SOAP 메시지의 무결성 보장을 위한 자체 적응 접근법
(A Self-Adaptive Approach to Ensure Integrity of SOAP Messages)

1. Introduction

The WS-Security group of security standards is used to secure exchanges of SOAP messages in Web Service environment. However, although all of these security standards, certain attacks on integrity of SOAP messages, may still occur and lead to significant security faults.

Mcintosh and Austel [6] demonstrated that the integrity of a SOAP message, secured with XML Digital Signature, can be attacked without invalidating the signature. These types of attacks are called XML Rewriting attacks and they can be performed because XML Digital Signature refers to a signed object of an XML Document in a way that it does not take care of the location of that object. This weakness of XML Digital Signature may create a foundation to perform XML Rewriting attack on integrity of SOAP messages.

In order to deal with these types of attacks, we propose a self-adaptive approach to ensure integrity of SOAP messages. In this approach we first build SOAP message elements structure using ontology and then attach it in SOAP message header. If any changes in the pattern of attack are occurred, we learn them and save in a reliable storage. Also, in our approach, all modifications on SOAP messages are written to a log. Experiments show that the proposed solution has better performance in ensuring integrity of SOAP messages.

Key words: WS-Security, XML-based attack, SOAP message, self-adaptive approach

Abstract

Several Web Service Security (WS-Security) technologies are used to secure exchanges of SOAP messages in Web Service environment. However, although all of these security standards, SOAP message can still be vulnerable XML-based attacks. In order to deal with these types of attacks, in this paper, we propose a self-adaptive approach to ensure integrity of SOAP messages. In this approach, we first build SOAP message elements structure using ontology and then attach it in SOAP message header. If any changes in the pattern of attack are occurred, we learn them and save in a reliable storage. Also, in our approach, all modifications on SOAP messages are written to a log. Experiments show that the proposed solution has better performance in ensuring integrity of SOAP messages.

Copyright:©2012 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 다중성 복사본의 제작을 허가합니다. 이 때, 사용은 상업적으로 수단으로 사용할 수 없으며, 및 대안화에 본 문責에게 출처를 반드시 명시하여야 합니다. 이외의 목적으로 복사, 배포, 출판, 전송 등 모든 유효한 사용행위를 하려는 경우에 대하여는 사전에 허가를 받고 비용을 지불해야 합니다.

정보과학회논문지: 컴퓨팅의 실제 및 레터 제18권 제9호(2012.9)
The rest of the paper is organized as follows. In Chapter 2 we discuss related works. In Chapter 3 we demonstrate our proposed approach. Chapter 4 presents our implementation. Chapter 5 shows performance evaluation and Chapter 6 highlights conclusion and future work.

2. Related Studies

There have been many efforts to protect SOAP messages from XML Rewriting attacks. For example, in [10] authors proposed an inline approach that considers information about the structure of the SOAP message by adding a new header element called SOAP Account. In [4] the authors extended the inline approach by proposing to take into account new characteristics of SOAP message such as the depth information and parent elements of the signed node. In [3] it is proposed to use a new header in SOAP message containing the signed elements positions in the message.

In [7] we showed that above mentioned methods [3,4,10] cannot address all type of XML Rewriting attacks and proposed a new approach called Secure-Tolerant SOAP (ST-SOAP). In that approach, we developed ontology based detection and log-based recovery mechanisms to protect SOAP messages from XML Rewriting attack. However, this approach has low capacity to adapt itself to the changes in the pattern of attack, which may cause a reduction of its effectiveness. Current paper mainly reflects future directions of our previous research and enhances it by adding self-adaption ability.

In [8] authors proposed the self-adaptive architecture to deal with Denial–of–Service (DoS) attack in Web Service environment. DoS attack is a class of attack where an attacker aims to prevent legitimate users from accessing information or services. The main difference with our approach is that we try to protect Web Service environment from XML Rewriting attack. XML Rewriting attack is a distinct class of attack based on the malicious interception, manipulation, and transmission of SOAP messages.

3. System Design

The main idea of our approach is to extend existing message flow by proposing four new modules. These modules and their functions are as follows (Figure 1):

- The Initialization Module: In this module, elements structure of outgoing SOAP message is built and logged in a reliable storage.
- The Diagnostics Module: In this module, integrity of received SOAP messages is diagnosed and security report is generated.
- The Warning Analyzer Module: This module, based on security report, chooses the suitable recovery action for the detected problem. If the changes in patterns of attack or new attack are occurred, the structure of attacks is learned.
- The Recovery Module: the module applies the appropriate recovery actions chosen by the Warning Analyzer Module.

3.1 Initialization Module

We used ontology to build the SOAP elements structure. In the content of our proposed approach, ontology allows us to construct hierarchies and thesauri that can be used for expressing how elements within SOAP message relate to one another. So before sending any SOAP message we first build ontology of SOAP elements structure and include it in outgoing SOAP message's header. After ontology of outgoing SOAP elements structure is build and thus, message is ready to be sent, logging mechanism of the initialization module is activated.

3.2 Diagnostic Module

The diagnostics module resides on the receiver side and is responsible to diagnose the incoming message, detect all the security threats and generate security report. So, any legitimate receiver of the message, as soon as message arrives, builds ontology of incoming SOAP message elements structure and compares it with the attached structure information. The differences of these two ontologies are reflected in a security report.

3.3 Warning Analyzer Module

By observing the security report, generated by Diagnostics module, the receiver can immediately recognize if message was attacked. Warning Analyzer module is responsible to do this action. Specifically, the security report, based on comparison, indicates what sort of security vulnerability may exist and...
choose the suitable recovery action for the detected problem.

In order to deal with the problems such as changes in the pattern of attack, learning mechanism is designed. The idea of learning module is to solve new problems by adapting solutions that have been used to solve similar problems in the past. So if any changes in the pattern of attack are occurred, we learn them, save in a reliable storage and use them for the further detect actions.

3.4 Recovery Module

The module applies the appropriate recovery actions identified by the Warning Analyzer module. If message has been attacked and recovery action is needed, Recover module is in charge to retrieve the logs from the log DBMS and compliment the missing parts of SOAP message.

4. Implementation

We will describe an implementation of our proposed method. Apache Extensible Interaction System (AXIS) [1], a SOAP processor engine, is chosen for the developing and processing of SOAP messages. The prototype has been developed using NetBeans 6.9 environment.

Suppose a client requests loan from bank. To simulate the attacker in this scenario, we wrote a Java class called RewritingAttack which denotes the malicious host. After receiving the message from the sender, it moves <MessageID> element, which is used by the server to keep track of client request, into new bogus header element and creates its own <MessageID> element.

With this action, the attacker went through the attack detection checks, and by replaying this message, we may cause the same request to be processed several times, making the client pay for the same query and forcing the server to do redundant work. This is called Replay attack which is typical scenario of XML Rewriting attack. The result of this modification on SOAP message is shown in Figure 2.

In order to avoid these kinds of attacks, in the initialization module of our approach, we build ontology of outgoing SOAP message structure shown in Figure 3. For the purpose of this implementation, we build ontology using Protégé tool [9]. Jena Ontology API [5] is used for interrogating, manipulating, inferencing and querying the ontology. With ontology approach, we are able to detect replay attacks

![Figure 1. Proposed extension to existing message flow](image-url)
Figure 2 Replay Attack on SOAP message

Figure 3 Ontology schema of SOAP message structure

Let’s consider the scenario where our Java attacker class with the slight variation as above method, redirects a SOAP request. Specifically, it wraps the signed <ReplyTo> element, which is used to specify URI of the server, under a bogus header and introduces different <ReplyTo> element with different URI. As we saw in the previous example, the application logic will consider the <ReplyTo> element introduced by the attacker instead of the <ReplyTo> element that was wrapped by the bogus header. As a result the message will be redirected to a different location.

In order to deal with this problem, learning mechanism is designed to learn and adapt to changes in attack pattern. When the attacker class modifies <ReplyTo> element of SOAP message, we learn pattern of attack. Specifically we register attacks on <ReplyTo> element as a redirection attack and save in a reliable storage. So if the same attack on <ReplyTo> element is performed, we can use experience obtained from the previous solution in our reliable storage.

5. Performance Evaluation

We have used Colored Petri Net (CPN) [2] tools to conduct performance evaluation. The simulation of self-adaptive approach in CPN tools is shown in Figure 5.

The model starts working from WS-Requester place. This place holds all initial SOAP requests. For the purpose of our simulation, we have created 100 SOAP messages. As the message flows, the WS-Security transition will intercept the message and enrich it with security headers. As soon as enrichment is done, the request will be logged. In Attack transition, we have simulated several typical XML rewriting attack cases such as Replay Attack, Redirection Attack and Multiple Header Attacks. Once attacks are simulated, Detection Mechanism transition is activated. In this transition we simulated how our system performs with self-adaptive approach and without. The result of both approaches is reflected in Figure 4. From this graph, it is easy to recognize that our approach with self-adaptive approach performs much better than without it. Thus, we believe it can be effectively utilized in battling against XML Rewriting attacks.

With self-adaptive approach new attacks are recorded in Learning Mechanism place and are used to detect for the further detect actions. Learned data (in our model we learn changed pattern of attacks) are divided into three parts:

1. The training set is used to fit the model and consists of 50% of whole dataset.
2. The validation set is used to verify the model and consists of 25% of whole dataset.
3. The testing set is used for testing the usability of the model and consists of 25% of whole dataset.

Once detection process is done and threats are
identified, we pass security report to Recovery Mechanism transition. If security failures are occurred, we could easily check the log and recover from effects of successful execution.

6. Conclusion

In this paper, we proposed a self-adaptive approach to ensure integrity of SOAP messages. In this approach, we used experience obtained from similar problems in the past and then adapted successful solutions to the current problem. Results of implementation and performance have shown that our approach showed can protect integrity of SOAP messages from XML Rewriting attacks and so bring a reasonable protection to entire Web Service environment.

However in reality, Web services may be vulnerable not only to security faults but to all the categorized classes of faults. Therefore, in future, we will extend our system to cope with not only security, but with other faults as well.

Reference