A Tolerant Scheme for SOAP Attacks in Web Services Security

Pham Phuoc Hung, Aziz Nasridinov, Lin Qing, Jeongyong Byun
Department of Computer Engineering, Dongguk University
e-mail :hung205a2@yahoo.com, aziz_nasridinov@yahoo.com , qinglin9@gmail.com, byunjy@dongguk.ac.kr

웹서비스 보안에서 솝공격에 대한 관대한 계획

밤복흥, 아지즈 나스리디노프, 림청, 변정용
동국대학교 컴퓨터멀티미디어학부

Abstract

Nowadays Web Services are one of the most rapidly developed technologies and have become platform for e-ecommerce as well as B2B model. Therefore, when Web Services use SOAP as a protocol for communication, their security should be considered. However, those SOAP messages are prone to XML attacks that can create a foundation for typical faults and make it vulnerable to use. Unfortunately, recent researches established that solutions to deal with these problems have several limitations. In this paper, we explore attacks on SOAP messages and also provide confidentiality and integrity solutions. It is a tolerant scheme which is able to automatically detect and fix typical faults occurred in SOAP messages to combat with the security threats in order to improve its reliability.

1. Introduction

Web Services technologies are used for facilitating the integration of applications in form Enterprise Application Integration (EAI) and B2B systems via SOAP (Simple Object Access Protocol). This protocol supports the delivery of SOAP messages which carry data from the sender to the receiver [1]. Therefore, once SOAP messages contain significant information, their integrity and confidentiality is critical to the security of the Web Services. However, there is much vulnerability in XML documents caused by several external factors such as Denial of Service (DoS), Oversize Payloads, Replay attacks, XML rewriting attacks, parameter tampering attacks and so on. These attacks cause damages to the SOAP messages and make it vulnerable to use in Web Services environment. Unfortunately, recent researches established that solutions to deal with these problems have several limitations. Thus the security of SOAP message is a challenge of system integration.

In this paper, we discover some attacks on SOAP messages in SOA such as such as DoS attacks, XML rewriting attacks, parameter tampering and provide a tolerant scheme to deal with these threats on confidentiality and integrity of SOAP messages. It is a system which is able to detect and fix typical faults in SOAP messages to combat the security threats in Web Services environment. It allows system to self-optimize its performance in different conditions and improve the reliability of Web Services.

The remainder of this paper is made up of as follows. In Section 2 related study are discussed. Section 3 describes some kinds of attacks in SOAP message. Section 4 illustrates the motivating scenario and system design. Our System Implementation is presented in section 6. The paper will end with conclusion and future work.

2. Related Studies

Similar classification and functional relationships were explored in various discovery working groups.

In [2,3], researchers describe solutions to prevent XML rewriting attacks on SOAP messages by signing SOAP messages structure in it header. So the SOAP messages can be safe from intermediates to the final receiver. But the problem can happen if Web Services' clients are attackers. They may inject some dangerous data by using XML Injection or SQL Injection on SOAP messages before sending them to providers. In order to deal with this problem, we propose a system has a XML schema validator and a SQL validator to validate SOAP message to prevent XML Injection and SQL Injection.

In [4], authors developed a technique called Bit-Stream which is able to automatically detect the vulnerabilities and risks, while offering advice for higher security. In [5], a system named Security Description Assistant is proposed to identify and fix typical faults in SOAP messages because of XML rewriting attacks. In this paper, we have employed more comprehensive functions to detect and possibly fix faults occurred due to attacks on SOAP messages. Specifically, our current system is able to detect XML injection attacks, XML tempering attacks, content reordering, deleting attacks, Denial-of-Service, Oversize Payloads, and Replay attacks. We will provide details of each function in next sections. By that the security of the SOAP message is improved.

3. Attacks on SOAP messages

In this section, some possible attacks on SOAP messages are discussed here.

A. Parameter tampering attacks:

Parameter tampering attacks based on the manipulation of parameters exchanged between client and server in order to change application data, such as user credentials and permission, price, quality of products, etc. The parameter tampering attack can be subdivided into XML Injection and the SQL Injection attack [6].

B. XML rewriting attacks:
XML rewriting attacks refers to a message modified by malicious attackers without altering the signature.

C. DoS Attacks:
Denial of Service attacks are used by attackers to prevent the service from functioning as intended. These attacks are usually implemented by depriving the application of its resources [7]. Attackers can send a large amount of data as input to the application so that the application overwrites other instruction or they can send repetitive SOAP message, a recursive payload in an attempt to overload a Web Service.

D. Error Handling
Attackers can obtain the sensitive information from the internal state of Web Service by catching uncaught exceptions as SOAP fault elements within the response to the client.

For above mentioned attacks, we propose a tolerant scheme to deal with them. Next step illustrates the System Design of our proposal.

4. System Design

We first explore Motivating Scenario where SOAP messages can be attacked and then describe proposed system design.

4.1 Motivating Scenario

Consider a scenario where a customer connects to retailer shop's Web Service going to update its order. In this scenario, we assume that customer and retailer shop have established reliable connection. Figure 1 demonstrates SOAP messages after it was being exposed to attack. The red box in the figure shows that the attacker injected, deleted some part of SOAP messages and changes some values of some parameters.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <Header>
        <To xmlns="http://www.w3.org/2005/08/addressing">http://localhost:808
        <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
        </ReplyTo>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">
            uuid:181ce3ef-5545-42d4-a466-c8e71860a5af
        </MessageID>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">
            uuid:181ce3ef-5545-42d4-a466-c8e71860a5af
        </MessageID>
    </Header>
    <Body>
        <Order>
            <username>Jack</username>
            <password>Jack' OR '1=1</password>
            <productname>Tyre</productname>
            <quanlity>wwwww</quanlity>
            <price>450</price>
        </Order>
    </Body>
</Envelope>
```

Figure 1. SOAP message after attack

Like scenario shown in this subsection, the SOAP message may be attacked and it may lead to significant vulnerabilities. Especially, the attacks in previous section can be considered as a serious threat to the security of the SOAP message. A proposal for this XML Injection problem is to validate the input parameter by using XML schema validation which is an important measure for checking the syntactical correctness of incoming message [8]. And a solution preventing SQL Injection attacks is to filter the input parameter extracted from SOAP message by making predefined setting and using the store procedure to manipulate database instead of normal SQL commands and catch all errors to prevent attackers from getting the sensitive information from the internal state of Web Service.

If the attacker tries to use DoS attacks by sending a large SOAP message so that the application overwrites other instruction or sending repetitive SOAP message, a recursive payload in an attempt to overload a Web Service, we can easily prevent them by using DoS validator as following:

- Use XML - SAX Parser instead of XML – DOM Parser.
- Check for buffer overflows by limiting the size of the SOAP request to avoid large XML SOAP messages.
- Perform checks a recursive payload in the program flow by giving a strict depth of SOAP message.
- Checking repetitive SOAP message requests in an attempt to overload a Web Service by issuing a limitation of SOAP messages in a specific period.

In case that a provider does not have any tools to detect the SOAP message to reduce the vulnerability, it is real risk for his/her business. The reliability of the Web Services decreases. Therefore, we propose our system to solve those problems as a whole and thus provide more reliable message exchange in Web Services.

4.2 System Architecture

Figure 2 represents the general design of the proposed system flow module. The system architecture is structured in following three layers:

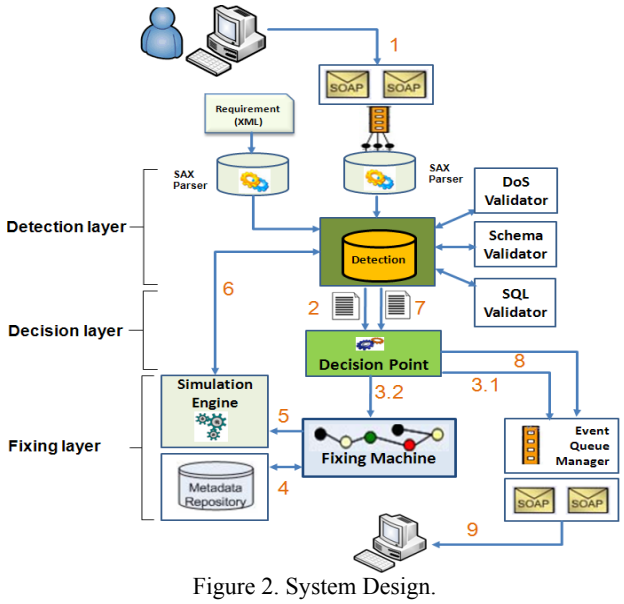

Figure 2. System Design.

Detection layer: The layer where SOAP request is received from Web services client, Detection Management will use Bit-Stream technique to analyze, detect faults and produce Security Report.

Decision layer: The layer where according to the Security Report, Decision Point will make a decision if the SOAP request should be fixed or not. Decision layer also chooses the optimal SOAP message among fixed ones.

Fixing layer: The layer where SOAP request is being fixed. We used simulation engine to generate alternative SOAP messages.

4.3 System Sequence Chart

Our system architecture is described in details through system sequence chart in Figure 3. Web service client sends a message to Web services provider to get some information. We subdivided explanation into several parts to make it more comprehensible. Each part corresponds to a numerated section of the figure.

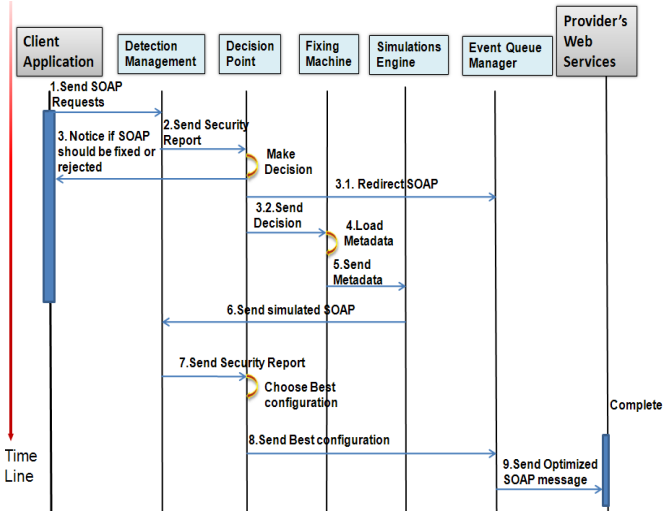

Figure 3. System Sequence Chart.

① Client request is parsed by SAX parser instead of DOM parser and accepted in Detection Management.

② Detection Management asks the DoS validator, the schema validator as well as the SQL validator to perform operations to determine the safety of this message for the system. The content of SOAP message have to be followed by DoS validator, XML Schema of the schema validator and be marshaled by regular expressions of the SQL validator. If not, the system will reject this SOAP message. If SOAP message passes this step, systems will analyzes it and send security report to Decision Point.

③ After receiving security report from Detection Management, Decision Point first notifies customer about decision.

- 3.1 If SOAP message does not need to be fixed then the system will redirect SOAP message to Event Queue Manager.
- 3.2 If SOAP should be fixed then the system sends it to Fixing Machine.

④ In case if SOAP message should be fixed, Fixing Machine receives decision from Decision Point and loads a set of metadata from database. Metadata has corresponding instructions to fix SOAP fault elements.

⑤ Loaded metadata transmits to Simulation Engine and Simulation Engine generates SOAP messages according to provider's requirements.

⑥ Newly generated SOAP messages are sent to Detection Management.

⑦ Detection Management receives the SOAP messages, analyzes them and sends the security reports about newly generated SOAP messages to Decision Point.

⑧ Based on security reports Decision Point chooses the best result and sends it to the Event Queue Manager.

⑨ Event Queue Manager sends the optimized SOAP message to provider.

After finishing above steps, the original SOAP message has been optimized and now it satisfies the provider's security requirement. The system has detected and fixed faults in SOAP messages. Thus reliability of Web Services is improved.

5. System Implementation

This section will show the experimental results of our system. Experimental results reflect XML attacks on SOAP message (figure 1) which was mentioned in section 4. We can see some vulnerabilities in this SOAP message such as quantity's value is "wwwwww" and password's value is Jack' OR '1=1. "wwwwww" is not valid for quantity which should be a integer. And Jack' OR '1=1 is often used for SQL Injection. For example, UPDATE user SET quantity=3500 WHERE username= 'Jack' AND password='Jack' OR '1=1'. This SQL command can be executed although the password Jack' OR '1=1 does not match with the password in database. However, those vulnerabilities can be easily identified by the following regular expression pattern (Figure 4) and a part of XML schema (Figure 5).

```
((?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%]).{6,20})
```

Figure 4. Password Regular Expression Pattern

```
<xs:element name="Header" type="Header"/>
<xs:complexType name="Header">
  <xs:sequence>
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unboun
  </xs:sequence>
  <xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
<xs:element name="Body">
 <xs:complexType>
  <xs:sequence>
    <xs:element ref="order" minOccurs='1' maxOccurs='1'/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="order">
 <xs:complexType>
  <xs:sequence>
    <xs:element ref="username" minOccurs='1' maxOccurs='1'/>
    <xs:element ref="password" minOccurs='1' maxOccurs='1'/>
    <xs:element ref="productname" minOccurs='1' maxOccurs='1'/>
    <xs:element ref="quanlity" minOccurs='1' maxOccurs='1'/>
    <xs:element ref="price" minOccurs='1' maxOccurs='1'/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="username" type="xs:string"/>
<xs:element name="password" type="xs:string"/>
<xs:element name="productname" type="xs:string"/>
<xs:element name="quanlity" type="xs:integer"/>
<xs:element name="price" type="xs:decimal"/>
```

Figure 5. XML Schema Validator

Therefore, that SOAP message is rejected by the system. And the system announces Web Service Client to know about that. We assume that clients send to the providers another SOAP message as Figure 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <Header>
        <To xmlns="http://www.w3.org/2005/08/addressing">http://localhost:806
        <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
        </ReplyTo>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">
            uuid:181ce3ef-5545-42d4-a466-c8e71860a5af
        </MessageID>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">
            uuid:181ce3ef-5545-42d4-a466-c8e71860a5af
        </MessageID>
    </Header>
    <Body>
        <Order>
            <username>Jack</username>
            <password>asefghaerthfegrghjae</password>
            <productname>Tyre</productname>
            <quanlity>280</quanlity>
            <price>450</price>
        </Order>
    </Body>
</Envelope>
```

Figure 6. SOAP message

The detection result is as following.

Element	Importance Level	Require	Result	Parent	Parent Detection	Quantity
S:Envelope	30	1	1	#document	#document	1
S:Header	29	1	1	S:Envelope	S:Envelope	1
S:Body	15	1	1	S:Envelope	S:Envelope	1
To	3	1	1	S:Header	S:Header	1
Action	10	1	0	S:Header		0
ReplyTo	9	1	1	S:Header	S:Header	1
MessageID	5	1	1	S:Header	S:Header	2

Bit Stream: 1 1 1 1 0 1 1
Security Level: 6/8
Define weak: <70% , medium : 70-> 90% , high : > 90%
Security Degree: 91/106= 86 %
=>Evaluate Security Degree: Medium
Advice: Insert element Action in SOAP message

Figure 7. Security Report before fixing

After detection phase, two faults are detected. These are, Action element is deleted and MessageID element is added. The system generates a security report shown in Figure 7 with Bit-Stream String of 1111011 and security level of 6/8. And fix corrupted elements. Through this Bit-Stream String, system can quickly recognize which element has fault. In our case it is Action element. After fixing, Action element is added to SOAP message as Figure 8.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <Header>
        <To xmlns="http://www.w3.org/2005/08/addressing">http://localhost:808
        <Action xmlns="http://www.w3.org/2005/08/addressing">http://calculato
        <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
        </ReplyTo>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">
            uuid:181ce3ef-5545-42d4-a466-c8e71860a5af
        </MessageID>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">
            uuid:181ce3ef-5545-42d4-a466-c8e71860a5af
        </MessageID>
    </Header>
    <Body>
        <Order>
            <username>Jack</username>
            <password>asefghaerthfegrghjae</password>
            <productname>Tyre</productname>
            <quanlity>280</quanlity>
            <price>450</price>
        </Order>
    </Body>
</Envelope>
```

Figure 8. SOAP message after fixing

This Bit-Stream should be called up again to analyze newly generated SOAP message. So after analyzing SOAP message, we have following bits 11111111 with security level (7/8) which indicates that SOAP message is enough secured. This result is shown in Figure 9.

Element	Importance Level	Require	Result	Parent	Parent Detection	Quantity
S:Envelope	30	1	1	#document	#document	1
S:Header	29	1	1	S:Envelope	S:Envelope	1
S:Body	15	1	1	S:Envelope	S:Envelope	1
To	3	1	1	S:Header	S:Header	1
Action	10	1	1	S:Header	S:Body	1
ReplyTo	9	1	1	S:Header	S:Header	1
MessageID	5	1	1	S:Header	S:Header	2

Bit Stream: 1 1 1 1 1 1 1
Security Level: 7/8
Define weak: <70% , medium : 70-> 90% , high : > 90%
Security Degree: 101/106= 95 %
=>Evaluate Security Degree: High

Figure 9. Security Report after fixing

6. Conclusion

Web Services are gaining rapidly because they are easy to use and independent of platforms and languages. Accompany with them, security vulnerabilities in SOAP messages are more and more serious. However, most of the approaches that have been proposed do not deal with protecting SOAP message as a whole. On contrast, in this paper, we explored some kind of attacks on SOAP messages and proposed a tolerant scheme which is able to detect and fix typical faults in SOAP messages against SOAP-based attacks. Experimental result shows us that various XML attacks, such as XML injection, SQL injection, DoS attacks, reordering attacks and elimination of important parts of message are detected and possibly fixed according to Web Service provider's security requirements. So the SOAP messages can be safe in both content and structure. The reliability of the SOAP is higher.

In our future work in this direction we are trying to research overhead of the proposed solution and explore the limitations if exists. We also plan to conduct a performance evaluation by comparing the current approach with other existing solutions.

Reference

[1] Tawfiq S. Barhoom, Raed S. K. Rasheed, "Position of Signed Element for SOAP Message Integrity", ISSN 2229-5208, 2011.
[2] Smriti Kumar Sinha, Azzedine Benameur, "A Formal Solution to Rewriting Attacks on SOAP Messages", SWS'08, 2008, Fairfax, Virginia, USA.
[3] Azzedine Benameur, Faisal Abdul Kadir, Serge Fenet, "XML Rewriting Attacks: Existing Solutions and their Limitations", IADIS Applied Computing 2008.
[4] Pham Phuoc Hung, Aziz Nasridinov, Jeongyong Byun, "Importance-Based Security Level Verification in Web Services", KIPS Spring Conference, 2010.
[5] Pham Phuoc Hung, Jeongyong Byun, "An Enhanced Description Assistant for SOAP Message Exchange in SOA", KCC, 2011.
[6] Navya Sidharth, Jigang Liu, "Intrusion Resistant SOAP Messaging with IAPF", 2008 IEEE.
[7] Irfan siddavatam, Jayant Gadge, "Comprehensive Test Mechanism to Detect Attack on Web Services", -1-4244-3805-1/08/2008 IEEE.
[8] Nils Gruschka, Luigi Lo Iacono, "Vulnerable Cloud: SOAP Message Security Validation Revisited", 2009 IEEE DOI 10.1109/ICWS.2009.