do not set the maximum expected size. The SOAP message will not be more than 2048. Therefore, any SOAP message more than 2048 byte is count as a malicious intent. In the other hand the services provider can set exactly size according to his expectations. **Figure 4.10** illustrate the algorithm strategies:

1. Get XML document of client
2. Get messaging attribute
3. If maxMessagingLength = 2048 byte then process the SOAP message
4. Else discard

Figure 4.10 Oversized filter Algorithm

4.3.2. Message Replay Filter

This filter is focus in expire time of the SOAP message which exist as a Child node of the Timestamp element. Therefore any SOAP message has not expire time, must be rejected immediately. If the expire is exist, this filter bring the value of expire time and compare it with the current time. If expire time has passed and less than the current time this SOAP message must be rejected. Else, if the current time less than expire time the SOAP message will passed as a non-malicious SOAP message.

Figure 4.11 demonstrate the steps followed in the checking process of this filter.

1. Get Time-Stamp element value
2. If Time-Stamp is missing reject the message
3. Compare the Time-stamp value with the current time depending on the MaxMessageAge
4. If there is any mismatch message reject
5. Else continue

Figure 4.11 The Algorithm of Message Replay Filter

المنارة للاستشارات
www.manaraa.com

4.3.3. Parameter Tampering Filter

In the Web Services, the WSDL file contain the input data type, the attacker can use this vulnerable and sent another input type which cause the Web Services halt. To avoid this situation we need to check the input type, the checking process focus in type of input no the expected value. This checking process follow define procedure.

First, we check the input type for the null or empty value. If the input is null or empty, the SOAP message must be rejected. Else if the input type not null or empty the input must checked to make sure that this input type is the expected type. So the filter has the expected type of input and compare the type of the expected and the input. If they were different, the SOAP message must be reject. Else, they is same therefore, the SOAP message passed.

Figure 4.12 illustrates in algorithmic steps the parameter tampering filter steps.

1. Get the input data
2. If the input data equal null or empty
3. Reject the Message
4. If the input data mismatch the data type
5. Reject the message
6. Else continue

Figure 4.12 The Algorithm of Parameter Tampering Filter

4.3.4. Coercive Parsing Filter

The filter verify the received message for wrong format of SOAP message by generating SOAP fault code. This filter blocks the input that has a strange format. This policy is used the values in SOAP fault code. They are:

- Version Mismatch Fault Code: it finds invalid namespaces for a SOAP envelop and throws exception.
- Must Understand Fault Code: it indicates whether a header entry is mandatory or optional for the recipient to process.

Many of things can go wrong with the Web Services message. The Web Services may encounter a problem, input data may be wrong or a header may come across which the server does not understand. The algorithm for coercive parsing filter is illustrate In **Figure 4.13**.

1. Get XML document
2. If(SOAP version == valid namespace)
   If(mustUnderstand==1)
   Allow
   Else throw Must Understand Fault Exception
3. Else
   Throw version Mismatch Fault Exception

Figure 4.13 The Algorithm of the Coercive Parsing Filter

المنارة للاستشارات
www.manaraa.com

4.4. Summary

Previous chapter try to present the proposed approach components, functionality, and their procedure. First we gives an overall illustration of approach steps. Through this steps we illustrate the checking implementation and procedure to detect the SOAP message attacks.

Then we present the steps of building the predefined SOAP message ontology depending on Section 2.5. In this section we inferred and classified the main terms to represent the class hierarchy. Also we define the relation between these classes to define the properties.

In our proposed approach we try to capture the structure of SOAP message. And this can be achieved using ontology, and to build the prefect ontology, we must define clearly the important classes (elements) in SOAP message we need it to be exist, also define the relationship between these classes. This definition prevent SOAP message from XML rewriting attacks.

Finally we illustrate the policy filters. The manner of illustrating policy filters is an algorithmic style, which can be achieved in any programming language. We try to define exactly the role of each filter and the way these policy filters checking the SOAP message against each attacks.

المنارة للاستشارات
www.manaraa.com

Chapter 5 Implementation

In this chapter, we implement the proposed approach. First, we illustrate the SOAP message creation process and how to attach it to the log file. We use the SAAJ API in the creation and attaching process. In the second section, we present the log file identifying process, in our approach the log file main role is capturing the SOAP message structure. In the other hand the log file record each logging event happened to the SOAP message during the transmission process.

Then we present the implementation steps to realize the components of the approach. The components include the ontology checker and the filters; Oversized filter, Replay filter, parameter-tampering filter, and coercive parsing filter.

5.1. SOAP message Creation

There are various APIs for creating a SOAP message. We choose the SOAP with Attachment API for Java (SAAJ) [45]. We need to attach a file to the SOAP message, so the SAAJ allow us to do so. The SAAJ API presents, defines methods, which is responsible for creating the SOAP message part (here we present the SOAP message part such as body, and header as element).

The SOAP message specification is an extension and flexible specification, it allows the user to define his own node (element), but this SOAP message created by the user must follow some restrictions according to the SOAP message version, and the existence of the envelop and the body part as a mandatory part. These parts must exist in any SOAP message to illustrate that this XML document is a SOAP message.

In our proposed approach we focus in the main important element, so our approach handle these elements as a mandatory element, which must exist in any received SOAP message. Missing elements means that SOAP message was modified.

Figure 5.1, presents the SOAP message parts we intent to add and must exist in any SOAP message our approach is checking. Therefore, the SOAP message must include these elements.

المنارة للاستشارات
www.manaraa.com

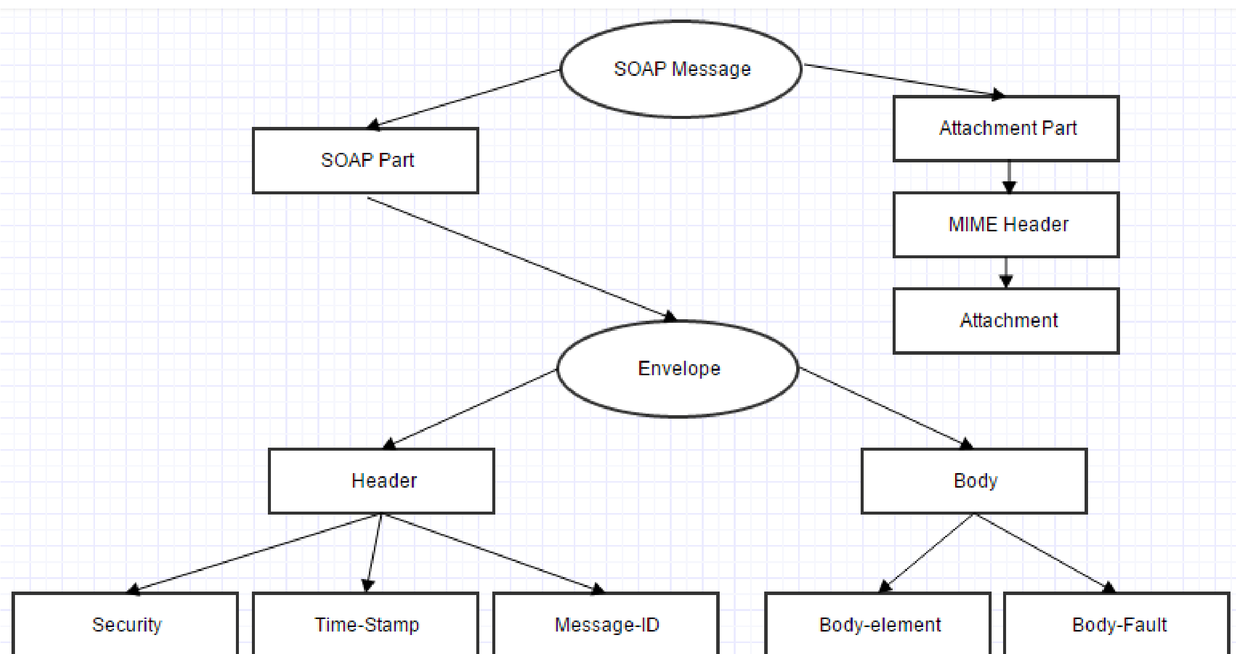


Figure 5.1 SOAP Message Element in SAAJ

We illustrate the code for creating SOAP message using the SAAJ API, so we introduce the main steps of defining and creating the SOAP message elements. Additionally, we explain how to attach the log file using the SAAJ API.

a) Create the SOAP message and the SOAP message Connection

A very important step in creating the SOAP message using the SAAJ API is using the SOAPConnectionFactory method instance() to instants the SOAP message.

Then we use the class MessageFactory and insatiate an object of this class using the instance() method, which will use by the createMessage() method to create empty SOAP message which has not any part. The SOAPConnection class use to create the SOAP message connection using the predefine method createConnection(). All these steps are illustrated in **Figure 5.2**.

```
SOAPConnectionFactory factory  =  SOAPConnectionFactory.newInstance();
SOAPConnection connection =  factory.createConnection();
MessageFactory mf =MessageFactory.newInstance();
SOAPMessage msg = mf.createMessage();
```

Figure 5.2 SOAP message Creation and Connection

b) Create the SOAP Part element (Body, Header, and Envelop)

The SAAJ API present an amazing way to create the SOAP message parts, but we must follow some steps as follow. First we need to create SOAPPart as a container for the other parts, to achieve this goal we insatiate the class SOAPPart using the predefine method getSOAPPart().

للاستشارات المنارة
www.manaraa.com

After that we create the SOAPPart object, which is used in creating the SOAPEnvelope object class using the method getEnvelope(). Then we need to call the envelope object to insatiate the header and body part using the methods (getHeader(), and getBody()).

These steps of creating an empty parts of the SOAP message illustrated in **Figure 5.3**

```
SOAPPart soapPart = msg.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPHeader header = envelope.getHeader();
SOAPBody body = envelope.getBody();
```

Figure 5.3create SOAP Part elements

c) Create the SOAP Header elements

The SOAP Header element has three elements (Security, Message-ID, and Time-Stamp) as shown in **Figure 5.1**. In **Figure 5.4** we show how to create each of these elements of the SOAP Header.

```
SOAPHeaderElement security = header.addHeaderElement(envelope.createName("Security", NS_PREFIX, NS_URI));
 SOAPHeaderElement Time = header.addHeaderElement(envelope.createName("TimeStamp",NS_PREFIX ,NS_URI));
SOAPElement creation = Time.addChildElement("creation",NS_PREFIX);
 creation.addTextNode(formatter.format(create));
  SOAPElement expire = Time.addChildElement("expire",NS_PREFIX);
  expire.addTextNode(formatter.format(expires));
  SOAPHeaderElement MsgID = header.addHeaderElement(envelope.createName("MessageID",NS_PREFIX ,NS_URI));
```

Figure 5.4 SOAP Header Elements create

d) Create the Attachment Part

The attachment file (log file) has two main roles: capturing the SOAP message structure, and registering and laying the logging events. As illustrated in **Figure 5.1**, the attachment presents the log file.

Before building the log file structure, we need to insatiate it, the log file instance is passed as input through the FileDataSource object, to enable us to use this object as an input to the DataHandler object which uses it in the AttachementPart class. Then we set the content ID of the attachment using the method setContentId() to make it easy to find the attachment. After that we add this attachment to the SOAP message using the method addAttachmentPart().

Figure 5.5 illustrates these steps, our intent here is to create and attach the log file but the structure of the logging event lay out and the capturing of the SOAP message structure is demonstrated in Section 5.2.

المنارة للاستشارات
www.manaraa.com

```
File myfile = new File("admin.log");
FileDataSource fds = new FileDataSource(myfile);
DataHandler dh = new DataHandler(fds);
AttachmentPart ap2 = msg.createAttachmentPart(dh);
ap2.setContentId("attachment");
 msg.addAttachmentPart(ap2);
 MimeHeaders headers = msg.getMimeHeaders();
String [] contentType = headers.getHeader("Content-Type");
```

Figure 5.5 Create the Attachment Part

5.2. Building Log File

We need to illustrate the log file structure and the way we build it to achieve its goals. The building of the log file focuses in two main role, first registering the logging event, secondly capturing the SOAP message structure.

First, we define the type of property we intent to use. So we use the class PropertyConfigurator predefine method configure() to indicates to the logging properties file where the logging event layout is built and specified. This step illustrated in **Figure 5.6.**

```
PropertyConfigurator.configure("loger.properties");
```

Figure 5.6 file of properties

Then in the properties file we specify and classify how the logging event layout and which logging information we are interested in. This specification is illustrated in **Figure 5.7**. In addition, the result of these specification are written in the log file.

```
log4j.rootLogger=DEBUG, file
#Log message in the admin.log file
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=admin.log
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern= %-4r [%t] %d{dd MMM YYY HH:mm:ss,sss} %-5p %c %F - %m%n
```

Second, we illustrate the way to capture the SOAP message structure. After we have created all the SOAP message elements and specification. We write this SOAP message to the log file. In this process we need to pass the log file through FileOutputStream. This step is show in

Figure 5.7 The layout of the Log file

المنارة للاستشارات
www.manaraa.com

Figure 5.8.

```
File file = new File("admin.log");
 FileOutputStream out = new FileOutputStream(file);
msg.writeTo(out);
```

Figure 5.8 Capture SOAP Structure

Figure 5.9 Illustrates an example of the resulting log file capturing the structure SOAP message and **Figure 5.10** presenting the logging event laying.

```
------=_Part_0_186370029.1446694511152
Content-Type: text/xml; charset=utf-8
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <jaxmtst:security xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
      <jaxmtst:signature>Signed Value<jaxmtst:reference>Refvalue</jaxmtst:reference>
      </jaxmtst:signature>
    </jaxmtst:security>
    <jaxmtst:TimeStamp xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
      <jaxmtst:creation>2015-11-05T03:35:05.897Z</jaxmtst:creation>
      <jaxmtst:expire>2015-11-05T03:40:05.897Z</jaxmtst:expire>
    </jaxmtst:TimeStamp>
    <jaxmtst:MessageID xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
      <jaxmtst:ID>IDvalue</jaxmtst:ID>
    </jaxmtst:MessageID>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="http://wombat.ztrade.com">
      <symbol>SUNW</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
------=_Part_0_186370029.1446694511152
Content-Type: text/plain
Content-ID: attachment 1001
```


المنارة للاستشارات
www.manaraa.com

```
Nov 2015 05:14:56,056 DEBUG test.Test Test.java - message creation
178  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - SOAPPart creation
190  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - Envelope creation
218  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - header creation
219  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - body creation
220  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - security header creation
222  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - signature creation
224  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - reference creation
224  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - time creation
225  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - messageID element creation
226  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - body element creation
228  [main] 05 Nov 2015 05:14:57,057 INFO  test.Test Test.java - attachment creation
```

Figure 5.9 An Example of Log File

Figure 5.10 represents the logging event layout. The way the logging event will register in the log

Figure 5.10 Log File Logging Event

file. This logging event specify the information we looking for.

5.3. SOAP-Message Ontology Structure Implementation

In this section, we implement the SOAP message ontology and show how it reflects the SOAP message structure based on the ontology design presented in Section 4.2. First we define the important classes and properties, then we explain the way that the SOAP message Ontology works to detect the XML rewriting attacks, and keeping the message structure as it was being sent from the original.

5.3.1. Define Important Terms (Class Hierarchy)

In this section, we illustrate the classes of the SOAP message ontology and explain the role of each class, the classes are shown in **Figure 5.11**. The OWL file presenting the ontology presented

Figure 5.11 The SOAP message Ontology Structure

للاستشارات
www.manaraa.com

in Appendix1.

The ontology as shown in **Figure 5.11** has many classes. There are two kinds the SOAP message classes which is represented in the predefined SOAP message structure. In addition, the position of these classes as super and sub classes is very important issue. Moreover, the SOAP case class, which is represented the famous kind of XML rewriting attacks as instances.

The following are the SOAP message ontology structure classes:

- **Envelope:** Envelope class represents the SOAP-Envelope element, which define the SOAP version and the name spaces used in the SOAP message. The missing of this class/element mean this XML document is not a SOAP message. So this Envelope is a mandatory element.
- **Body:** this class represent a very important element in the SOAP message. This element is SOAP-Body and it is a mandatory element, which contains the message purpose, input, or fault message information, the missing of Body element mean this XML document is not a SOAP message. Therefore body is a mandatory element.
- **Header**: Header class represent the SOAP-Header element. In our approach the SOAP-Header element is a mandatory element because it contains of sub-children define security restrictions.
- **Timestamps**: this class represent a container of two main element as it sub-classes, the creation and expire classes. This class is response of register the time of the creation and expire time of the SOAP message. So this element represents a mandatory element in our ontology.
- **Message-ID**: this element represent the identification number of the SOAP message and it is a very important element in detection duplicated message. This prevent SOAP message from replay attack. Therefore Message-ID is a mandatory element in our ontology.
- **Security**: this class represent any constrain Web Services provider want to add such as signing element. Encryption mechanism. In our ontology Security element demonstrates a mandatory element.
- **Creation:** this class represent the creation time of the SOAP message and it is a sub child of the Timestamps class. We determine the period of live of a SOAP message this element should exist. So this also a mandatory element in our ontology.
- **Expire:** as the creation class, this class is represented time where the SOAP message expire and become out of date. Moreover the absent of this element deprivation the Web Services

المنارة للاستشارات
www.manaraa.com

from the ability to check message against Replay attack. Therefore this class is a mandatory class in our ontology.

- **SOAP_Case:** this class represent the main XML rewriting attacks. The missing situation and the malicious modification.
- **SOAP_Message:** this class work as a container to the SOAP message classes.

These classes identify the main structure of the SOAP message the user can add any number of class that he wish. However, in our approach these classes present a mandatory element. The main idea is to detect the manipulation process in the SOAP message during the transmission process, the original message is exist in the log file, if the attacker change the structure of the SOAP message the ontology checker will detect that changes and return the SOAP message to the original structure.

This is the main idea of using predefined SOAP message ontology structure. Is to compare the received SOAP message structure with the hidden original SOAP message structure and detect any type of XML rewriting attacks that happened to the SOAP message during the transmission process.

5.3.2. Define Slots and Facets

In this Section, we discusses the properties of the predefined SOAP message Ontology Structure, and we illustrate the way they work together to make sure that the SOAP message has not been changed during the transmission process. In **Figure 5.12** we illustrate the main object properties in our SOAP message Based Ontology, and later we will discuss each property and it role.

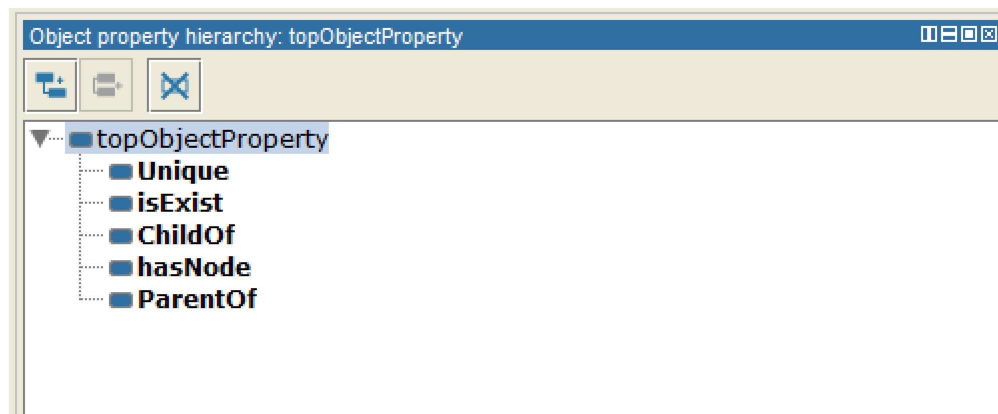


Figure 5.12 SOAP message Ontology Structure Properties

The properties in the ontology present the verb, and to make your sentences more powerful and meaningful you need to use the appreciate verbs.

The properties of our ontology is a very meaningful to describe the main relationship between the classes, let us illustrate and discuss them:

المنارة للاستشارات
www.manaraa.com

- parentOf: this properties focus on the position of the SOAP element to specify the relation of sub and super classes. As an example the Envelope class is parentOf the classes Header and Body. The **Figure 5.13** show the relation between them.

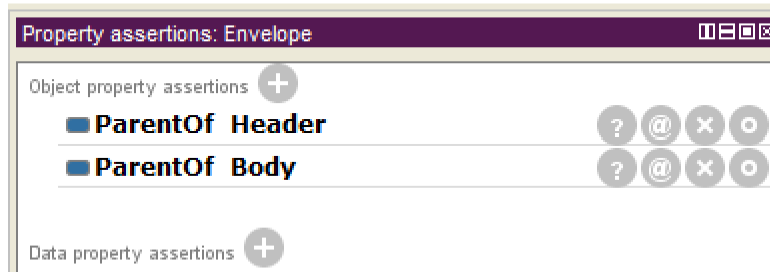


Figure 5.13 parentOf Properties Example

- childOf: this property is mapping the sub class to the super class to exactly define the SOAP message position. As an example, the Body class is a childOf the Envelope class. This example is illustrated in **Figure 5.14**. This property clarify which class is a sub class of which class.

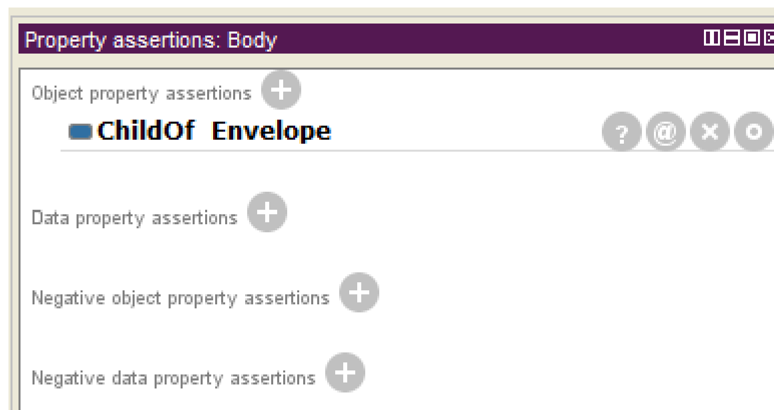


Figure 5.14 childOf Properties Example

- hasNode: this property specify which element has node as sub classes and which classes has not. The use of this property appear in the definition of the predefine SOAP attacks. When the SOAP message has not body class for example. **Figure 5.15** show the SOAP_Attack1 where are the creation and expire classes are missing.

المنارة للاستشارات
www.manaraa.com

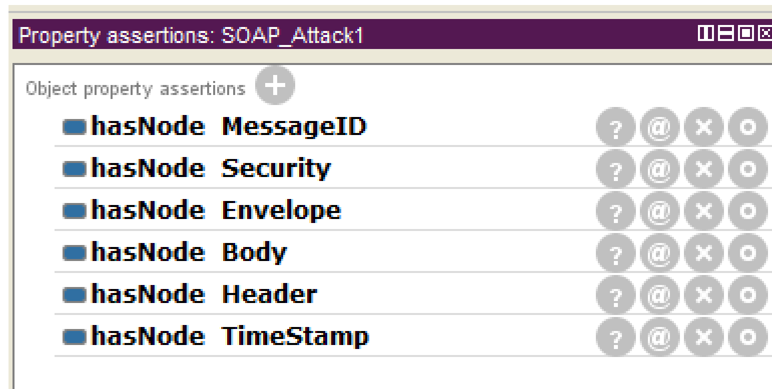


Figure 5.15 hasNode Property

- unique: this property clarify that this class never been duplicated. Example of this property is the Body class. The SOAP message impossible has two-body element. **Figure 5.16** show the property.

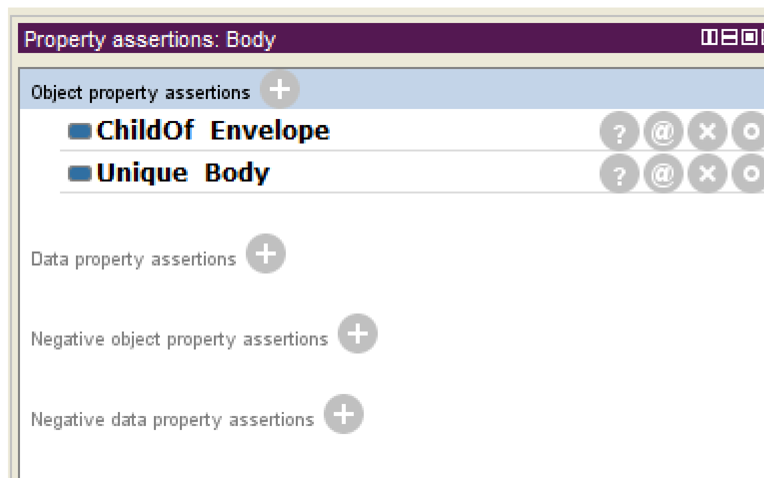


Figure 5.16 unique property

- isExist: this property clarify which classes of the SOAP message classes must be exist. Moreover the missing of this class is count a vulnerable in the SOAP message structure. An example of this property is the creation class. As shown in **Figure 5.17**. The missing of the creation time of the SOAP message reject the message.

المنارة للاستشارات
www.manaraa.com

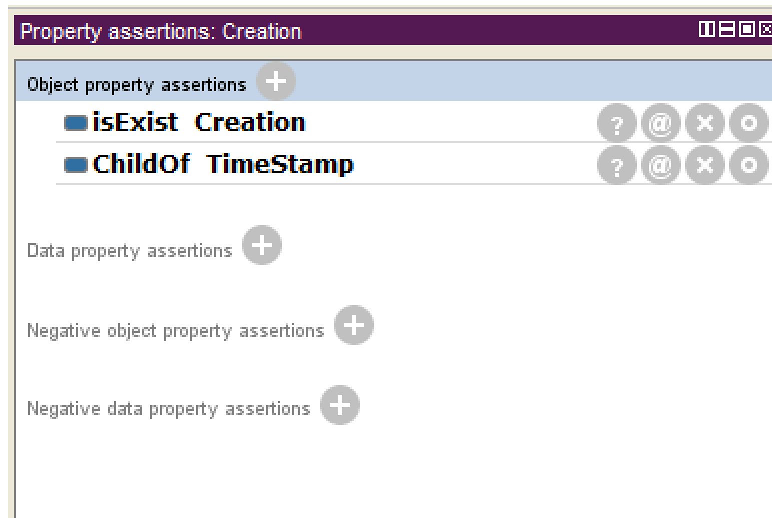


Figure 5.17 isExist Property

5.4. Detecting XML Rewriting Attacks in SOAP Message Using Ontology Checker

In this section we implement the checking process of the SOAP message structure using ontology checker component. The checking process is divided into the following steps. The entire code for checking SOAP message structure using ontology checker is demonstrated in Appendix 4.

Step 1: Get received SOAP message:

In this step, we receive SOAP message. It is received as an XML file. First we parse and restructure the message using Document Object Model (DOM) as shown in **Figure 5.18**. This step

```
File fXmlFile = new File("SOAP_xml.xml");
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(fXmlFile);
```

Figure 5.18 Prepare SOAP Message

makes SOAP message ready for using and checking.

Step 2: Check the received SOAP message based on ontology:

We call the received message from the previous step. We compare the received SOAP message structure that is captured based on the ontology at the sender side with the predefined SOAP message structure based on the ontology. This is separated into the following subtasks:

المنارة للاستشارات
www.manaraa.com

- First we make sure that all the mandatory elements exist. Therefore we query the ontology using SPARQL query that returns all elements based on the ontology. The SPARQL used for this purpose is demonstrated in **Figure 5.19**.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
"PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
"PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>\n" +
"SELECT ?a ?t ?x\n" +
"           WHERE { ?a rdf:type SOAP:SOAP_Message.}
```

Figure 5.19 Return All Elements based on Ontology

- Next we check the positions of these elements based on the predefined SOAP message structure using the ontology. To achieve this purpose, we SPARQL the ontology to check child elements existence and predefined relationship between SOAP elements. This checking procedure focuses on checking three main elements: Envelope, Header, and Timestamp. **Figure 5.20** demonstrates the SPARQL query over Envelope element to return its child elements; Header and Body.

Figure 5.21 illustrates the SPARQL query used over the Header element to return the child elements of the Header element, namely, Timestamp, Security, and Message-ID.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
"PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
"PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>\n" +
"
```

Figure 5.20 Return Envelope Sub-elements

```
"           WHERE { SOAP:Header SOAP:ParentOf ?a.}
```

Figure 5.21 Return Header sub-Elements

Figure 5.22 illustrates the SPARQL query used over the Timestamp element to return its child elements; Creation and Expire.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
"PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
"PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>\n" +
"SELECT ?a\n" +
"           WHERE { SOAP:TimeStamp SOAP:ParentOf ?a.}
```

```
           WHERE { SOAP:Envelope SOAP:ParentOf ?a.}
```

Figure 5.22 Return Timestamps sub-elements

المنارة للاستشارات
www.manaraa.com

- After performing the checking of the predefined SOAP message structure based on the ontology, we check the elements of the receiving SOAP message to insure the existence of all the mandatory elements and the correctness of their positions in the received message. **Figure 5.23** shows the way we return all the elements of the received SOAP message.
  Then in **Figure 5.24** we demonstrate the way we return the sub-elements of envelope; Header and Body. Similarly, we query all the elements in the received message.

```
DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
   DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
   Document document = docBuilder.parse(new File("xml.xml"));
     XPathFactory xPathfactory = XPathFactory.newInstance();
      XPath xpath = xPathfactory.newXPath();
      XPathExpression expr = xpath.compile("/Envelope/*");
      Object result = expr.evaluate(document, XPathConstants.NODESET);
      NodeList n1 = (NodeList) result;
       for (int i = 0; i < n1.getLength(); i++) {
         String child =n1.item(i).getNodeName();
         envelop.add(child); }
```

Figure 5.24 Return Envelope sub-elements received SOAP message

```
DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory
        .newInstance();
    DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
    Document document = docBuilder.parse(new File("SOAP_xml.xml"));
    NodeList nodeList = document.getElementsByTagName("*");
    for (int i = 0; i < nodeList.getLength(); i++) {
       if (node.getNodeType() == Node.ELEMENT_NODE) {
          String elements = node.getNodeName();
          array.add(elements);   } }
```

Figure 5.23 Return received SOAP message elements

Step 3: Restoring missing or modified elements in the received message based on the log file:

If there are missing or modified elements in the received message based on the Step 2, we retrieve the encrypted SOAP message from the log file. If all of these elements exist in the retrieved message, then we again check using the ontology checker the positions of these elements by comparing them with the elements from the predefined SOAP message structure. Any missing or modification in the encrypted message retrieved from the log file is detected, therefore the message is rejected and we don't need to proceed to the policy filters checking. But if the message in the log file is correct, then we use this message to restore the missing or modified elements on the received message. After the message is restored and corrected we pass it to the policy filters to check for Web Service attacks.

5.5. Policy Filters Implementation

We have created each filter as a separate thread using the Runnable interface in order to perform all kinds of filter checking concurrently therefore speeding up the filtering process.

المنارة للاستشارات
www.manaraa.com

5.5.1. Oversized Filter

Figure 4.10 Shows how the Oversized Filter works, first we load the SOAP message, then we calculate the size of the whole file and compare the size with the maximum allowable SOAP

```
public static double over ;
public static void main (String [] args ){
try {File file =new File("xml.xml");
over = 1800;
double size = file.length();
if (size <= over ){ String result = "Thm Message is less than the maximum Size";
JOptionPane.showMessageDialog(null, result);
}else{ String result = "Reject the SOAP message (OverSize Attack)";
System.out.println(result);
} }catch (Exception e){ System.out.println(e.getMessage());   }}
```

Figure 5.25 The Oversized Filter

message size. This implements calculate and compare step in Section 4.3.1.

5.5.2. Replay Attack Filter

In Section 4.3.2, we explain the algorithmic procedure, first we bring the expire node from the Time-Stamp. If the expire node is missing then reject the SOAP message. Then we compare its value with the current time, if the expire node value is less than the current time then the SOAP message is fine. Else, reject the SOAP message. In **Figure 5.26** we show the comparison

```
if (expireTime.compareTo(currentTime) > 0) {
        System.out.println("Expire Time is more than Current Time, Reject the message");
        System.exit(0);
    } else if (expireTime.compareTo(currentTime) < 0) {
        System.out.println("Expire time is less than Current Time");
     } else if (expireTime.compareTo(currentTime) == 0) {
        System.out.println("Expire Time is Equal Current Time, ");
    } else {
        System.out.println("Something weird happened...");
        System.exit(0);}
```

Figure 5.26 The Message Replay Attack

procedure.

5.5.3. Parameter Tampering Filter

In Section 4.3.3, we explained the algorithmic procedure, first we test the input for null and empty value then we compare the type of the input with the expected type, if they are the same then the message passed, else the message reject, these steps shown in **Figure 5.27.**

المنارة للاستشارات
www.manaraa.com

```
XPathExpression expr = xpath.compile("/Envelope/Body/GetLastTradePrice/symbol/text()");
        Object result = expr.evaluate(doc, XPathConstants.NODESET);
        NodeList n1 = (NodeList) result;
          for (int i = 0; i < n1.getLength(); i++) {
            input = n1.item(i).getNodeValue();  }
           if (input == null){ System.out.println("The input value is NULL. Reject SOAP message ");
              exit(); }else{System.out.println("the input type is:  "+input.getClass()); }
                type = "SUNW";
        if(type.getClass()== input.getClass()){
        System.out.println("the input is from the same type");
         }else { System.out.println("the type of the input is not equal "); } }
```

Figure 5.27 The Parameter Tampering Filters

5.5.4. Coercive Parsing Filter

In Section 4.3.4, we explained the algorithmic procedure, first we get the XML file (SOAP message) then get the Version of the SOAP message, and there are two SOAP version:

1. SOAP 1.1 : http://schemas.xmlsoap.org/soap/envelope/
2. SOAP 1.2 : http://www.w3.org/2003/05/soap-envelope

We must classify witch version is used in the SOAP message and work depend on the type is used.

The procedure of checking the version of the SOAP message demonstrated in **Figure 5.28**. If the name space of the received SOAP message did not match any expected one of the name spaces.

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(true);
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse("xml.xml");
        String ns = doc.getDocumentElement().getNamespaceURI();
        if (ns=="http://schemas.xmlsoap.org/soap/envelope/"){
         System.out.println("SOAP message version 1.1");}
             else if (ns =="http://www.w3.org/2003/05/soap-envelope"){
        System.out.println("SOAP message version 1.2 ");
        } else if (ns=="http://www.w3.org/2001/06/soap-envelope") {
            System.out.println("SOAP message version 1.2 ");
            }else {System.exit(0);}
```

Figure 5.28 Coercive Parsing Filter

The SOAP message is rejected.

المنارة للاستشارات
www.manaraa.com

5.6. Summary

In this chapter we have implemented the proposed approach. First we created the SOAP message using the SOAP with Attachment API for JAVA (SAAJ), and implement how we add the attachment (log file) to the SOAP message.

Then we present the steps for creating log file and build the properties file which response of displaying the logging event. After that we present the structure of the predefined SOAP message structure based on ontology using the protégé software.

Furthermore we demonstrate the checking and detection steps used by the ontology checker component, and how to check the received SOAP message structure. Finally we illustrated the policy filters, how the checking process implemented to check the received SOAP message against Web Services SOAP message attacks: Replay attack, Oversized attack, Parameter-Tampering attack, and Coercive-Parsing attack.

المنارة للاستشارات
www.manaraa.com

Chapter 6 Experimentation and Evaluation

In this chapter, we test the approach through a set of experiments. There are six test cases: normal case, modified structure case, replay case, parameter-tampering case, oversized case, and coercive case. We evaluate the approach for integrity and confidentiality and use them as a basis to compare with other related approaches. Additionally, we evaluate its time efficiency based on the concurrent execution of the policy filters.

6.1. Experimentation and testing cases

In this section we test the components of our approach, the first test case presents the normal case which the SOAP message is free of attacks. Secondly we modified the SOAP message in a malicious manner to test the ability of the ontology checker to determine that there is a tampered message. Then we illustrate the testing process for the policy filters, and check each filters against the corresponding attacks, which is four test cases.

6.1.1. Checking Non-modified SOAP message

In this test we aim to prove that the approach work fine in the normal case. So we create a SOAP message without any malicious content to demonstrate that approach work fine. The testing SOAP message is presented in **Figure 6.1** and the result of the checking process is shown in

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <jaxmtst:security xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
   <jaxmtst:signature>Signed Value<jaxmtst:reference>Refvalue</jaxmtst:reference>
   </jaxmtst:signature>
  </jaxmtst:security>
  <jaxmtst:TimeStamp xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
   <jaxmtst:creation>2016-01-12T00:56:12.706Z</jaxmtst:creation>
   <jaxmtst:expire>2016-01-12T01:01:12.706Z</jaxmtst:expire>
  </jaxmtst:TimeStamp>
  <jaxmtst:MessageID xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
   <jaxmtst:ID>IDvalue</jaxmtst:ID>
  </jaxmtst:MessageID>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
  <m:GetLastTradePrice xmlns:m="http://wombat.ztrade.com">
   <symbol>SUNW</symbol>
  </m:GetLastTradePrice>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 6.2.

المنارة للاستشارات
www.manaraa.com

```
Output - trythefilter (run)
run:
Thm Message is less than the maximum Size
SOAP message version 1.1
the input type is:  class java.lang.String
the input is from the same type
Expire time is less than Current Time
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 6.2 Normal Case Result

6.1.2. Checking the approach against XML Rewriting attack

In this test we aim to modify the SOAP message structure then test the SOAP message using the ontology checker. The ontology checker checks the SOAP message structure. This checking process is presented in Section 5.4.

First the ontology checker tests the existence of mandatory elements, then it checks their positions. **Figure 6.3** illustrates a modified SOAP message where the yellow area presents the modification in the SOAP message, namely, the element of Time-stamp (Creation, Expire) in the Security section.

Then the ontology checker returns the SOAP message to its original SOAP structure based on the attached SOAP message in the log file. **Figure 6.4** shows the result of catching the modification and the positions of the modifications. **Figure 6.1** illustrates the SOAP message after being returned to its original structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <jaxmtst:security xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
      <jaxmtst:creation>2015-10-23T15:18:23.421Z</jaxmtst:creation>
      <jaxmtst:expire>2015-10-23T15:23:23.421Z</jaxmtst:expire>
    </jaxmtst:security>
    <jaxmtst:TimeStamp xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
    </jaxmtst:TimeStamp>
```

Figure 6.3 A modified SOAP message

```
the element Expire is missing
the element Creation is missing
TimeStamp mandatory elements are exist

BUILD SUCCESSFUL (total time: 4 seconds)
```

Figure 6.4 Recognizing Time-Stamp modification

المنارة للاستشارات
www.manaraa.com

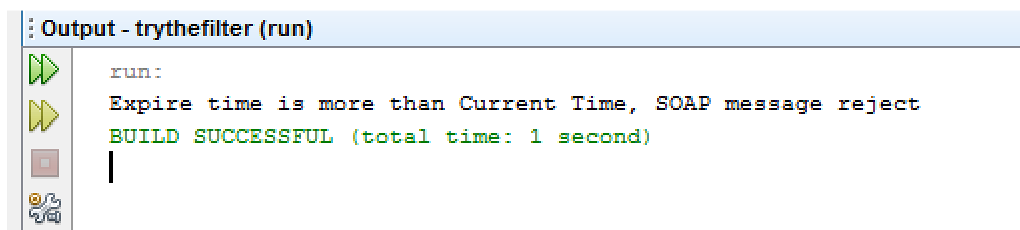
6.1.3. Checking the Approach against Replay Attack

Figure 6.5 presents a SOAP message with wrong time (an old time), according to the replay filter this message must be rejected. We take this old SOAP message and check it using Replay filter. **Figure 6.6** show the result of the testing process. Testing the implementation in Section 5.5.2.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Header>
  <jaxmtst:security xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
   <jaxmtst:signature>Signed Value<jaxmtst:reference>Refvalue</jaxmtst:reference>
   </jaxmtst:signature>
  </jaxmtst:security>
  <jaxmtst:TimeStamp xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
   <jaxmtst:creation>2015-10-23T15:18:23.421Z</jaxmtst:creation>
      <jaxmtst:expire>2015-10-23T15:23:23.421Z</jaxmtst:expire>
  </jaxmtst:TimeStamp>
```

Figure 6.6 Replay attack

Figure 6.5 An Old SOAP message



6.1.4. Checking the approach against Oversized attack

In this test we put the maximum size of the SOAP message equal 300 byte. This is very small value but this value depends on the average of the expected SOAP message. We make the SOAP message exceeding the maximum allowed size, so the Filter rejects the message. The result of this checking process is demonstrated in **Figure 6.7**. implementing the checking process for Section 5.5.1.

المنارة للاستشارات
www.manaraa.com

```
19            File file =new File("replay.xml");
20           over = 300;
21           double size = file.length();
```

trythefilter.Size 》 main 》 try 》

Output - trythefilter (run) ×

```
run:
Reject the SOAP Message (OverSize Attack)
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 6.7 Oversized Attack

6.1.5. Checking the approach against Parameter Tampering attack

In this test, we test the input of the SOAP message according to the parameter type. The input type must be the same as the Web Service is expected to receive, so we test the input of the Body elements and compare the type of the inputs with the expected one. If there is any change in the input type the SOAP message is rejected. This checking test the implementation in Section 5.5.3.

Figure 6.8 shows an example testing if the type String is expected as input in the SOAP message. If any other type is included instead of the String type, the SOAP message is rejected.

Source History

```
34          XPathExpression expr = xpath.compile("/Envelope/Body/GetLastTradePrice/symbol/text()");
35          Object result = expr.evaluate(doc, XPathConstants.NODESET);
36          NodeList nl = (NodeList) result;
37             for (int i = 0; i < nl.getLength(); i++) {
38               input = nl.item(i).getNodeValue();  }
39              if (input == null){
40                  String rslt = "The input value is NULL. Reject SOAP Message ";
41                  JOptionPane.showMessageDialog(null, rslt);
42                  exit(); }else{System.out.println("the input type is:  "+input.getClass()); }
43                    type = "SUNW";
44          if(type.getClass()== input.getClass()){
45          System.out.println("the input is from the same type");
46           }else { System.out.println("the type of the input is not equal "); } }
            catch(Exception e) {
```

Output ×

Retriever Output × trythefilter (run) ×

```
run:
the input type is:  class java.lang.String
the input is from the same type
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 6.8 Parameter Type Test

6.1.6. Checking the approach against Coercive-Parsing attack

In this checking process, we check the SOAP message namespace to identify which SOAP version this SOAP message applies. SOAP has two versions and each version has its specific namespaces. Therefore we aim to test SOAP message to determine which version it follows.

المنارة للاستشارات
www.manaraa.com

Figure 6.9 illustrates a modified namespace. So the Coercive parsing filter should detect this as malicious modification. The result of the detection process is shown in **Figure 6.10**. This checking process testing the implementation in Section 5.5.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://google.com/modified/soap/envelope/">
  <SOAP-ENV:Header>
    <jaxmtst:security xmlns:jaxmtst="http://www.jcommerce.net/soap/jaxm/TestJaxm">
      <jaxmtst:signature>Signed Value<jaxmtst:reference>Refvalue</jaxmtst:reference></jaxmtst:signature>
    </jaxmtst:security>
```

Figure 6.9 Coercive Parsing Attack

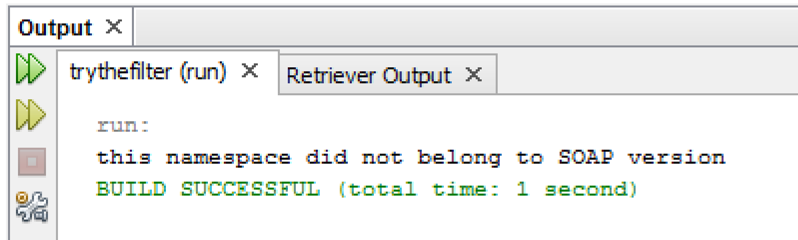


Figure 6.10 Coercive Parsing checking result

6.2. Evaluation

In this section we provide an evaluation for the proposed approach with comparison with other approaches. The evaluation process depends on three measures specified in research objectives to be preserved by the approach. These measures are: integrity, confidentiality, and time efficiency.

6.2.1. Integrity

We defined the integrity as protecting SOAP message from being modified by unauthorized parties during the transmission process. Every XML rewriting attack performs some sort of modifications on the SOAP message. By detecting these modifications and in some cases correcting them, we can preserve integrity. Our approach achieves integrity based on the ontology predefined SOAP message structure using the ontology checker. The receiver expects intact SOAP elements same as the originally sent one. If there is any missing elements, this means there is unauthorized modification. So as we mentioned, the approach needs to check against the hidden and encrypted SOAP message in the attached log file. If the message does not comply with the copy in the log file, the approach considers the message as a malicious one. Therefore this check preserves the integrity of the SOAP message.

More specifically, we define the mandatory elements in the SOAP message such as: Envelope, Header, Body, Security, Timestamps, MessageID, Creation, and Expire. Also we define the relation between these mandatory elements, and we classify the XML rewriting attacks using the

المنارة للاستشارات
www.manaraa.com

ontology checker, so any modification is classified as malicious and is defined as an attack. So our approach ensures the following constraints:

- The attacker cannot add a new element in a SOAP message
- The attacker cannot delete an existing element from a SOAP message
- The attacker cannot change the order of the signed elements of a SOAP message

Other approaches tried to preserve the integrity of the SOAP message depending on various ways such as copying the SOAP message in an element in the header block of the SOAP message itself [6]. But [23] shows that the main limitation of such an approach is that there is no unique mechanism to specify the parent in the message while it is an element in the header. Therefore the attacker can modify the sub elements with parent without violating the validation process. In our approach, we avoid such a limitation be encrypting the SOAP message in a log file that is attached to the SOAP message. This is more efficient and better in terms of integrity.

6.2.2. Confidentiality

In any networked system, the communicating parties need to exchange data while guaranteeing that only the expected receiver can read this data. This means that the exchanged data must be protected against eavesdroppers. The term confidentiality in the field of information security refers to the requirement for exchanged data between two communicating parties not to be available to a third party that may try to pry into the communication (O'Neill et al., 2003). In order to achieve confidentiality, one approach is to use a private connection between the communicating parties, such as a dedicated line or a virtual private network (VPN). However, the critical information of a Web service, such as WSDL and SOAP messages, are usually exchanged through untrusted networks, the Internet most likely, where the private connection is not achievable and another approach is used to meet the requirement of confidentiality, which is encryption.

In SOAP Account and RewritingHealer approaches, the confidentiality depending on signing and encrypted the elements, to ensure the confidentiality. Also they signing their elements (SOAP Account, and RewritingHealer) each time and added it to SOAP message.

In our approach, we achieve confidentiality using the encryption. We encrypt the log file, also in case of sensitive information in the Body. We make Security element as a mandatory element to allow the signature ability of the body or other elements.

المنارة للاستشارات
www.manaraa.com

6.2.3. Time efficiency

One main advantage of our approach is time efficiency because the ontology checker checks the message structure just once. In irregular cases the need for decrypting the captured copy is needed. Also filters are executed in a concurrent manner minimizing time of the filtering process.

To show time efficiency for the ontology checkering, assume that the number of nodes is n, and the checking time for each node for the element position in the ontology checker is t.

So the time taken by the ontology checker in the regular case is t*n

If we consider the decryption time (i.e., in the irregular case) to be c.

So the ontology checker in such cases is t*n*c and this is the worst case.

We should mention that the time consuming for node position checking is a fixed time. Also the time for encryption and decryption is fixed. Therefore, the complexity of the checking process is linear, i.e., O(n) depending on the number of nodes.

This complexity is less than that of the RewritingHealer approach [6], where the header block has to be checked using a search in a hash table.

It is worth to mention that our proposed approach discard the necessity of using WS Security [26] for the detection of XML rewriting attack. Where in WS Security each signing and encryption process take period of time, in our approach there is not any need for signing and un-signing procedure. Also WS Security Policy [26], each assertion needs to have a module to verify whether a message satisfies the assertion or not. Therefore if there are assertions in a WS Security Policy file associated with the different patterns of XML rewriting attack, each of these assertions will have a modules and each of these modules will have process the message. It is certainly a time consuming procedure.

The concurrent execution of filters are obviously preserving time. We show this through two cases: The first case using the multithreaded concurrent procedure for checking SOAP message and the second case without any concurrency. **Figure 6.11** shows the time taken by the concurrent procedure and **Figure 6.12** shows the time taken by the sequential procedure. The two figures show that the sequential time is 184071123 Nano second while the concurrent time is 5437086 Nano second. Let us calculate the percentage and the speedup equation.

First the time decrease in percentage between the sequential and the concurrent =

(184071123-5437086/184071123)*100= 97% .

Now we need to calculate the speedup equation. Which is (1/1-p) where p is the percentage between the sequential and concurrent procedure. So

Speedup = (1/1-.97)= 33.33 time.

المنارة للاستشارات
www.manaraa.com

The result shows that the difference is in nanosecond. But in case we want to handle thousands of SOAP messages, the difference becomes more significant.

```
Output - trythefilter (run)
run:
Thm Message is less than the maximum Size
5437086
SOAP message version 1.1
the input type is:  class java.lang.String
the input is from the same type
Expire time is less than Current Time, Message Reject
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 6.11 Time consumed by the concurrent procedure for filters checking process

```
Output - trythefilter (run)
run:
the input type is:  class java.lang.String
the input is from the same type
Thm Message is less than the maximum Size
Expire time is more than Current Time, Message Reject
SOAP message version 1.1
184071123
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 6.12 Non-concurrent procedure for filters checking process

Another advantage of our approach is handling multiple kinds of Web Services SOAP message attacks. Other approaches are specialized in one or some of these attacks and ignore other kinds.

6.3. Summary

In this chapter, we presented a number of test cases to prove that our approach achieves the detection and efficient handling of SOAP messages. The cases are divided into six cases; the first case illustrated a SOAP message without malicious attack. Then we modified the message and proved that our approach detects this malicious modification. Also we demonstrate four other checking cases for Web Services SOAP message attacks with one checking case for each filter. In evaluation section we introduced a brief evaluation process targeting integrity, confidentiality, and time efficiency. Also we presented a brief comparison with some related approaches.

المنارة للاستشارات
www.manaraa.com

Chapter 7 Conclusion and Recommendations

7.1. Conclusion

The security issue in Web Services depending in the first step on the SOAP message. Therefore, securing SOAP messages means securing the entire Web Services.

We have built an ontology-based detection approach to detect SOAP message attacks. The approach consisted of two main parts: the predefined ontology SOAP message part designed to preserve the SOAP message structure by capturing the message elements and defining the relationship between these elements. Therefore it was able to detect XML rewriting attacks. The second part consisted of the policy filters, these filters check if the SOAP Message is vulnerable to Denial of Services attack.

The ontology is designed to reflect the SOAP message structure based on the original SOAP elements (mandatory elements) and is used be the ontology checker component to check the existences and real position of these elements. The ontology checker is designed to check the structure of the SOAP message and compares it with the predefined ontology based structure to recognize any missing or modifications happened to the SOAP message during the transmission process.

The filters started their procedure after the received SOAP message pass ontology checker procedure. They are responsible for handling four kinds of SOAP message attacks: Oversized attack, Replay attack, Parameter Tampering attack, and the Coercive parsing attack. The message is rejected if it do not achieve the restrictions in any one of these filters.

Our approach handles the attacks before reaching the Web Service because it exists in the middle between the sender and the receiver. It has added new advantages to the SOAP message attack handling approach. It handle more than one attack and works concurrently to preserve time.

The approach is evaluated for preserving integrity and confidentiality and for ensuring time efficiency through the concurrent execution of the policy filters. The results show a 33.3 times improvement of the concurrent execution over the sequential one. Also the approach have proven to achieve these measure even better than similar approaches.

7.2. Recommendation

As a recommendation, we aim to add new SOAP message Web Services attacks such as cross-site scripting, also we aim to make our approach adaptable. The adaptability we aim to achieve as a future work is the ability to cure the SOAP message against any missing or modification in its structure. The ontology can be further extends to be used in the filtering process.

The approach needs to be implemented as a complete sub-system in any SOAP engine for Web Services and this needs a complete implementation of the approach.

المنارة للاستشارات
www.manaraa.com

Bibliography

[1] L. Liberti, "CA Survey Finds Security Concerns Slow SOA/Web Service Implementation," CA Advisor Security Management Newsletter, 2009.

[2] A. Nadalin, C. Kaler, Hallam-Baker, PH. and R. Mozillo, "Web Services Security: SOAP Message Security 1.0," http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, , OASIS Standard 200401, March, 2004.

[3] M. McIntosh and P. Austel, "XML Signature Element Wrapping Attacks and Countermeasures," in *Proceedings of the International Workshop on Secure Web Services (SWS);*, Fairfax, VA, USA, Nov, 2005.

[4] W. W. W. consortiim, "Web Service Architecture Requirments," http://www.w3c.org/TR/wsa-reqs/, 2008.

[5] N. Aziz, B. Jeong-Yong and P. Young-Ho, "a study on Detection Techniques of XML Rewriting Attacks in Web Services," *International Journal of Control and Automation,* vol. 7, 2014.

[6] F. A. Kadir, "RewritingHealer: An Approach for securing Web Services Communication," *Master Thesis in KTH Royal Institute of Technology,* vol. Sweden, 2008.

[7] K. Bhargavan, C. Fournet, A. D. Gordon and G. O'Shea, "An Advisor for Web Services Security Policies, SWS`05,," *Fairfax, Virginia, USA,* 2005.

[8] U. Elangovan and K. Arputharaj, "Improving cross site scripting filters for input validation against attacks in Web Services," *Kuwait J. Sci. 41(2),* pp. 175-203, 2014.

[9] C. Dan, v. H. Frank, H. Ian, L. M. Deborah, F. ,.-S. Peter and A. S. Lynn, DAML + OIL (March 2001) Reference Description, http://www.w3.org/TR/daml+oil-reference, 2001.

[10] D. L. McGuiness, "Conceptual Modeling for Distributed Ontology Environments," in . *In the Proceedings of The Eighth International Conference on Conceptual Structures in Logical, Linguistic, and CompuatationalIssues (ICCS 2000)*, Darmstadt, Germany, 2000.

[11] J. Hendler and E. Miller, Web Ontology Language, http://www.w3.org/2004/OWL/, 2004.

[12] Beckett and David, RDF/XML Syntax Specification (Revised)., http://www.w3.org/TR/rdf-syntax-grammar/, Feb 2004.

[13] M. K. Smith, C. Welty and D. L. McGuinness, OWL Web Ontology Language Guide, http://www.w3.org/TR/owl-guide/, 2004.

[14] X. Wang, Q. ,. Da, G. Tao and H. Pung, Ontology Based Context Modeling and Reasoning

المنارة للاستشارات

www.manaraa.com

using OWL., 2004.

[15] M. A., M. B. and S. L., "Managing Multiple Distributed Ontologies in the Semantic Web," *VLDB Journal,* pp. 286-302, 2003.

[16] P. Boyce, "Developing Domain Ontologies For Course Content," *Educational Technology & Society,* pp. 275-288, 10 (3) 2007.

[17] Rebekah Gilmour, "An ontology for Hazard Identification in Risk Management," *Department of Chemical Engineering,* 2004.

[18] U. Mike and G. Micheal, "Ontologies: Principle, Methods and applications," *Knowledge Engineering Review,* February 1996.

[19] B. McBride, "Jena: A Semantic Web Toolkit," *IEEE Computer Sociaty,* pp. 55-59, November 2002.

[20] E. Moradian and A. Hakansson, "Possible Attacks on XML Web Services," vol. Vol. 6, pp. 154-170, 2006.

[21] M. A. Rahman, A. Schaad and M. Rits, "Towards Secure SOAP message Exchange in a SOA," *Proceedings of the secure Web Services Workshop,* no. Fairfax, USA, pp. 77-84, 2006.

[22] M. A. Rahaman and A. Schaad, "SOAP-Based Secure Conversation and Collaboration," *International Conference on Web Services (ICWS),* no. Los Angeles, CA, USA, 2009.

[23] S. Gajek, L. Liao and J. Schwenk, "Breaking and Fixing the Inline Approach," *Proceeding of the ACM Workshop on Secure Web Services (SWS),* no. Los Angeles, CA, USA, 2008.

[24] T. S. Barhoom and R. S. K. Rasheed, "Position of Signed Element for SOAP Message Integrity," *International Journal of Computer Information Systems,* vol. 2, no. 4, 2011.

[25] S. Anderson and a. e. al, "web services trust language, (WS-Trust),," *WS-trust,* 2005.

[26] S. Anderson and a. e. al, "web services secure conversation language," *http://schemas.xmlsoap.org/ws/2005/02/sc,* 2005.

[27] M. Rajni and D. Deepak, "A Proposed SOAP Model Against Wrapping and Issue Conversation," *International Journal of Computer Science Issues,* vol. 10, no. 2, p. 3, March 2013.

[28] S. Gajek, M. Jensen, L. Liao and J. , "analysis the Signature Wrapping attacks and Countermeasure," *in the Proc of IEEE International Conference on Web Services, ICWS, Los Anglese, CA,* pp. 575-582, 2009.

المنارة للاستشارات

www.manaraa.com

[29] S. K. Sinh and A. Benameur, "A Formal Solution to Rewriting Attack on SOAP Message," *in Proc of ACM workshop on Secure Web Services SWS`08 ACM New Yourk, NY, USA,* pp. 53-60, 2008.

[30] V. Patel, R. Mohandas and A. R. Pais, "Attacks on Web Services and Mitigation Schemas," *International Journal of Computer Engineering,* vol. 3, no. 5, 2010.

[31] A. Nasridinov and J. Y. Byun, "A Self-Adaptive Approach to Ensure Integrity of SOAP Message," 2012.

[32] N. Aziz, P. H. Pham, Q. Lin and Y. B. Jeong, "XAT-SOAP: XML-based attack-tolerent SOAP Message," 2012.

[33] A. Nasridinov, J. Y. Byun and Y.-H. Park, "UNRWAP: an approach on wrapping-attack tolerant SOAP Message," *Second International Conference on Cloud and Green Computing,* 2012.

[34] G. ,. Chan, H. S. Wong and G. Rao, "an Adaptive Intrusion Detection and Prevention (ID/IP) Framwork for Web Service," *Proc. Int`1 Conf. Convergence Information Technology, IEEE,* pp. 528-534, 2007.

[35] D. Shipra, B. Suman and T. Deepika, "Security Issues in cloud Computing and Countermeasures," *International Journal of Engineering Science and Technology (IJEST),* 2011.

[36] Thilagam, A. N. Gupta and D. P. Santhi, "Attacks on Web Services Need to Secure XML on Web," *Computer Science & Engineering: an International Journal (CSEIJ),* vol. 3, p. 5, October 2013.

[37] I. B. Mohamed and S. ,. R. Dr. Mohamed, "Server SOA Security Threats on SOAP Web Services-A Critical Analysis," *ISOR Juornal of Computer Engineering (ISOR-JCE),* vol. 16, no. 2, Mar-Apr 2014.

[38] A. V. a. J. Han, "Security Attack ontology for web service," in *proceeding of the second International Conference on Semantics, Knowledge, and Grid*, IEEE, 2006.

[39] M. Jensen, N. Gruschka and R. Herkenhoner, "A survey of attack on Web Services," *Computer Science- Research and Developmemt,* pp. 185-197, November 2009.

[40] R. K. Swati and D. Aarti, "A Survey On XML-Injection Attacks Detection Systems," *International Journal of Science and Research (IJSR),* vol. 3, no. 5, May 2014.

[41] A. N. Gupta and S. Thilagam, "Attacks ON Web Services Need to Secure XML On Web," *Computer Science & Engineering: An International Journal (CSEIJ),* vol. Vol. 3, no. No. 5, October 2013.

المنارة للاستشارات

www.manaraa.com

[42] W. Negm, "Anatomy of Web Services Attack: A guide to threat and preventive Countermeasures," *Forum Systems Inc,* 2004.

[43] J. Meiko, G. Nils and H. Ralph, "A survey of attack on Web Services," *Computer Science-Research and Developmemt Volume 24, Number 4,* pp. 185-197, 2009.

[44] M. Ficco and M. Rak, "Intrusion Tolerant Approach for Denial of Service Attacks to Web Services," *First International Conference on Data Compression, Communications and Processing,* pp. 285-292, 2011.

[45] S. w. A. A. f. Java, "https://docs.oracle.com/javaee/5/tutorial/doc/bnbhg.html," SUN Microsoft, July 2005. [Online].

[46] S. B. Palmer, The Semantic Web: An Introduction, http://infomesh.net/2001/swintro/, 2001.

[47] "The World Wide Web Consortium," [Online]. Available: http://www.w3.org/ .

[48] M. McIntosh and P. Austel, "XML Signature Element Wrapping attacks and Countermeasures," in *Proceedings of the International Workshop on Secure Web services (SWS)*, Fairfax, VA, USA, 2005.

[49] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, Extensible Markup Language (XML) 1.0 (Third Edition), http://www.w3.org/TR/2004/REC-xml-20040204/, 2004.

[50] G.-C. Javier, T. David and B. Claudio, "Description Logics for Matchmaking of Services.," *Hewlett-Packard Company Technical Report,* 2001.

[51] K. Holger, W. F. Ray, F. N. Natalya and A. M. Mark, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications," *ISWC,* 2004.

[52] C. Trim, "Jena: A Semantic Web Framework," 2013. [Online]. Available: http://trimc-nlp.blogspot.com/2013/06/introduction-to-jena.html.

المنارة للاستشارات

www.manaraa.com

Appendixes

Appendix 1: The SOAP Structure Ontology in OWL Format:


```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
   <!ENTITY SOAP "http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#" >
]>
<rdf:RDF xmlns="http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#"
    xml:base="http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:SOAP="http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#">
   <owl:Ontology rdf:about="http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah"/>
   <!--
   ///////////////////////////////////////////////////////////////////////////////////////
   //
   // Object Properties
   //
   ///////////////////////////////////////////////////////////////////////////////////////
    -->

   <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#ChildOf -->
   <owl:ObjectProperty rdf:about="&SOAP;ChildOf">
      <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
      <owl:inverseOf rdf:resource="&SOAP;ParentOf"/>
   </owl:ObjectProperty>
   <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#ParentOf -->
   <owl:ObjectProperty rdf:about="&SOAP;ParentOf">
      <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
   </owl:ObjectProperty>
   <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Unique -->
   <owl:ObjectProperty rdf:about="&SOAP;Unique">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
   </owl:ObjectProperty>
   <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#hasNode -->
   <owl:ObjectProperty rdf:about="&SOAP;hasNode">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
   </owl:ObjectProperty>
   <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#isExist -->
   <owl:ObjectProperty rdf:about="&SOAP;isExist">
      <rdf:type rdf:resource="&owl;FunctionalProperty"/>
   </owl:ObjectProperty>
   <!--
   ///////////////////////////////////////////////////////////////////////////////////////
   //
   // Classes
   //
   ///////////////////////////////////////////////////////////////////////////////////////
    -->
   <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Body -->
```


المنارة للاستشارات
www.manaraa.com

```
<owl:Class rdf:about="&SOAP;Body">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Creation -->
<owl:Class rdf:about="&SOAP;Creation">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Envelope -->
<owl:Class rdf:about="&SOAP;Envelope">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Expire -->
<owl:Class rdf:about="&SOAP;Expire">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Header -->
<owl:Class rdf:about="&SOAP;Header">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#MessageID -->
<owl:Class rdf:about="&SOAP;MessageID">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Case -->
<owl:Class rdf:about="&SOAP;SOAP_Case"/>

<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Message -->

<owl:Class rdf:about="&SOAP;SOAP_Message"/>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Security -->

<owl:Class rdf:about="&SOAP;Security">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#TimeStamp -->
<owl:Class rdf:about="&SOAP;TimeStamp">
   <rdfs:subClassOf rdf:resource="&SOAP;SOAP_Message"/>
</owl:Class>
<!--
///////////////////////////////////////////////////////////////////////////////////////
//
// Individuals
//
///////////////////////////////////////////////////////////////////////////////////////
 -->
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Body -->
<owl:NamedIndividual rdf:about="&SOAP;Body">
   <rdf:type rdf:resource="&SOAP;Body"/>
   <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
   <Unique rdf:resource="&SOAP;Body"/>
   <ChildOf rdf:resource="&SOAP;Envelope"/>
</owl:NamedIndividual>

<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Creation -->
<owl:NamedIndividual rdf:about="&SOAP;Creation">
   <rdf:type rdf:resource="&SOAP;Creation"/>
   <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
   <isExist rdf:resource="&SOAP;Creation"/>
```


المنارة للاستشارات
www.manaraa.com

```
    <ChildOf rdf:resource="&SOAP;TimeStamp"/>
</owl:NamedIndividual>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Envelope -->
<owl:NamedIndividual rdf:about="&SOAP;Envelope">
    <rdf:type rdf:resource="&SOAP;Envelope"/>
    <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
    <ParentOf rdf:resource="&SOAP;Body"/>
    <ParentOf rdf:resource="&SOAP;Header"/>
</owl:NamedIndividual>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Expire -->
<owl:NamedIndividual rdf:about="&SOAP;Expire">
    <rdf:type rdf:resource="&SOAP;Expire"/>
    <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
    <ChildOf rdf:resource="&SOAP;TimeStamp"/>
</owl:NamedIndividual>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Header -->
<owl:NamedIndividual rdf:about="&SOAP;Header">
    <rdf:type rdf:resource="&SOAP;Header"/>
    <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
    <ChildOf rdf:resource="&SOAP;Envelope"/>
    <ParentOf rdf:resource="&SOAP;MessageID"/>
    <ParentOf rdf:resource="&SOAP;Security"/>
    <ParentOf rdf:resource="&SOAP;TimeStamp"/>
</owl:NamedIndividual>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#MessageID -->
<owl:NamedIndividual rdf:about="&SOAP;MessageID">
    <rdf:type rdf:resource="&SOAP;MessageID"/>
    <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
    <ChildOf rdf:resource="&SOAP;Header"/>
</owl:NamedIndividual>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Attack1 -->
<owl:NamedIndividual rdf:about="&SOAP;SOAP_Attack1">
    <rdf:type rdf:resource="&SOAP;SOAP_Case"/>
    <hasNode rdf:resource="&SOAP;Body"/>
    <hasNode rdf:resource="&SOAP;Envelope"/>
    <hasNode rdf:resource="&SOAP;Header"/>
    <hasNode rdf:resource="&SOAP;MessageID"/>
    <hasNode rdf:resource="&SOAP;Security"/>
    <hasNode rdf:resource="&SOAP;TimeStamp"/>
</owl:NamedIndividual>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Attack2 -->

<owl:NamedIndividual rdf:about="&SOAP;SOAP_Attack2">
    <rdf:type rdf:resource="&SOAP;SOAP_Case"/>
    <hasNode rdf:resource="&SOAP;Body"/>
    <hasNode rdf:resource="&SOAP;Creation"/>
    <hasNode rdf:resource="&SOAP;Envelope"/>
    <hasNode rdf:resource="&SOAP;Expire"/>
    <hasNode rdf:resource="&SOAP;Header"/>
    <hasNode rdf:resource="&SOAP;Security"/>
    <hasNode rdf:resource="&SOAP;TimeStamp"/>
</owl:NamedIndividual>
<!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Attack3 -->
<owl:NamedIndividual rdf:about="&SOAP;SOAP_Attack3">
    <rdf:type rdf:resource="&SOAP;SOAP_Case"/>
    <hasNode rdf:resource="&SOAP;Body"/>
    <hasNode rdf:resource="&SOAP;Envelope"/>
</owl:NamedIndividual>
```


المنارة للاستشارات
www.manaraa.com

```
    <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Attack4 -->
    <owl:NamedIndividual rdf:about="&SOAP;SOAP_Attack4">
        <rdf:type rdf:resource="&SOAP;SOAP_Case"/>
        <hasNode rdf:resource="&SOAP;Creation"/>
        <hasNode rdf:resource="&SOAP;Envelope"/>
        <hasNode rdf:resource="&SOAP;Expire"/>
        <hasNode rdf:resource="&SOAP;Header"/>
        <hasNode rdf:resource="&SOAP;MessageID"/>
        <hasNode rdf:resource="&SOAP;Security"/>
        <hasNode rdf:resource="&SOAP;TimeStamp"/>
    </owl:NamedIndividual>
    <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Attack5 -->
    <owl:NamedIndividual rdf:about="&SOAP;SOAP_Attack5">
        <rdf:type rdf:resource="&SOAP;SOAP_Case"/>
        <hasNode rdf:resource="&SOAP;Body"/>
        <hasNode rdf:resource="&SOAP;Creation"/>
        <hasNode rdf:resource="&SOAP;Expire"/>
        <hasNode rdf:resource="&SOAP;Header"/>
        <hasNode rdf:resource="&SOAP;MessageID"/>
        <hasNode rdf:resource="&SOAP;Security"/>
        <hasNode rdf:resource="&SOAP;TimeStamp"/>
    </owl:NamedIndividual>
    <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#SOAP_Normal -->
    <owl:NamedIndividual rdf:about="&SOAP;SOAP_Normal">
        <rdf:type rdf:resource="&SOAP;SOAP_Case"/>
    </owl:NamedIndividual>
    <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#Security -->
    <owl:NamedIndividual rdf:about="&SOAP;Security">
        <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
        <rdf:type rdf:resource="&SOAP;Security"/>
        <ChildOf rdf:resource="&SOAP;Header"/>
    </owl:NamedIndividual>
    <!-- http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#TimeStamp -->
    <owl:NamedIndividual rdf:about="&SOAP;TimeStamp">
        <rdf:type rdf:resource="&SOAP;SOAP_Message"/>
        <rdf:type rdf:resource="&SOAP;TimeStamp"/>
        <ParentOf rdf:resource="&SOAP;Creation"/>
        <ParentOf rdf:resource="&SOAP;Expire"/>
        <ChildOf rdf:resource="&SOAP;Header"/>
    </owl:NamedIndividual>
</rdf:RDF>
<!-- Generated by the OWL API (version 3.5.0) http://owlapi.sourceforge.net -->
```


www.manaraa.com

Appendix 2: SOAP Creation Code

```
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.ConsoleAppender;
import org.apache.log4j.PatternLayout;
//import junit.framework.TestCase;
import java.io.BufferedWriter;
import javax.xml.soap.*;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import java.text.DateFormat;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URL;
import java.util.Iterator;
import java.util.logging.FileHandler;
import java.util.logging.Handler;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import java.text.SimpleDateFormat;
import java.util.TimeZone;
/**
 *
 * @author Mahmoud Hamouda
 */
public class Test {
    private final static    Logger logger = Logger.getLogger(Test.class);
    public   static String MIME_MULTIPART_RELATED = "multipart/related";
    public  static String MIME_APPLICATION_DIME = "application/dime";
    public static  String NS_PREFIX = "jaxmtst";
    public  static String NS_URI = "http://www.jcommerce.net/soap/jaxm/TestJaxm";
    public static String getSOAPMessageAsString(SOAPMessage soapMessage) {
      try {
        TransformerFactory tff = TransformerFactory.newInstance();
        Transformer tf = tff.newTransformer();
        // Set formatting
        tf.setOutputProperty(OutputKeys.INDENT, "yes");
        tf.setOutputProperty("{http://xml.apache.org/xslt}indent-amount","2");
        Source sc = soapMessage.getSOAPPart().getContent();
        ByteArrayOutputStream streamOut = new ByteArrayOutputStream();
        StreamResult result = new StreamResult(streamOut);
        tf.transform(sc, result);
        String strMessage = streamOut.toString();
        return strMessage;
      } catch (Exception e) {
        System.out.println("Exception in getSOAPMessageAsString "
            + e.getMessage());
        return null;}   }
```


المنارة للاستشارات
www.manaraa.com

```
public static void main(String[] args) throws SOAPException {
   try{
   //This is used to get time in SOAP request in yyyy-MM-dd'T'HH:mm:ss.SSS'Z' format
  SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:dd.SSS'Z'");
  formatter.setTimeZone(TimeZone.getTimeZone("GMT"));

  //This is for TimeStamp element value
   java.util.Date create = new java.util.Date();
   java.util.Date expires = new java.util.Date(create.getTime() + (5l * 60l * 1000l));
   PropertyConfigurator.configure("loger.properties");
      // BasicConfigurator.configure();
       logger.debug("message creation ");
       //Create a SOAPConnection
      SOAPConnectionFactory factory =
              SOAPConnectionFactory.newInstance();
      SOAPConnection connection =
              factory.createConnection();
       //create the message and the connection
   MessageFactory mf = MessageFactory.newInstance();
   SOAPMessage msg = mf.createMessage();
   //msg.setProperty(SOAPMessage.WRITE_XML_DECLARATION ,Boolean.TRUE);
   logger.info("SOAPPart creation");
   SOAPPart soapPart = msg.getSOAPPart();
   logger.info("Envelope creation");
   SOAPEnvelope envelope = soapPart.getEnvelope();
   logger.info("header creation");
   SOAPHeader header = envelope.getHeader();
   logger.info("body creation");
   SOAPBody body = envelope.getBody();
   logger.info("security header creation");
   // header create
   SOAPHeaderElement security = header.addHeaderElement(envelope.createName("security", NS_PREFIX, NS_URI));
  // security.setActor();
   logger.info("signature creation");
       // signature
   SOAPElement signature = security.addChildElement("signature", NS_PREFIX);
   signature.addTextNode("Signed Value");
   logger.info("reference creation");
  //reference
    SOAPElement reference = signature.addChildElement("reference", NS_PREFIX);
   reference.addTextNode("Refvalue");
   logger.info("time creation");
   //time stamp
   //Adding Timestamp
   SOAPHeaderElement Time = header.addHeaderElement(envelope.createName("TimeStamp",NS_PREFIX ,NS_URI));
   SOAPElement creation = Time.addChildElement("creation",NS_PREFIX);
   creation.addTextNode(formatter.format(create));
   SOAPElement expire = Time.addChildElement("expire",NS_PREFIX);
   expire.addTextNode(formatter.format(expires));
   logger.info("messageID element creation");
   //message id
    SOAPHeaderElement MsgID = header.addHeaderElement(envelope.createName("MessageID",NS_PREFIX ,NS_URI));
   SOAPElement ID = MsgID.addChildElement("ID",NS_PREFIX);
   ID.addTextNode("IDvalue");
   logger.info("body element creation");
   //body element
   //Create a SOAPBodyElement
      Name bodyName = envelope.createName("GetLastTradePrice", "m", "http://wombat.ztrade.com");
```


المنارة للاستشارات
www.manaraa.com

```
          SOAPBodyElement bodyElement = body.addBodyElement(bodyName);
          //Insert Content
          Name name = envelope.createName("symbol");
          SOAPElement symbol = bodyElement.addChildElement(name);
          symbol.addTextNode("SUNW");
//  add attachment
        logger.info("attachment creation ---------------------------------------------");
        File myfile = new File("admin.txt");
        FileDataSource fds = new FileDataSource(myfile);
        DataHandler dh = new DataHandler(fds);
        AttachmentPart ap2 = msg.createAttachmentPart(dh);
        ap2.setContentId("attachment");
        msg.addAttachmentPart(ap2);
        MimeHeaders headers = msg.getMimeHeaders();
        //assertTrue(headers != null);
        String [] contentType = headers.getHeader("Content-Type");
        //assertTrue(contentType != null);
        // create the file
        String st = getSOAPMessageAsString(msg) ;
            System.out.print(st);
             File file = new File("soap.xml");
        FileOutputStream out = new FileOutputStream(file);
        msg.writeTo(out);
      File newFile  = new File("newone.xml");
      FileOutputStream str = new FileOutputStream(newFile);
      if (!newFile.exists()) {
                                        newFile.createNewFile();
                               }
         byte[] contentInBytes = st.getBytes();
                               str.write(contentInBytes);
                               str.flush();
                               str.close();
        }catch (Exception e) {
          System.out.println(e.getMessage());
        }
    }
  }
```


المنارة للاستشارات
www.manaraa.com

Appendix 3: Policy Filters Code

```
import java.io.File;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Locale;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class FiltersThread {
  public static void main (String [] args){
      Replay replay = new Replay();
      Thread rep = new Thread(replay);
      rep.start();

      Parameter parameter = new Parameter();
      Thread t1 = new Thread (parameter);
      t1.start();

      OverSized over = new OverSized();
      Thread t2 = new Thread(over);
      t2.start();

     Parsing parse = new Parsing();
     Thread t3 = new Thread(parse);
     t3.start();
    }
}
class Replay implements Runnable{
public static String msgDate;
    @Override
    public void run()  {
    try {
        replaymethod();
    } catch (ParserConfigurationException ex) {
        Logger.getLogger(Replay.class.getName()).log(Level.SEVERE, null, ex);
    } catch (SAXException ex) {
        Logger.getLogger(Replay.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Replay.class.getName()).log(Level.SEVERE, null, ex);
    } catch (XPathExpressionException ex) {
        Logger.getLogger(Replay.class.getName()).log(Level.SEVERE, null, ex);
    } catch (ParseException ex) {
        Logger.getLogger(Replay.class.getName()).log(Level.SEVERE, null, ex);
    }
    }
```


المنارة للاستشارات
www.manaraa.com

```
    public void replaymethod() throws ParserConfigurationException, SAXException, IOException, XPathExpressionException,
ParseException{
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse("xml.xml");
        XPathFactory xPathfactory = XPathFactory.newInstance();
        XPath xpath = xPathfactory.newXPath();
        XPathExpression expr = xpath.compile("/Envelope/Header/TimeStamp/expire/text()");
        Object result = expr.evaluate(doc, XPathConstants.NODESET);
        NodeList n1 = (NodeList) result;
                for (int i = 0; i < n1.getLength(); i++) {
                    msgDate = n1.item(i).getNodeValue();
                    if (msgDate ==null){
                        String empty = " The Expire Node Value is Missing. Reject the SOAP Message ";
                        System.out.println(empty);
                        System.exit(0);
                    }    }
          SimpleDateFormat ISO8601DATEFORMAT = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:dd.SSS'Z'",
Locale.US);
          java.util.Date date = new java.util.Date();
          String dateToday = ISO8601DATEFORMAT.format(date);
          java.util.Date expireTime = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:dd.SSS'Z'", Locale.US)
                  .parse(msgDate);
          java.util.Date currentTime = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:dd.SSS'Z'", Locale.US)
                  .parse(dateToday);
           if (expireTime.compareTo(currentTime) > 0) {
              String rsult = "Expire Time is more than Current Time, TimeStamp is right";
              System.out.println(rsult);
                     } else if (expireTime.compareTo(currentTime) < 0) {
              String rsult ="Expire time is less than Current Time, Message Reject";
              System.out.println(rsult);
               System.exit(0);
           } else if (expireTime.compareTo(currentTime) == 0) {
              String rsult ="Expire Time is Equal Current Time, Message Reject";
              System.out.println(rsult);
                     } else {
             String rsult ="Something weird happened...";
             System.out.println(rsult);
             System.exit(0);
           }
    }
    }
class Parameter implements Runnable{
    public static Object input ;
    public static Object type;
    @Override
    public void run() {
        try {
            parameterTampering();
        } catch (ParserConfigurationException ex) {
            Logger.getLogger(Parameter.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SAXException ex) {
            Logger.getLogger(Parameter.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(Parameter.class.getName()).log(Level.SEVERE, null, ex);
        } catch (XPathExpressionException ex) {
            Logger.getLogger(Parameter.class.getName()).log(Level.SEVERE, null, ex);
        }
```


المنارة للاستشارات
www.manaraa.com

```
    }
        public void parameterTampering() throws ParserConfigurationException, SAXException, IOException,
XPathExpressionException{
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse("xml.xml");
        XPathFactory xPathfactory = XPathFactory.newInstance();
        XPath xpath = xPathfactory.newXPath();
        XPathExpression expr = xpath.compile("/Envelope/Body/GetLastTradePrice/symbol/text()");
        Object result = expr.evaluate(doc, XPathConstants.NODESET);
        NodeList n1 = (NodeList) result;
          for (int i = 0; i < n1.getLength(); i++) {
           input = n1.item(i).getNodeValue();  }
          if (input == null){
              String rslt = "The input value is NULL. Reject SOAP Message ";
              System.out.println(rslt);
              System.exit(0);
          }else{System.out.println("the input type is:  "+input.getClass()); }
               type = "SUNW";
        if(type.getClass()== input.getClass()){
        System.out.println("the input is from the same type");
         }else { System.out.println("the type of the input is not equal "); }
    }
}
class OverSized implements Runnable{
    public static double over ;
    @Override
    public void run() {
        overSizedFilter();
    }
    public void overSizedFilter(){
        File file =new File("xml.xml");
    over = 1800;
    double size = file.length();
    if (size <= over ){
    String result = "Thm Message is less than the maximum Size";
    System.out.println(result);
    }else{
    String result = "Reject the SOAP Message (OverSize Attack)";
    System.out.println(result);
    System.exit(0);

    }
    }
}
class Parsing implements Runnable{
    @Override
    public void run() {
        try {
            coerciveParsing();
        } catch (ParserConfigurationException ex) {
            Logger.getLogger(Parsing.class.getName()).log(Level.SEVERE, null, ex);
        } catch (SAXException ex) {
            Logger.getLogger(Parsing.class.getName()).log(Level.SEVERE, null, ex);
        } catch (IOException ex) {
            Logger.getLogger(Parsing.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```


المنارة للاستشارات

www.manaraa.com

```
public void coerciveParsing()throws  ParserConfigurationException, SAXException, IOException{

  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setNamespaceAware(true);
   DocumentBuilder builder = factory.newDocumentBuilder();
   Document doc = builder.parse("xml.xml");
   String ns = doc.getDocumentElement().getNamespaceURI();
   //System.out.println(ns);
   String version = null;
   if (ns=="http://schemas.xmlsoap.org/soap/envelope/"){
    version = "SOAP message version 1.1";
    System.out.println(version);
   }
   else if (ns =="http://www.w3.org/2003/05/soap-envelope"){
   version = "SOAP message version 1.2 ";
   System.out.println(version);
   } else if (ns=="http://www.w3.org/2001/06/soap-envelope") {
      version = "SOAP message version 1.2 ";
      System.out.println(version);
      }else {System.exit(0);}
}}
```


المنارة للاستشارات

www.manaraa.com

Appendix 4: The Ontology Checker Code

```
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSetFactory;
import com.hp.hpl.jena.query.ResultSetRewindable;
import com.hp.hpl.jena.rdf.model.InfModel;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.reasoner.Reasoner;
import com.hp.hpl.jena.reasoner.rulesys.RDFSRuleReasonerFactory;
import com.hp.hpl.jena.util.FileManager;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;
import org.apache.jena.iri.impl.Main;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
public class FinalFilters {
  public static ArrayList TimeStamp = new ArrayList();
  public static ArrayList OntTime = new ArrayList();
  public static ArrayList header = new ArrayList();
  public static ArrayList OntHeader = new ArrayList();
  public static ArrayList envelop = new ArrayList();
  public static ArrayList Ontenvelop = new ArrayList();
  public static ArrayList array = new ArrayList();
  public static ArrayList OntologyArr = new ArrayList();
  public static String [] toString ;
  public static String mem;
  public static String s ;
  public static void getAll() throws ParserConfigurationException, SAXException, IOException{
     DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory
         .newInstance();
   DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
   Document document = docBuilder.parse(new File("xml.xml"));
   NodeList nodeList = document.getElementsByTagName("*");
   for (int i = 0; i < nodeList.getLength(); i++) {
      Node node = nodeList.item(i);
      if (node.getNodeType() == Node.ELEMENT_NODE) {
         String elements = node.getNodeName().substring(9);
         array.add(elements);
      }
    }
        FileManager.get().addLocatorClassLoader(Main.class.getClassLoader());
```


المنارة للاستشارات
www.manaraa.com

```
        Model model = FileManager.get().loadModel("E:\\xml research\\doctor\\ontology\\mah.owl");
        Reasoner reasoner = RDFSRuleReasonerFactory.theInstance().create(null);
        InfModel inf = ModelFactory.createInfModel(reasoner, model);
                String   mem1 = ("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
                    "PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
                    "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
                    "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
                    "PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>\n" +
                     "SELECT ?a ?t ?x\n" +
                  "       WHERE { ?a rdf:type SOAP:SOAP_Message.}");
  Query query = QueryFactory.create(mem1);
  QueryExecution agri = QueryExecutionFactory.create(query, inf);
  ResultSetRewindable results = ResultSetFactory.makeRewindable(agri.execSelect());
  while (results.hasNext()){
  QuerySolution qs = results.nextSolution();
  RDFNode a1 = qs.get("a");
  String[] b2 = a1.toString().split("\\^\\^http://www.w3.org/2001/XMLSchema#string");
  mem = b2[0].substring(65);
   OntologyArr.add(mem);
   }
   for (int i =0; i<=7; i++){
   String member = (String) OntologyArr.get(i);
   if (array.contains(member)){

   } else {System.out.println("the element "+member+" is missing");}
   }
  System.out.println("All the mandatory element are exist ");
  }
  public static void envelopChild() throws ParserConfigurationException, SAXException, IOException,
XPathExpressionException{
      DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory
          .newInstance();
   DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
   Document document = docBuilder.parse(new File("xml.xml"));
      XPathFactory xPathfactory = XPathFactory.newInstance();
       XPath xpath = xPathfactory.newXPath();
       XPathExpression expr = xpath.compile("/Envelope/*");
       Object result = expr.evaluate(document, XPathConstants.NODESET);
       NodeList n1 = (NodeList) result;
        for (int i = 0; i < n1.getLength(); i++) {
           String child =n1.item(i).getNodeName();
           String after = child.substring(9);
           envelop.add(after);}
   FileManager.get().addLocatorClassLoader(Main.class.getClassLoader());
        Model model = FileManager.get().loadModel("E:\\xml research\\doctor\\ontology\\mah.owl");
        Reasoner reasoner = RDFSRuleReasonerFactory.theInstance().create(null);
        InfModel inf = ModelFactory.createInfModel(reasoner, model);
                 String mem1 = ("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
"PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
"PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>\n" +
"SELECT ?a\n" +
"            WHERE { SOAP:Envelope SOAP:ParentOf ?a.}");
  Query query = QueryFactory.create(mem1);
  QueryExecution agri = QueryExecutionFactory.create(query, inf);
  ResultSetRewindable results = ResultSetFactory.makeRewindable(agri.execSelect());
  while (results.hasNext()){
```


المنارة للاستشارات
www.manaraa.com

```
QuerySolution qs = results.nextSolution();
RDFNode a1 = qs.get("a");
String[] b2 = a1.toString().split("\\^\\^http://www.w3.org/2001/XMLSchema#string");
s = b2[0].substring(65);
Ontenvelop.add(s);
}
    for (int i =0; i<=1; i++){
    String member = (String) Ontenvelop.get(i);
    if (envelop.contains(member)){

    } else {System.out.println("the element "+member+" is missing");}
    }
   System.out.println("Envelope mandatory elements are exist");
   }
   public static void headerChild() throws ParserConfigurationException, SAXException, IOException,
XPathExpressionException{
          DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory
            .newInstance();
    DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
    Document document = docBuilder.parse(new File("xml.xml"));
       XPathFactory xPathfactory = XPathFactory.newInstance();
        XPath xpath = xPathfactory.newXPath();
        XPathExpression expr = xpath.compile("/Envelope/Header/*");
        Object result = expr.evaluate(document, XPathConstants.NODESET);
        NodeList n1 = (NodeList) result;
         for (int i = 0; i < n1.getLength(); i++) {
             String child =n1.item(i).getNodeName();
             String after = child.substring(9);
             header.add(after); }
              FileManager.get().addLocatorClassLoader(Main.class.getClassLoader());
           Model model = FileManager.get().loadModel("E:\\xml research\\doctor\\ontology\\mah.owl");
           Reasoner reasoner = RDFSRuleReasonerFactory.theInstance().create(null);
           InfModel inf = ModelFactory.createInfModel(reasoner, model);
                  String mem1 = ("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
"PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
"PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>\n" +
"SELECT ?a\n" +
"              WHERE { SOAP:Header SOAP:ParentOf ?a.}");
   Query query = QueryFactory.create(mem1);
   QueryExecution agri = QueryExecutionFactory.create(query, inf);
   ResultSetRewindable results = ResultSetFactory.makeRewindable(agri.execSelect());

while (results.hasNext()){
QuerySolution qs = results.nextSolution();
RDFNode a1 = qs.get("a");
String[] b2 = a1.toString().split("\\^\\^http://www.w3.org/2001/XMLSchema#string");
s = b2[0].substring(65);
//System.out.println(b2[0].substring(65));
OntHeader.add(s);
   }
  for (int i =0; i<=2; i++){
    String member = (String) OntHeader.get(i);
    if (header.contains(member)){
        } else {System.out.println("the element "+member+" is missing");}      }
  System.out.println("All header mandatory element are exist");     }
```


المنارة للاستشارات
www.manaraa.com

```
  public static void TimeChild() throws ParserConfigurationException, SAXException, IOException,
XPathExpressionException{
      DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory
          .newInstance();
   DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
   Document document = docBuilder.parse(new File("xml.xml"));
     XPathFactory xPathfactory = XPathFactory.newInstance();
      XPath xpath = xPathfactory.newXPath();
      XPathExpression expr = xpath.compile("/Envelope/Header/TimeStamp/*");
      Object result = expr.evaluate(document, XPathConstants.NODESET);
      NodeList n1 = (NodeList) result;
       for (int i = 0; i < n1.getLength(); i++) {
                 String child =n1.item(i).getNodeName();
          String after = child.substring(9);
          TimeStamp.add(after); }
  FileManager.get().addLocatorClassLoader(Main.class.getClassLoader());
        Model model = FileManager.get().loadModel("E:\\xml research\\doctor\\ontology\\mah.owl");
        Reasoner reasoner = RDFSRuleReasonerFactory.theInstance().create(null);
        InfModel inf = ModelFactory.createInfModel(reasoner, model);
                 String mem1 = ("PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
"PREFIX owl: <http://www.w3.org/2002/07/owl#>\n" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n" +
"PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>\n" +
"PREFIX SOAP: <http://www.semanticweb.org/mahmoudhamouda/ontologies/2015/10/mah#>\n" +
"SELECT ?a\n" + " WHERE { SOAP:TimeStamp SOAP:ParentOf ?a.}");
  Query query = QueryFactory.create(mem1);
  QueryExecution agri = QueryExecutionFactory.create(query, inf);
  ResultSetRewindable results = ResultSetFactory.makeRewindable(agri.execSelect());
while (results.hasNext()){
QuerySolution qs = results.nextSolution();
RDFNode a1 = qs.get("a");
String[] b2 = a1.toString().split("\\^\\^http://www.w3.org/2001/XMLSchema#string");
s = b2[0].substring(65);
OntTime.add(s); }
   for (int i =0; i<=1; i++){
   String member = (String) OntTime.get(i);
   if (TimeStamp.contains(member)){
   } else {System.out.println("the element "+member+" is missing");}    }
     System.out.println("TimeStamp mandatory elements are exist");    }
  public static void main (String [] args) throws ParserConfigurationException, SAXException,
          IOException, XPathExpressionException{
     System.out.println();
     getAll();
     System.out.println();
     envelopChild();
     System.out.println();
     headerChild();
     System.out.println();
     TimeChild();}}
```


المنارة للاستشارات
www.manaraa.com