AN ONTOLOGY-BASED APPROACH FOR DETECTING SOAP MESSAGE ATTACKS

Mahmoud A. Hamouda[1] and Rebhi S. Baraka[2]

[1]Information Technology Graduate Studies, Islamic University of Gaza, Palestine

[2]Faculty of Information Technology, Islamic University of Gaza, Palestine

ABSTRACT

An ontology-based detection approach aiming to check SOAP messages for XML rewriting attacks is presented. The approach comprises a SOAP message ontology and a set of policy filters. The ontology is used in preserving the message structure including its constituent elements and their relationships. The policy filters check if the message complies with denial-of-service vulnerability restrictions. Message integrity is preserved using the ontology-based checker, which checks that the message has not been modified during the transmission process. Message confidentiality is preserved by encrypting a copy of the message in a log file combined with the message. Time efficiency is achieved by executing the policy filters in a concurrent manner.

KEYWORDS

SOAP Message, SOAP Ontology, XML Rewriting Attacks, Replay Attack, Coercive Attack, Oversized Attack, and Parameter Tampering Attack.

1. INTRODUCTION

The security issue in Web Services and SOAP messages remains a vital field of study and despite the growing use of Web Services by many organizations; security is still a major concern slowing their deployment [10]. Many security standards have been developed to secure SOAP messages; according to [2] and [12] the SOAP message is still vulnerable to many kinds of attacks.

Various approaches have been developed to handle multiple kinds of SOAP message attacks but mostly they 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 goal of this paper is to design an efficient approach that uses a domain ontology together with policy filters for the detection of SOAP message attacks, which preserves integrity and confidentiality.

The sender of the SOAP message attaches a copy of it to a log file, encrypt the log file, attach the log file to the SOAP message, then send the SOAP message to a receiver such as a Web Service. At the provider side, a predefined ontology checker checks the SOAP message and compares element positions with the predefined ontology structure. If there is any modification, the ontology checker clarifies it, and checks if it is a malicious intent or not. If the modification is not malicious the ontology checker passes the SOAP message to the policy filters. However, if the modification is malicious, the ontology checker decrypt the SOAP message captured in the log file and compares it to the predefined SOAP ontology structure. If the log file containing the SOAP message has not any structuring problems, the ontology checker modifies the message such that it becomes as its copy that is found in the attached log file. After that, the message is forwarded to the policy filters. If the SOAP message copy in the log file suffers from malicious vulnerability, then the ontology checker rejects the SOAP message.

In addition, the log file plays 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. The log file also specifies which intermediate node modifies the SOAP message and when has this modification happened.

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. To make the approach more time efficient, these four filters are set to execute in parallel as multiple threads.

The rest of the paper is organized as follows: Section 2 presents some basic definitions related to SOAP attacks. Section 3 includes a review of some related works. Section 4 covers the design of the domain ontology capturing the SOAP structure. Section 5 covers the structure and the details of the ontology-based approach for detecting SOAP message attacks. Section 6 presents the experimental results and evaluation of the proposed approach. Finally, Section 7 concludes the paper highlighting future improvements to the approach.

2. DFINITIONS

2.1. SOAP Message

Simple Object Access Protocol (SOAP) is a lightweight protocol intended for exchanging structured information in a distributed environment. It uses XML technologies to define an extensible messaging facility, providing a message structure that can be exchanged over a variety of underlying protocols such as HTTP and SMPT. The protocol has been designed to be independent of any particular programming model and other implementation specific semantics.

2.2. SOAP Message Attacks

There are many kinds of SOAP message attacks targeting the manipulation, transmission, and/or execution of a SOAP message. In our approach, we target the following types of attacks as defined in [6]:

- XML Rewriting attack: SOAP message is an XML-based document. One particular vulnerable case is that of a XML rewriting 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 [13], WS-Policy [4] and other standards that they can`t exactly specify the real location of the element.

- Buffer Overflow attack: The attacker inserts malicious content with well-formed message in a SOAP request, which is beyond the allowable size of the buffer and causes denial-of-service attack.

- 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 identity. For example, when a legitimate client transfers money from his account in a bank to the receiver, the attacker steals the password and uses it multiple times in order to cause money lose.

- 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 causing the crash of the receiver of the message.

- 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 receivers, e.g., 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-service attack.

3. RELATED WORK

Various approaches for dealing with SOAP message attacks have been proposed. Some of them is standardized such as WS-Policy [4]. Each of which has its advantages and disadvantages depending on what kind of attacks it deals with and in what scenarios is used. In [9], they pinpointed the limitations of existing Web Services security in providing end-to-end integrity of multiple signed parts in a SOAP message specifically in a document production workflow environment. Some severe SOA security threats and their implications on SOAP Web Services are analysed by [8]. They stress the possibility to identify the severity of SOA threats and the attackers' move on those threats in advance for each individual SOA applications keeping the opportunity for experts to design and implement security measures.

The kinds of attacks that can modify a SOAP message such as XML rewriting and signature's weakness is covered by [1]. They discuss existing solutions, present their limitations and make some recommendations to ensure the integrity of a SOAP messages. Additionally, [2] study some detection techniques of XML Rewriting attacks in Web Services communication. They present general countermeasures for prevention and mitigation of XML Rewriting attacks.

In [11], an inline approach is proposed to capture information about the structure of the SOAP message and adding a new header element called SOAP Account containing this information. If this header is tampered, it would not be easy to discover that and the message would be passed to the receiver. The inline approach is proposed to be fixed [7] by retuning XPath with position information, but this is not sufficient to detect all types of XML rewriting attacks.

In [18], three steps for detecting XML Rewriting Attacks are recommended. Firstly using shared key for encrypting timestamps in the message body for generating corresponding signature. Secondly, using value referencing both for signature validation and message processing. Finally encrypting the whole SOAP body instead of sending an open SOAP message in the network to prevent unauthorized access. Another encryption approach to prevent attacks of the SOAP message when exchanging data between server and clients [16]. They use a method with an encryption key of 256 bits length for the encryption algorithm. Dealing with authorizations and using access control for Web Services has been proposed by [5], they introduce new structures/elements in a SOAP message.

An XML injection strategy-based detection system is developed by [14] to mitigate the time gap for 0-day attacks resulting from an ontology attack variations. Because many new and unknown attacks are derived from known strategies, the system is developed to be hybrid such that it supports knowledge-based detection derived from a signature-based approach and applies an ontology to design the knowledge database for XML injection attacks against Web Services. A limitation of this system is focusing on one kind of SOAP message attacks.

An adaptable framework [17] applying agents, data mining and fuzzy logic techniques is designed to compensate for differences between anomaly and signature based detection for Web Services. Normal user behaviours and service request/response are captured, profiled and agents are then used to identify the suspected items. These suspected items are further analysed using sequential rule based techniques and fuzzy logic.

An ontology-based technique [3] is proposed to capture the SOAP message structure to avoid the wrapping-attack. They first build SOAP message elements structure using ontology and then attach it in SOAP message header. By validating the ontology in the receiving end, attacks can be detected early in validating process. All modifications on SOAP messages are written to a log. Therefore, if security failures occurs, this log is checked and recovered from effect of successful execution. The main limitation of this approach is that there is no mechanism to specify the parent in the message while it is an element in the header which we avoid in our proposed approach. Furthermore, it handles one kind of attack while our approach handles several types as will be presented later.

4. THE SOAP MESSAGE ONTOLOGY

In this section, we present the steps of developing the SOAP message ontology that is used by the approach presented in Section 5 for detecting SOAP message attacks. The ontology captures the structure of the SOAP message together with elements positions and integrity.

Step 1: The domain of the ontology is the SOAP message, which is a specific and limited domain and is used in detecting any manipulation that might occur to the SOAP message during the transmission process. Also the number and kind of such manipulations, the affected element(s) as also whether it is deleted or repositioned and its position in the message, and place and identity of the manipulator.

Step 2: Specifying class hierarchy and related properties of the ontology is the most important step in building the ontology. The class hierarchy together with some properties are shown in Figure 1 and they include:

- **SOAP-Message** class has the properties ParentOf. hasNode
- **Envelop** class has the properties: ParentOf, isExist, hasNode, and ChildOf.
- **Header** class has the properties Unique, isExist, ParentOf, hasNode, and ChildOf.
- **Body** class represents message main and mandatory content, purpose, input, or fault. It has the properties Unique, isExist and ChildOf.
- **MessageID** class has the properties Unique, isExist, and ChildOf.
- **Time-Stamp** class is a container of two main elements as its sub-classes, the creation and expire classes pertaining to the creation and the expiration times of the SOAP message. It has the properties Unique, isExist, ParentOf, hasNode, and ChildOf.
- **Security** class represents constrains a provider needs to add to a message such as signing or encryption. It has the properties Unique, ChildOf, and isExist.
- **Creation** subclass represents the creation time of the message and is a child of the Timestamps class. It is mandatory and has the properties Unique, ChildOf, and isExist.
- **Expire** class represents expiration time of the message and its absence leads to the inability to check Replay attacks. It has the properties Unique, ChildOf, and isExist.

There are two additional data properties namely, *hasValue* and *hasDepth*. The property *hasValue* applies to the **MessageID**

Creation and **Expire** classes as its domain with Integer and Date as its range. While the property *hasDepth* applies to the other classes as its domain and Integer as its range.

Step 3: Ontology slots have different facets that describe the value types, allowed values, the number of these values (cardinality), and other features of the values the slot can take. Most of the slot values in the SOAP message ontology are of integer type. For example, the value type of *hasDepth* property is integer and value type of *hasValue* in **Time-Stamp** class is Date.

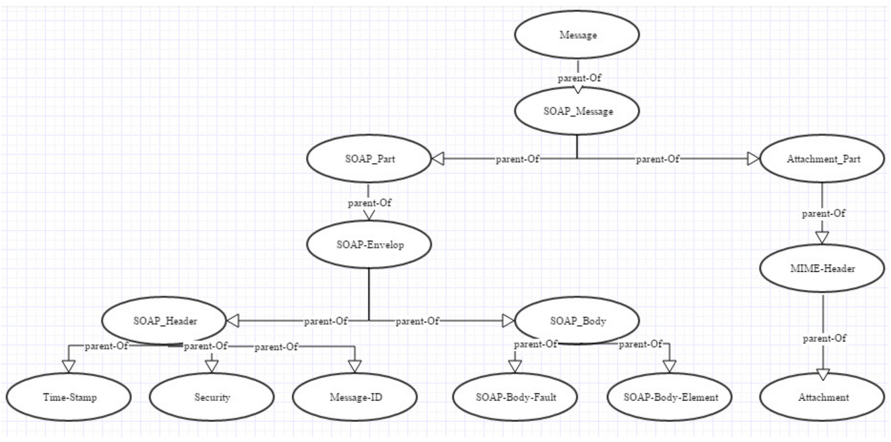


Figure 1. SOAP Message Structure in the Ontology

Step 4: After defining classes, properties, slots with facets, the ontology is populated with various instance (individuals) to be fully functioning and useful in the SOAP message attacks detection. The ontology with instance is called a knowledge base. Individuals include XML rewriting attacks cases to allow the properties of the classes to be record. Other individuals include cases that present the various types of modification or missing elements in the SOAP message, these cases are defined for each kind of the XML rewriting attacks.

Step 5: Finally, the ontology is checked for consistency and correctness through a reasoner and is evaluated through a number of SPARQL queries to retrieve/deduce the SOAP message relevant knowledge.

5. DETECTION OF SOAP MESSAGE ATTACKS

The proposed approach consists mainly of five components as shown in Figure 2, the ontology-based checker and the four policy filters. We illustrate the approach through the following steps:

1. The SOAP message is created at the sender side, its structure is captured and added to the log file in an encrypted manner.

2. The Log File contains the logging events of all modifications in the SOAP message during the transmission between the intermediary nodes. The log file is attached to the message as an attachment.

3. The ontology-based Checker resides at the receiver side and compares the SOAP message structure with its predefined structure in the ontology. Section 4 presents the SOAP message ontology where Figure 1 shows the predefined SOAP message structure represented in the ontology. If the checker finds any malicious modification or missing elements, it decrypts the SOAP message existing in the log file and compares the received message with it. If there is really a malicious modification or missing elements, the SOAP message is rejected. Otherwise, the ontology checker forwards the message and the log file to the filters. It is worth

mentioning that the ontology represents all kinds of modifications and element absences. If the SOAP message looks like any of these cases, the ontology checker will reject the message. Figure 3 shows the steps followed by ontology checker in checking the SOAP message for any modifications in its structure or any missing elements based on the ontology.

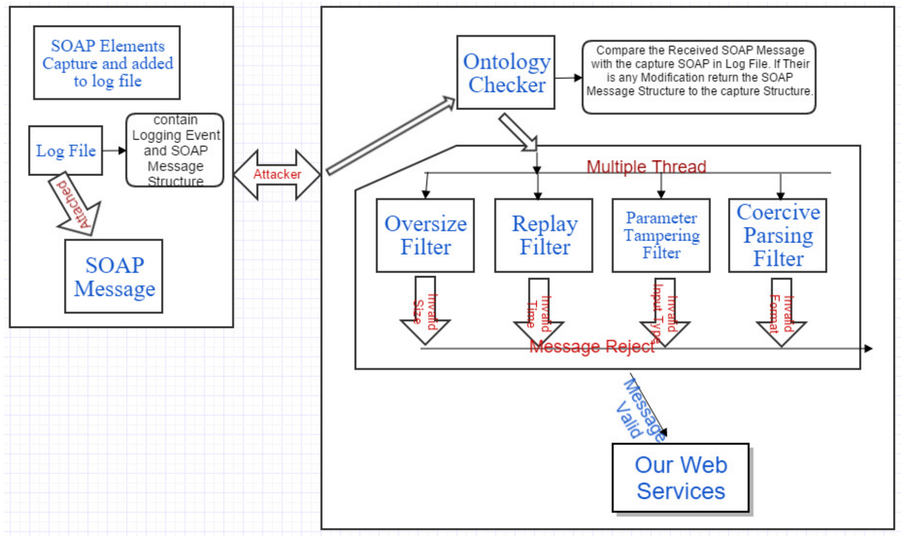

SOAP Elements Capture and added to log file
Log File
contain Logging Event and SOAP Message Structure
Attached
SOAP Message
Attacker
Ontology Checker
Compare the Received SOAP Message with the capture SOAP in Log File. If Their is any Modification return the SOAP Message Structure to the capture Structure.
Multiple Thread
Oversize Filter
Replay Filter
Parameter Tampering Filter
Coercive Parsing Filter
Invalid Size
Invalid Time
Invalid Input Type
Invalid Format
Message Reject
Message Valid
Our Web Services

Figure 2. Architecture of the Proposed Approach

4. The filters start their attacks checking procedure in parallel manner. Each filter is specialized in one of the SOAP message attacks. Figure 4 presents the procedure followed by the filters in checking the message for attacks. Next, we explain the role of each filter:

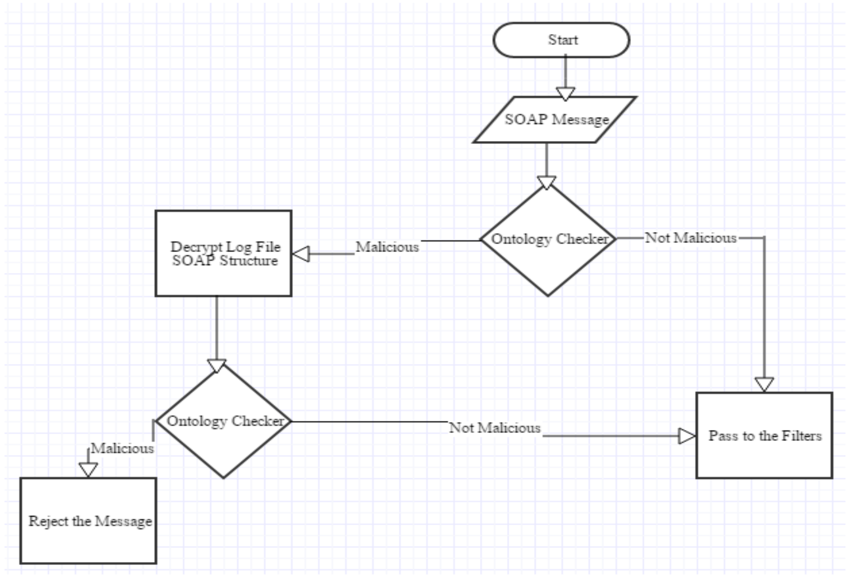


Figure 3. Ontology-Based Checker Procedure

a. The Oversized Filter focuses on the size of the SOAP message to avoid Overflow and Denial-of-service attack. It checks the size of the SOAP message and compares it with maximum allowed size of a SOAP message. If the message size is larger than the maximum allowed size, then the SOAP message must be rejected.

b. The Replay Filter focuses on the element value (creation element and expiration element). The creation element consists of the time when the SOAP message has been created. Consequently the SOAP message has limited live period which spans the time from creation to the expiration. The replay filter compare the current time with the expiration time if the current time is less than the expiration time, then message must be rejected.

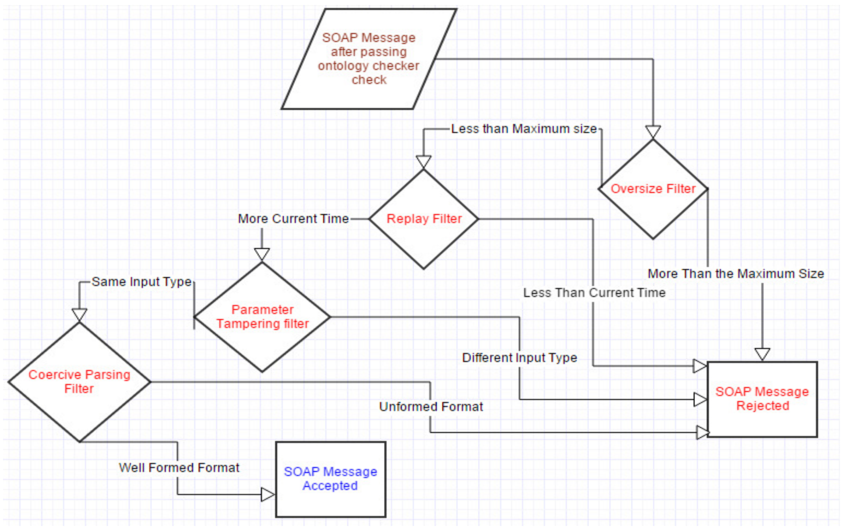


Figure **4**. Filters-Based Message Checking Procedure

c. Parameter Tampering Filter focuses on checking the input type of the SOAP message. If the sent input type is different from the expected type of the receiving side, then this causes a Daniel of Services attacks (DOSs). To avoid this attack the parameter tampering filter checks and makes sure that the input type is the same as the expected one by the receiving side input type. If not, the SOAP message must be rejected.

d. Coercive Parsing Filter focuses on checking the SOAP message format, e.g., in terms of version, namespace, or "Must Understand" header entry if the format of the SOAP message is fine, the message passed to the Web Services.

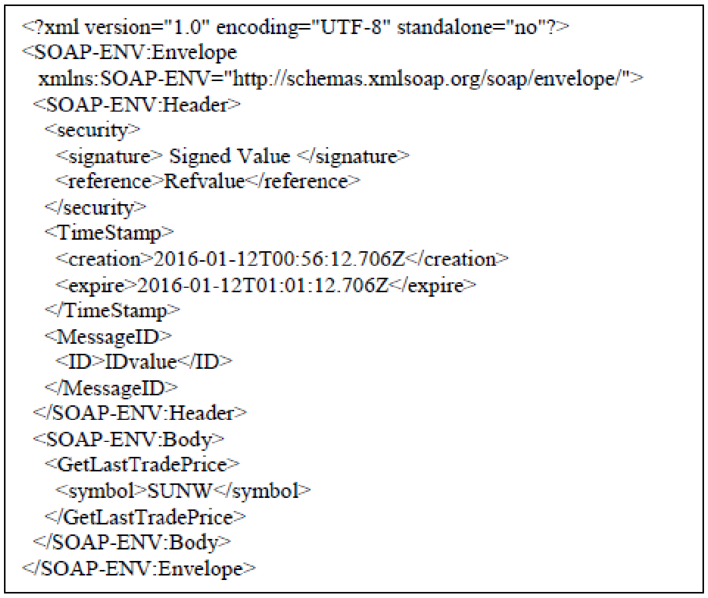
Each filter is realized as a separate thread running concurrently with the other filters therefore speeding up the filtering process.

6. Experimental Results And Evaluation

The approach is tested and evaluated through a set of experiments with six test cases: the normal case, the modified structure case, the replay case, the parameter-tampering case, the oversized case, and the coercive case. The approach is also evaluated for integrity and confidentiality as well as for time efficiency.

6.1. Checking a Non-Modified SOAP Message

In this test, the aim is to prove that the approach works fine in the normal case where a SOAP message is sent and received intact. A message, such as that in Figure 5. A SOAP Message without any Modification, is created and sent without any modifications or malicious content, and then it is passed to the ontology checker to test its structure where it successfully passed this test. After that, it is passed to the four concurrent filters and checked for oversize, replay, parameter-tampering and coercive parsing attacks. Since the message has not exposed to any such attack, it will also successfully pass the filters checks.



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

Figure 5. A SOAP Message without any Modification

6.2. Checking the Message Against XML Rewriting Attack

In this test, the SOAP message structure is modified as shown in Figure 6. A Modified SOAP Message where the yellow colored elements are modified, namely, the elements of Time-stamp, Creation and Expire are moved under the Security element. The ontology checker, upon receiving the message, checks the message structure, i.e., the existence of mandatory elements and their positions. It finds out that the message is modified and therefore returns it to its original structure (shown in Figure 5) based on the attached message in the log file.

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

Figure 6. A Modified SOAP Message

6.3. Checking the Message Against Replay Attack

Figure 7 presents a SOAP message with wrong time (an old time). This means that it has been

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

Figure 7. A Replay Attack

Exposed to a replay attack and according to the replay filter, this message must be rejected.

6.4. Checking the Message Against Oversized Attack

In this test, the maximum size of the SOAP message is determined to be 300 bytes. This is very small value but it is determined based on the average size of the expected message. We tempered the message such that it exceeds this maximum allowed size. Upon reaching the oversized filter, it is has been rejected. This indicated that the filter works successfully in intercepting oversized attacks.

6.5. Checking the Message Against Parameter Tampering attack

In this test, some typed elements (parameters) in body of the SOAP message are altered such that they have a different type. Since the receiver of the message expects a certain parameter types, the parameters in the body of the message are checked and compared to the expected ones found on the receiver side or on the message copy found in the log file. Since they are not the same, the parameter tempering filter discovers this and reject the message right away.

6.6. Checking the Message Against Coercive-Parsing attack

In this checking face, the SOAP message namespace is checked to determine which SOAP version this message complies with. SOAP has two versions and each version has its specific namespace. Figure 8 illustrates a message with a modified namespace, therefore, the coercive

Figure 8. Coercive Parsing Attack

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

parsing filter has detected this and considered the message as a malicious one.

These four filters execute concurrently in separate threads so that the message is checked for each type of attack at the same time. This allows to determine if the message is allowed or rejected right away and at the same time speeds up the process of execution and therefore improves the performance of the approach. We measure the speedup of concurrent execution in Section **Error! Reference source not found.**6.9.

6.7. Integrity

We define 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-based 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. Therefore, 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, Message ID, Creation, and Expire as well as the relation between them. Based on this definition, we classify the XML rewriting attacks using the ontology-based checker with 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

A similar approach to ours [3] tried to preserve the integrity of the SOAP message depending on copying the SOAP message in an element in the Header of the SOAP message. The main limitation of such an approach is that there is no 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 it which is more efficient and better in terms of integrity.

6.8. Confidentiality

Confidentiality refers to the requirement for the communicated data between two parties not to be available to a third party that may try to interfere into the communication. In order to achieve confidentiality, our approach uses encryption. The log file containing the original message is encrypted and in some case of sensitive information, the Body is encrypted. This is done through considering the Security element as a mandatory element to apply a signature on the Body and elements of the message.

6.9. Time efficiency

One main advantage of our approach is time efficiency because the ontology-based checker checks the message structure just once. In some cases as mentioned in the confidentiality, there is a need for decrypting the captured copy if it is encrypted. Filters are executed in a concurrent manner minimizing time of the filtering process.

To show time efficiency for the ontology-based checking, assume that the number of nodes is n, and the checking time for each node for the element position in the ontology-based checker is t. Therefore, the time taken by the ontology-based checker to check a non-encrypted message is t*n and to check an encrypted message is t*n*c where c is the decryption time.

We should mention that the time consumed for an SOAP message element position checking is a fixed time. In addition, 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 elements.

The concurrent execution of filters obviously enhances time efficiency of the approach. We measured the time improvement by executing the filters sequentially and then concurrently through multithreading. The sequential execution took 184071123 ns while the concurrent execution took is 5437086 ns. The time decrease in percentage between the sequential and the concurrent executions is:

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

The speedup (1/1-p where p is the percentage between the sequential and concurrent executions) is calculated as:

$$\text{Speedup} = (1/1-0.97) = 33.33 \text{ time.}$$

This performance improvement is more significant if large number of SOAP messages are exchanged.

7. CONCLUSIONS

We have built an ontology-based approach for detecting SOAP messages for rewriting as well as denial-of-service attack. An ontology has been 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. Four policy filters has been used to check if the SOAP message is vulnerable to Denial-of-service 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-based checker to check the existence and real positions of these elements. The ontology-based checker has been designed to check the structure of the SOAP message and compares it with the predefined ontology based structure to recognize any messing or modifications happened to the SOAP message during the transmission process. The filters started their checking right after the received SOAP message passes the ontology-based checker. 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 and therefore is considered as exposed to denial-of-service attack.

Our approach handles the attacks before the message reaches the ultimate receiver because it exists in the middle between the sender and the receiver. It handle more than one attack and works concurrently for time efficiency 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 shows a 33 times improvement of the concurrent execution over the sequential one.

As future work, the approach can be extended to deal with other attacks such as cross-site scripting. The ontology can be further extended to be used in the filtering process. Additionally, the approach can be enhanced to be adaptable such that it has the ability to cure the SOAP message against any missing or modification in its structure.

REFERENCES

[1] A. Benameur, F. A. Kadir and S. Fenet, (2008) "XML Rewriting Attacks: Existing Solutions and their Limitations", Proceeding of the International Conference on Applied Computing, pp 1-9.

[2] A. Nasridinov, J. Y. Byun and Y.-H. Park, (2014) "A Study on Detection Techniques of XML Rewriting Attacks in Web Services" International Journal of Control and Automation, Vol. 7, No.1, pp 391-400.

[3] A. Nasridinov, J. Y. Byun and Y.-H. Park, (2012) "UNRWAP: an approach on wrapping-attack tolerant SOAP Message," Second International Conference on Cloud and Green Computing, pp 794-798.

[4] Bajaj, et al., Web Services Policy Framework (WS-Policy), September, 2004, http://www.ibm.com/developerworks/library/specification/ws-polfram

[5] Damiani E., Vimercati S., Paraboschi S., Samarati P., (2001) "Securing SOAP E-Services, International Journal of Information Security", Volume 1, Number 2, pp 100-115.

[6] Elangovan Uma and Arputharaj Kannan, (2014) "Improving Cross-Site Scripting Filters for Input Validation Against Attacks in Web Services", Kuwait J. Sci. 41 (2), pp 175-203.

[7] Gajek S., Liao L., Schwenk J., (2007)"Breaking and fixing the inline approach", Proceedings of the 2007 ACM workshop on Secure Web Services, November 02-02, Fairfax, Virginia, USA, pp 37-43.

[8] I. B. Mohamed and S. ,. R. Dr. Mohamed, (2014) "Server SOA Security Threats on SOAP Web Services-A Critical Analysis", ISOR Juornal of Computer Engineering (ISOR-JCE), vol. 16, no. 2, pp135-141.

[9] Kumar Sinha, S.; Sinha, S., (2008) "Limitations of Web Service Security on SOAP Messages in a Document Production Workflow Environment," Advanced Computing and Communications, 2008. ADCOM 2008. 16th International Conference on , vol., no., pp 342-346.

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

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

[12] M. McIntosh and P. Austel, (2005) "XML Signature Element Wrapping attacks and Countermeasures," in Proceedings of the International Workshop on Secure Web services (SWS), Fairfax, VA, USA, pp 20-27.

[13] 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.

[14] R. K. Swati and D. Aarti, (2014) "A Survey On XML-Injection Attacks Detection Systems," International Journal of Science and Research (IJSR), vol. 3, no. 5, pp 1628-1631.

[15] U. Mike and G. Micheal,(1996) "Ontologies: Principle, Methods and applications," Knowledge Engineering Review, Vol 11, No 2.

[16] Waleed, G.M.; Ahmad, R.B., (2008) "Security protection using simple object access protocol (SOAP) messages techniques", International Conference on Electronic Design, ICED 2008. vol., no., pp1-6.

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

[18] Rajni Mohana, (2018) ”Aproposed SOAP Model in WS-Security to avoid Rewriting Attacks and Ensuring Secure Conversation” International Journal of Information Security and Privacy (IJISP). Vol 12, pp 74-88.