Security Testing Methodology for Evaluation of Web Services Robustness - Case: XML Injection

Marcelo Invert Palma Salas, Paulo Lício de Geus, Eliane Martins
Institute of Computing, UNICAMP, Campinas, Brazil
E-mail: {mpalma, paulo}@lasca.ic.unicamp.br
eliane@ic.unicamp.br

***Abstract*—Web services, due to their distributed and open nature, raise new challenges for information security. This technology is susceptible to XML Injection attacks, which would allow an attacker to collect and manipulate information in order to insert malicious code in servers and clients. Different studies show that current detection methods for these vulnerabilities, such as penetration testing and fuzzy scanning, generate many false positives and negatives. The fault injection technique has greater flexibility to modify the tests and find vulnerabilities, consequently improving robustness in web services. This research describes a fault injection technique for evaluation of web services robustness with WS-Security (UsernameToken) and the development of a set of rules for vulnerability analysis and the reduction of false positives and negatives. Furthermore, the results show that 82% of web services tested are vulnerable to XML Injection attacks, the most employed attack against web applications according to the OWASP Top 10.**

Keywords*—*Web services; XML Injection; fault injection; WS-Security; UsernameToken.

I. Introduction

Security incidents take place due to exploitation of vulnerabilities made during system development or failures not found in the development platform. There are numerous causes for vulnerabilities, among which we mention system complexity and the lack of a mechanism to check the inputs provided to programs (e.g. web services). An attack that exploits such vulnerabilities, maliciously or not, may compromise any of the information security attributes[1]. The result of a successful attack is an intrusion to the system [1].

Injection Attacks, such as XML Injection or Cross-site Scripting (XSS) may be consequences of intercepting and modifying messages. The targets of these attacks are search for vulnerabilities in the server-side, in order to execute malicious commands and to access unauthorized data or even to gain control of the server. Among injection attacks are: XML Injection, SQL Injection, XPath Injection, Cross-site Scripting (XSS), Cross-site Request Forgery (XSRF), Fuzzing Scan, Invalid Types, Parameter Tampering, Malformed XML and Frankenstein Message (Timestamp Tampering) [2], [3], [4].

XML Injection is an attack technique used to manipulate or compromise the logic of web services through the insertion of an unwanted content or structures into a SOAP message. Moreover, the XML Injection attack can cause the insertion of malicious content into the resulting SOAP message/document, i.e. malware [3], [4].

[1]Confidentiality, integrity and availability.

In this research, we propose to use a variation of the security testing methodology oriented to communication protocols [5] with the objective to evaluate the robustness of web services through of XML Injection attacks. For this purpose, we develop scripts for WSInject fault injector. This tool can emulate different types of attacks such as XML Injection, Cross-site Scripting (XSS), Brute Force attacks, Middleware Hijacking, among others.

The results show that 82% of the web services tested have vulnerabilities to this attack, e.g. the response returned information about path directory, database, function library, among others. The use of UsernameToken does not guarantee a meaningful reduction of vulnerabilities. In this case, it is recommended the use of XML Encryption and XML Signature. The former encrypts information between the client and the server while the latter can be used to verify if the information was not modified during its communication [2].

Finally, this paper is organized as follows. Section II describes fault tolerance, fault injection techniques and XML Injection attack used in this research. Besides to describe the challenges and related works in web services Security, the Section III analyzes the security in web services through fault injection technique. The security testing methodology for web services is described in Section IV. Section V shows our approach and experimental study. Section VI concludes the article, emphasizing its main contributions and addressing for future work.

II. Fault Tolerance and Fault Injection Technique in Web Services

In 1990, the IEEE defined "fault tolerance" as the ability of a system or component to continue normal operations even in the presence of faults [6]. Prevention and removal of faults, however, are not sufficient when the system requires high reliability and availability. In such cases, the system must be built using fault tolerance techniques to ensure its correct operation, even under faults, errors and defect events.

In this research, we use this sequence (fault→error→flaw) to find faults through the model of attack→vulnerability→intrusion, applied to malicious faults. This model limits the fault space of interest to the composition (attack+vulnerability)→intrusion [7]. Fig. 1 classified –in schematic form– the security faults types such as malicious faults (attacks) and accidental faults.

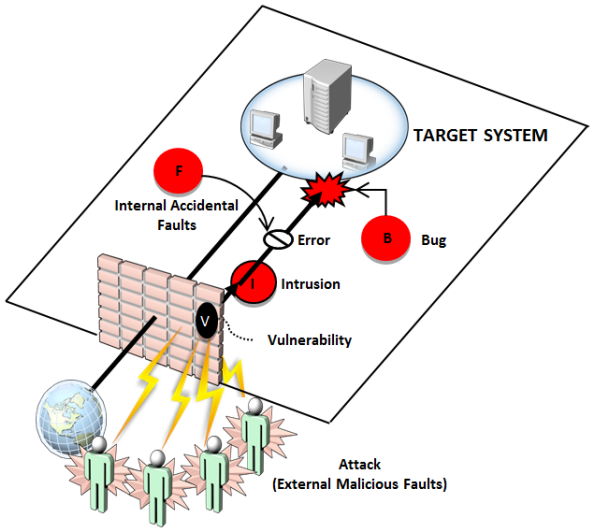

Fig. 1. Security threats and vulnerabilities.

A. Fault Injection Technique

By following the best practices of software testing and standards, languages and techniques have been developed in order to analyze and detect vulnerabilities on systems [1]. One such technique is fault injection.

Fault Injection can be used to assess aspects of computing systems dependability through observation of its behavior under a stressful environment. This technique emulates errors, failures or anomalies in the target system through of a fault injector tool. Also, this technique can be used to validate fault tolerant systems, assisting in the removal and prevention of faults while minimizing its occurrence and severity [8].

In this technique, the tests consist of two input sets: the workload and the faultload. The former represents the usual entry to the system that serves to activate its functionality, while the latter represents the faults to be introduced.

However, the use of this technique over black-box testing can generate a high percentage of false positive and negative. In order to reduce erroneous responses, is recommended to develop a set of rules to validate each response (see Sec. V-B).

For our research, we use the WSInject fault injector. This tool allows us to emulate XML Injection, inserting code –as part of SOAP message– and record the answers in its database.

The use of the fault injection technique, when compared to others, allows for the emulation of various types of attacks by varying parameters and data, including penetration testing and fuzzing scan techniques.

B. XML Injection Attack

XML Injection is a type of Injection Attack that modifies the SOAP message structure (or any other XML document) by insertion, removal or duplication of labels (tags). The goal of this attack is to insert malicious content in the resulting message. If the web service considered a valid message, its processing can cause undesirable effects such as disclosure information of the path directory, access to databases and XML files with usernames and passwords, among others [3].

For example, a bank maintains communication with their users through web services. Each user must send his username, password and user account to user authentication web service (Fig. 2).

Web service of user authentication

```
1: <bank:authentication>
2:     <user>
3:     <user:username>Alice</user:username>
4:     <user:password>4w3rjsolfo3prye9ort8</user:password>
5:     <user:account>4859-3</user:account>
6:     </user>
7: </bank:authentication>
```

Fig. 2. Alice user authentication using by web services.

Suppose that communication between the bank and users do not use encryption (XML-Enc) or digital signatures (XML-Sig), e.g, Alice wants to transfer money to Bob through money transfer web service (Fig. 3). For that, she needs to authenticate (i.e. send her username, encrypted password) and transfer the money to Bob (i.e. amount, his username and account).

Web service of money transfer

```
1: <bank:sendmoney>
2:     <user:username>Alice</user:username>
3:     <user:password>4w3rjsolfo3prye9ort8</user:password>
4:     <user:account>4859-3</user:account>
5:     <user:pin>840580</user:pin>
6:     <user:amount>1000,00</user:amount>
7:     <user:username_receiver>Bob</user:username_receiver>
8:     <user:account_receiver>2598-0</user:account_receiver>
9: </bank:sendmoney>
```

Fig. 3. Alice authentication and money transfer to Bob through web service.

These conditions are favorable for a XML Injection attack, i.e. an attacker discovers this vulnerability in the money transfer web service. To carry out the attack (Fig. 4), the attacker add another user to receive the money (i.e add new tag between <user:username_receiver>Bob and </user:username_receiver> and account (i.e. add new tag between <user:account_receiver>2598-2 and </user:account_receiver>). Now, the new SOAP message is sent to server.

XML Injection attack to steal money

```
  1:  <bank:sendmoney>
2-6:    <!-- Alice authentication… -->
  7:    <user:username_receiver>Bob
  8:      <user:username_receiver>attacker
  9:      </user:username_receiver>
 10:    </user:username_receiver>
 11:    <user:account_receiver>2598-0
 12:      <user:account_receiver>4982-3
 13:      </user:account_receiver>
 14:    </user:account_receiver>
 15: </bank:sendmoney>
```

Fig. 4. XML Injection attack in the sendmoney web service.

In the best case, the attacker generates an error in the server-side, which returns the HTTP status-code 400 Bad Request. In the worst case, the web service can response with the HTTP status-code 200 OK, i.e. the request has succeeded.

III. Security Evaluation of Web Services

Under the concept of SOA (Service Oriented Architecture), web services are in constant communication with other services[2]. Their clients make requests for services through a communication channel such as the Internet, sending and receiving information simultaneously. Another benefit is the possibility to develop web services in different languages and platforms. This technology transmits their information using two protocols, XML and HTML.

In this section we describe several methods to protect web services against security attacks. Furthmore we review the challenges and the state of the art of security testing in web services.

A. Security in Web Services

Every day, new vulnerabilities and attacks are found. As a result, the W3C[2] has developed various specifications to protect web services. The first specification proposed was WS-Security (WSS). This specification incorporates integrity and confidentiality to protect messages and allows the communication of various Security Tokens [9], such as UsernameToken, SAML[3], Kerberos and X.509. XML Signature [10] (XML-Sig) defines rules to generate and validate digital signatures expressed in XML to protect the integrity and authentication of the SOAP Message. XML encryption [11] (XML-Enc) describe the encryption process for any type of data and its XML representation to protect its confidentiality.

These specifications can be partially or fully implemented in the wer service, allowing multiple users to encrypt and sign parts of the message, providing greater security in end-to-end communication[12]. In Fig. 5, we show the stack of WS-Security specifications. Because our interest is in the WS-Security and Security Tokens, the reader can find in [2] and [13] more information about WS-Security and its specifications.

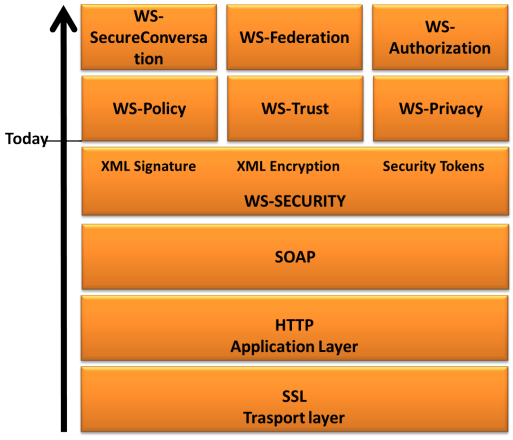

Fig. 5. The WS-Security Stack.

[2]World Wide Web Consortium (W3C)
[3]Security Assertion Markup Language

B. Security Tokens in Web Services

Security Token verify the authentication and authorization in web services in order to identity the user and his services provider, as well as allows access to the last. Represented in the SOAP message by the tag <wsse:SecurityToken>, this specification provides differets types of tokens, such as UsernameToken, SAML, Kerberos and X.509[2], [9]. In this research is using UsernameToken. Its basic syntax is detailed in Fig. 6.

Username Token

```
1:  <soapenv:Envelope xmlns:soapenv="…" …>
2:   <soapenv:Header>
3:    <wsse:Security SOAP:role="…">
4:     <wsse:UsernameToken wsu:Id="…">
5:      <wsse:Username>Alice</wsse:Username>
6:  <Password Type="PasswordText">Pass</Password>
7:     </wsse:UsernameToken>
8:    </wsse:Security>
9:   </soapenv:Header>
10:  <soapenv:Body>
11:   …
12:  </soapenv:Body>
13: </soapenv:Envelope>
```

Fig. 6. Request of SOAP message with the UsernameToken.

In Fig. 6, one inserts the tag <wsse:Security> to use a security specification in the header of the SOAP message. The web service can use more than one specification in the SOAP message, just only add one tag <wsse:Security>, so as to insert more security specifications (Security Token, XML Encryption, XML Signature, among others). Within this tag one uses the tag <role> that specifies the privileges for a specific user. The tag <role> cannot be repeated or omitted because it would allow any user to modify the SOAP message. The web service recipient is informed –in lines 5 and 6– of the user credentials through of the tags Username and Password. If the user credentials are valid, the SOAP message is processed on the server-side, otherwise the web service returns a message, usually with the HTTP 500 code describing an error. In Fig. 7 we describe the elements that the UsernameToken uses to provide the user's identity.

Elements of the tag <UsernameToken>

/Username: User associated with token.
/Password: User password associated with token.
/Password/@Type: Type of password provided, two predefined types:
- **PasswordText**: password in plain text.
- **PasswordDigest**: Implicit password in has velue with the cryptosystem SHA-1 in base64-encoded and UTF8-enconded.

/Nonce: Random string for each SOAP message.
/Created: Date and time of creation of token.

Fig. 7. Elements of the Tag <UsernameToken> [9].

C. Security Challenges and Related Works

Web services have evolved into a more inclusive technology in order to integrate applications and data exchange in SOA. However, this technology presents a lot of new security risks [2]. This subsection focuses in reviewing the security challenges and related works in web services Security.

In [2], the author defines the main challenges related to standards and interoperability in web services. This research emphasizes the relative immaturity of this technology in regard to security threats, Quality of Service (QoS) and scalability. In [14], the authors classify the security challenges in web services, involving threats, attacks and security problems. These are classified in:

- Service-level threats: attacks against WSDL and UDDI, malicious code injection, phishing, Denial of Service (DoS), XML spoofing schema and session hijacking.
- Message-level threats: fault injection attack, message forwarding, message validation attacks, interception and message confidentiality loss.

Since our approach to analyze message-level threats, such as Injection Attack, we review several works in the literature suggesting the use of the fault injection technique. In [5], this technique was applied to test a security protocol used in the communication between mobile devices in the Internet. Also, in [15] the authors use perturbations in the SOAP messages to emulate attacks, similarly to our proposal but without using a methodology.

In this way, in Table I presents a summary of the main features of the researches related to fault tolerance techniques and vulnerability detection. Also we conducted a comparison with our approach using the following aspects:

1) What is the fault tolerance techniques uses in this research?
2) Did the researchers use a security testing methodology?
3) Did the researchers analyze the response messages?
4) Did the research test web services?
5) Did the research test WS-Security or other specification?

TABLE I. STATE OF THE ART IN WEB SERVICES SECURITY.

Tools-Reference	1.	2.	3.	4.	5.
WebScarab [16]	fault injection		✓	✓	
WS-TAXI [17]	penetration testing			✓	
CDLChecker [18]	fault injection	✓	✓	only WS-CDL[4]	
VS.WS [19]	penetration testing	✓	✓	✓	
IBM Rational [19]	penetration testing	✓	✓	✓	
WSInject	fault injection	✓	✓	✓	✓

In Table I, the state of the art of web service security testing are oriented towards the use of fault injection and penetration testing techniques. Most researches used a security testing methodology and analyzed the response messages. Only this research tested the robustness of WS-Security (UsernameToken) against fault injection attacks.

IV. SECURITY TESTING METHODOLOGY FOR WEB SERVICES

The main challenge in finding vulnerabilites is to determine which attack scenarios are appropriate for testing during the implementation phase of web service. These scenarios can be obtained from various sources, such as the Internet, text books and research papers (c.f. sec. II-A). However, it is hard to find and set up a database with relevant attacks and automating them according to the testing environment. Our purpose in this section is to use a variation of the security testing methodology[5] in order to test web services from a set of XML Injection attacks.

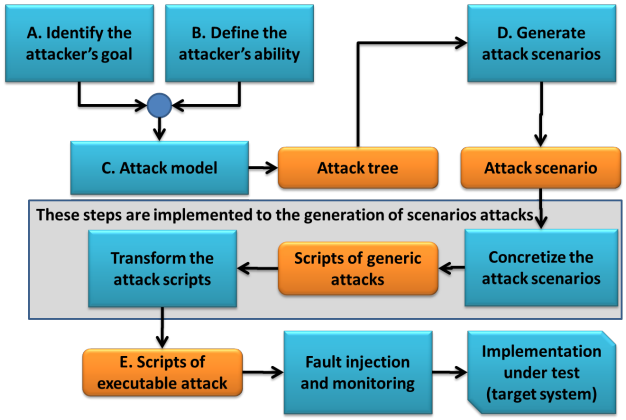

Fig. 8. Security testing methodology and their phases[20].

Since the scripts of executable attack are obtained from implementing the attack scenario, some phases were partially implemented (see Fig. 8). The phases and results of this methodology are described in the following subsections. The reader who wishes to know more about this methodology should look at [20].

A. Identify the Attacker's Goals

The web services attacker aims to find vulnerabilities using different kinds of methods (e.g. Denial-of-Services, spoofing, injection attack, man-in-the-middle, among others). In this research, the goal of the attacker is focused on finding vulnerabilities in servers that work with web services.

The attacker intercepts SOAP messages between the client and the server. His goal is perform unauthorized operations (violation of integrity) or escalate privileges (violation of control access). For that, he tries to inject various XML tags in order to modify the XML message structure and generate faults in the server.

B. Define the Attacker's Ability

Based on the Dolev-Yao model [21], the attacker has the following capabilities:

- Partial control of the network and knowledge of the endpoints (client and server).
- Ability to intercept SOAP messages and modify expressions, delay and duplicate.
- Knowledge of the status of all participants, i.e. the attacker is able to intercept messages and phishing client/server or perform man-in-the-middle attacks.
- Ability to recognize the access points, operations and parameters of WSDL[5].

[5]The Web Services Description Language (WSDL) is a XML-based interface definition language that is used for describing the functionality offered by a web service[2].

C. Build the Attack Model with Attack Trees

The attacks modeling are used to represent the steps of an attack. These models can use methods based on Petri nets to attack tree [20].

The attack tree emphasizes the point of view of an attacker. This model analyzes all possible attacks to a system in an organized way, taking the worst scenarios in order to: i) prevent vulnerabilities during the development; ii) find vulnerabilities in the implementation phase; and iii) evaluate the system with known vulnerabilities and attacks [20].

The attack tree can be represented by a data structure to facilitate the security analysis. This analysis represents the steps of an attack and their interdependencies. It can be used to represent and calculate probabilities, risks, costs, among other variables in order to verify if an attack is feasible.

To represent the web services attack tree in a data structure, first selected a set of known attacks. Next, classified them by the target of the attack, e.g. integrity and availability. To finish, analyzed the requirements that an attack must satisfy to be successful:

i. Analyzed the attacker capability (knowledge of the attack);
ii. Evaluated the possibility of emulating the attack with some testing tool, e.g. WSInject fault injector;
iii. The most of attack need that the web services must contain at least one parameter described in the WSDL to insert the selected attack;
iv. Assessed whether UsernameToken is able to protect against this attack, e.g. XML Injection attack.

These four requirements were used to classify some attacks with boolean values, namely <Possible (P), Impossible (I)>. The result is a set of selected attacks that fulfill the 4 requirements described in Fig. 9, i.e. XML Injection, Cross-site Scripting (XSS) and XPath Injection. In the other side, WS-Security does not protect web services against to Denial-of-Service attack, such as Oversize Payload, Oversize Parsing and Oversize Cryptography.

```
OR 1. Objective: Attacks against Web Services
    OR  1.1  Attacks against integrity
        OR   1.1.1 XML Injection <P, P, P, P>
             1.1.2 Cross-site Scripting (XSS) <P, P, P, P>
             1.1.3 XPath Injection <P, P, P , P>
    OR  1.2  Attacks against availability
        OR   1.2.1 Oversize Payload <P, P, P, I>
             1.2.2 Coercive Parsing <P, P, P, I>
             1.2.3 Oversize Cryptography <P, P, P, I>
```

Fig. 9. Web services attack tree.

Each of the four "P's" represent one requirement, e.g. for XML Injection we have the knowledge [3] to reproduce this attack using WSInject fault injector, the web services contain at least one parameter described in the WSDL and WS-Security need others specifications such XML-Enc or XML-Sig to protect web services against this attack, but there are concerns about UsernameToken that will be analyzed in the following sections.

D. Generate Attack Scenarios

At this stage, the attack scenarios are produced automatically according to the criteria defined in previous phases. The scenarios can be used to create an useful and reusable library of attacks to test protocols and software [20]. For example, a XML Injection attack scenario is described in Fig. 10 using the attack pattern information obtained in [3].

```
Objective: Finding vulnerabilities in Web Services
using XML Injection attack.
Preconditions:
    ❖ The client sends a request to the Web Service
      through SOAP message;
    ❖ The client does not use a secure communication
      scheme such as XML-Sig or XML-Enc;
    ❖ The WSDL describes at least one parameter to
      access the web service.
Attack:
AND 1. If the SOAP message is a request AND
    2. It contain the string searched;
    3. THEN modify/eliminate/duplicate the tags or
       parameters in the string searched AND
    4. Send the modified SOAP message to the web
       service;
    5. In case the response is received;
    6. Then search for vulnerabilities in the response.
```

Fig. 10. XML Injection attack pattern.

E. Implement Attack Scenarios

The attack scenarios –generated in the previous phase– are described in a text notation, i.e. at the same level of the attack tree abstraction. This type of description is useful for verification by analysts and security experts due to their easy configuration, but not to be processed by an injection tool. In this stage, the analysts must perform a set of refinement steps in order to transform the text notation into a proper executable script for the WSInject fault injector, as shown in Fig. 11.

```
Rule 1:
ON event:       env(A,B,String,<EP=SOAP,<Po_A>,<Po_B>)
IF condition:   (1. isRequest() == True) AND
                (2. contains(String) == True)
DO action:      3. stringCorrupt(String, String_Corrupt)
                4. GenerateNewMessage(message)
```

Fig. 11. Executable attack script to emulate XML Injection with WSInject.

V. Proposed Approach

The UDDI Business Registry (UBR) is a single registry for web services through its WSDL. The WSDL file allows us to know how the service can be called, what parameters it expects, and what data structures it returns [2].

Through the UBR Seekda, we select 10 web services from a set of 22,272 services. 5 of which use UsernameToken and the others do not. Each web services has at least one parameter to submit requests. In this section applies the security testing methodology described in Sec. IV for to 10 web services.

A. Fault injection with WSInject

The WSInject fault injector allows to emulate injection attacks like XML Injection, Cross-site Scripting (XSS), XPath Injection, among others. This tool works as a proxy between the client and the web services tested. The interception and

modification of SOAP message exchange are transparent between the client and servers. This way, WSInject does not need the source code of web service or to interfere with the execution platform, allowing it to be used by developers and users. Is enough configure the client to the WSDL of the web service through the proxy. In this research, the fault injector intercepts request messages sent by the client (soapUI [16]) before being passed to the server, as illustrated in Fig. 12.

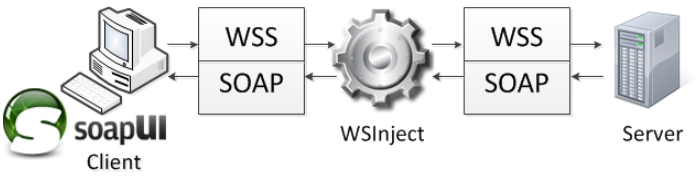

Fig. 12. Architecture used with WSInject and soapUI to test web services.

WSInject uses scripts in plain text format. The user should recognize the operations described in WSDL and use the tool to intercept SOAP message in order to corrupt their parameter values. Examples of generated scripts are shown in Table II.

TABLE II. SCRIPTS TO EMULATE XML INJECTION ATTACK WITH THE WSINJECT FAULT INJECTOR

WSInjects scripts for XML Injection
isRequest(): stringCorrupt("</ser:Username>", "</ser:Username><ser:Username2>hacker</ser:Username2>");
isRequest(): stringCorrupt("</ser:Password>", "</ser:Password> <ser:Password2>admin</ser:Password2>");
isRequest(): stringCorrupt("</ser:Identifier>", "</ser:Identifier> <ser:Identifier>xignite</ser:Identifier>");
isRequest(): stringCorrupt("</ser:Tracer>", "</ser:Tracer> <ser:Tracer>123</ser:Tracer>");
isRequest(): stringCorrupt("<ser:Username>EdgarFilings", "<ser:Username><a>EdgarFilings</ser:Username>");

These scripts use the condition isRequest() to differentiate requests from responses. In the first request, WSInject uses the stringCorrupt action to replace the "</ser:Username>" by "<ser:Username><ser:Username2>hacker<ser:Username2>" to incorporate parameters in the requests, also adding tags in the SOAP message to perform operations not allowed and cause undesirable effects in the web service.

B. Analysis of Vulnerabilities in Web Services

An important aspect of this phase is to identify when a vulnerability was effectively detected, excluding or at least reducing the potential false positives and negatives. It is also necessary to differentiate when a result is invalid due to an internal server failure (unintentional) or is a consequence of a successful attack.

There are several ways to check for vulnerabilities in SOA, e.g. by comparing server responses in the presence/absence of attacks (i.e. gray-box testing), looking for sensitive information exposure, XML schema modification requests, among others. Our approach uses the HTTP status-code in the response message in order to analyzes the behavior of web services under stressful environment, i.e. XML Injection attack.

For example, when the request message is processed by server without detecting the attack, i.e. the web services does not generated an error or identify a possible vulnerability (HTTP status-code 200 OK), then it is a successful attack. Or if the web services returned the HTTP status-code 400 Bad Request, then is considered a robust response because the request could not be understood by the web services due to malformed syntax.

In case of code 500 Internal Server Error, the web services response using a <soap:Fault> tag inside the SOAP message's body, which provides errors and status information of the message containing the following sub-elements:

- <faultcode> Fault code identification
- <faultstring> Descriptive explanation of the fault
- <faultactor> Information about what or who caused the fault to happen
- <details> Information that describes the server error

Based on this analysis, we developed 8 rules to determine the existence of vulnerabilities in web services for Injection Attacks, described below.

Rule 1. If the header contains the code "200 OK" AND the server ran the SOAP message with any attack, THEN there is a Vulnerability Found (VF) in the web service. OTHERWISE, if the SOAP message describes the existence of a syntax error or warning about the presence of any attack, THEN there is No Vulnerability Found (NVF) in the web service.

Rule 2. If the header contains the code "400 Bad request message", e.g. request format is invalid: missing required soap: Body element, THEN there is No Vulnerability Found (NVF) in the web service.

Rule 3. If the header contains the code "500 Internal Server Error" AND there was information disclosure in the SOAP message (e.g. it shows path directory, function library and object information, access to database and XML files with usernames and passwords, among others), THEN there is a Vulnerability Found (VF), OTHERWISE there is No Vulnerability Found (NVF) in the web service.

Rule 4. i) If in the absence of any attack, the header contains the code "500 Internal Server Error" AND there was information disclosure in the SOAP message; AND ii) if in the presence of any attack, the header contains the code "HTTP 200 OK", THEN there is a Vulnerability Found (VF) in the web service.

Rule 5. i) If in the absence of any attack, the header contains the code "500 Internal Server Error" AND there was information disclosure in the SOAP message; AND ii) if in the presence of any attack, the header contains the code "400 Bad request message", THEN there is a Vulnerability Found (VF) in the web service.

Rule 6. i) If in the absence of any attack, the header contains the code "500 Internal Server Error" AND there was information disclosure in the SOAP message; AND ii) if in the presence of any attack, the header contains the code "500 Internal Server Error" too, THEN there is a Vulnerability Found (VF) in the web service.

Rule 7. If the server does not respond, it is considered as crash, THEN the result is considered Inconclusive, because one cannot guarantee that the error was caused by the attack.

Rule 8. If none of the rules above may be applied, THEN the result is considered Inconclusive, because there is no way to confirm if there really were vulnerabilities in the web service.

Rules 4, 5 and 6 analyze the response of web services, which in the absence of the XML Injection attack presents the code "500 Internal Server Error" in the header. In Rule 7, the XML Injection attack analyzes the the web services unavailability (crash), similar to a Denial-of-Service attack (DoS). In this case, the response is classified as inconclusive, because it is not possible to conclude whether the error was caused by the unavailability of the service or by the attack. Rule 8 is an exception, because in case no other rule is matched, the result is inconclusive.

C. Faultload Campaign with WSInject

An important aspect in web services testing is the generation of network traffic (the workload). To generate this workload, we use the Load Testing add-on soapUI. This tool represents the client, as shown in Fig. 12. The generated traffic consists of requests made to web services in order to emulate a real client making requests. It represents the requests that activate the target web service.

The faultload campaign had the following procedure. For each web service were developed 5 XML Injection scripts, each one specifying a corruption of the value of a particular parameter, as shown in Table II. The workload consisted of sending 100 requests per script. In total, 5,000 XML Injection attacks were carried out. Fig. 13 illustrates this campaign.

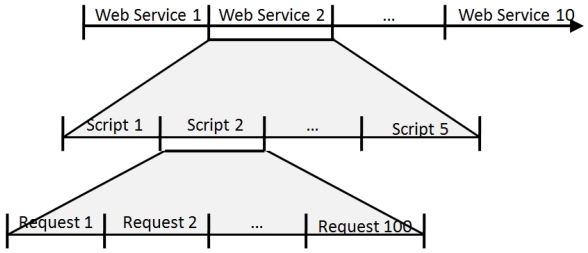

Fig. 13. Faultload campaign.

D. Evaluation of Fault Injection Results

Given a black-box proposed approach, we collected the requests and responses obtained during the XML Injection attacks in the 10 web services. These information are stored in the WSInject fault injector. For example, the Fig. 14 describe the SOAP message (request) modified by the XML Injection scripts and its response. The script modify the "</ser:Username>" paramater in lines 7 and 8 of the request message to "<ser:Username><ser:Username2>hacker<ser:Username2>".

In the first line of the response message (Fig. 14), the web services did not detect the attack and ran the malicious script. In this case, the server returned the HTTP status-code 200 OK. Also the response message showed information about the class used in the software. Thus, we can use Rule 1 of the vulnerability analysis (c.f. Sec. V-B) and conclude that the web service and its server are not robust against XML Injection attack.

```
XML Injection script:
isRequest():stringCorrupt("</ser:Username>",
"</ser:Username><ser:Username2>hacker</ser:
Username2>");
```

Request

```
1:  <soapenv:Envelope xmlns:...">
2:     <soapenv:Header>
3:        <ser:Header>
4:           <!--Optional:-->
5:  <ser:Username>admin</ser:Username>
6:  !--Optional:-->
7:  <ser:Username>admin</ser:Username>
8:  <ser:Username2>hacker</ser:Username2>
9:  ...
10: </soapenv:Body>
11: </soapenv:Envelope>
```

Response

```
1:  HTTP/1.1 200 OK
2:  <!-WS NOT DETECTED THE ATTACK>
3:  Content-Type: text/html; charset=utf-8
4:  Content-Length: 31046
5:  <soap:Envelope ...>
6:  <soap:Body>
7:  <!-- BEGIN: navigation -->
8:  <div class="navigation">
9:  <div class="xMenu" id="ctl00_ctl00_uc
10: Navigation_mnMainNavigation">
11: …
12: </soap:Body>
13: </soap:Envelope>
```

Fig. 14. Log generated by WSInject.

Based on this information, we applied these rules each request and response stored. This procedure also allows to detected vulnerabilities in web services with UsernameTokens. The results of the injection attacks are described in Table III.

The application of the security testing methodology using WSInject fault injector detected that 82% of requests present some vulnerability to XML Injection attack. Our analysis detected many types of vulnerabilities such as servers showed information about its path directory, access to database, function library, among others.

TABLE III. RESULTS OF XML INJECTION ATTACKS USING WSINJECT

Web services	*Total attacks*	*Vulnerabilities Found*	*No Vulnerability Found*
without WSS	2,500	2,300	200
% injected	100%	92%	8%
with WSS	2,500	1.800	700
% injected	100%	72%	28%
Total	5,000	4,100	900
% injected	100%	82%	18%

The use of UsernameTokens reduces the percentage of web services vulnerabilities from 92% to 72%, but not enough (Fig. 15). This happens because these services use password to authenticate the user. However, these techniques do not protect the integrity of SOAP message and its confidentiality. XML-Enc or XML-Sig may be used to provide confidentiality and integrity of both the UsernameToken and the entire message body.

Furthermore, the use of this specification forces the programmers to create more robust web services. i.e. use message timestamps, nonces, and caching, the password should be

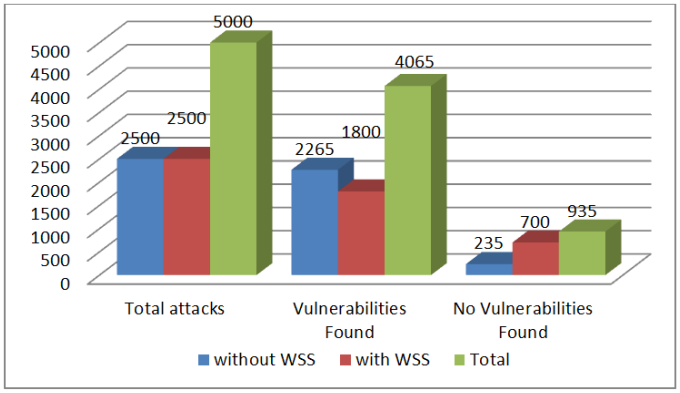

Fig. 15. Faultload campaign.

digested for protect against eavesdropping attacks, as well as other application-specific tracking mechanisms[9].

VI. Conclusions and Future Works

One advantage of the security testing methodology is that it relies on the use of the WSInject fault injector as a general purpose tool, which can be used to emulate several types of attacks (i.e. XML Injection, XPath Injection, Cross-site Scripting (XSS), Fuzzing Scan, Malformed XML, Oversize Payload, Coercive Parsing and Oversize Cryptography) and may generate variants of the same.

The variation of the security testing methodology oriented to communication protocols allow us to applicated it to tested web services. From a set of phases, we obtained executables scripts for the WSInject fault injector.

The attack tree model allowed –in an structured way– to determine which are the 4 requirements (c.f. Sec. IV-C) to apply XML Injection attack.

Also was created a set of rules for vulnerabilities analysis in web services for Injection Attacks. The purpose of these rules are automated the search for vulnerabilities, improve the detection of vulnerabilities and understand the behavior of web services in a non-robust environment.

The result show that WS-Security reduces the number of vulnerabilities (92% to 72%). However, this can be improved with the use of other specifications such as XML-Encryption or XML-Signature.

This procedure can be used to evaluate new technologies and analyze their robustness against various attacks.

As future work, we plan to use variants of attacks to improve detection of new vulnerabilities, always considering the service as a black-box.

References

[1] Cachin, C., and J. Camenisch, *Malicious and Accidental-Fault Tolerance in Internet Applications: Reference Model and Use Cases*, LAAS, MAFTIA, 2000.

[2] Holgersson, J., and E. Soderstrom, *Web Service Security-Vulnerabilities and Threats within the Context of WS-Security*. SIIT 2005, ITU.

[3] Williams J., and D. Wichers, *OWASP Top 10*, OWASP Foundation, 2010, URL: https://www.owasp.org/.

[4] Meiko J., G. Nils, H. Ralph, *A Survey of Attacks on Web Services*, Computer Science - Research and Development, 01 Nov 2009. Springer Berlin, Heidelberg; ISSN: 1865-2034, volume 24, Edição 4. 2009.

[5] Martins E., A. Morais, and A. Cavalli, *Generating Attack Scenarios for the Validation of Security Protocol Implementations*, In Proceedings of the II Brazilian Workshop on Systematic and Automated Software Testing, SBC, Campinas-SP, Brasil, 2008.

[6] Weber TS, *Tolerancia a falhas: conceitos e exemplos*, Intech Brasil, São Paulo, Volume 52, 2003.

[7] Hsueh MC, TK Tsai, and RK Iyer, *Fault Injection Techniques and Tools*, IEEE Computer Society Press, Computer; Volumen 30, Ed. 4, Apr 1997.

[8] Carreira JV, D. Costa, and JG Silva, *Fault Injection Spot-Checks Computer System Dependability*, Spectrum, IEEE, Volume 36, Edição 8, Aug 1999.

[9] Lawrence, K., C, Kaler, A. Nadalin, R. Monzillo, and P. Hallam-Baker, *Web Services Security: UsernameToken profile 1.1*, OASIS, 2006.

[10] Eastlake, D., et al, *XML Signature Syntax and Processing*, 2nd Edition, 2008.

[11] Eastlake, D., et al, *XML Encryption Syntax and Processing*, W3C Recommendation, 2002.

[12] Della-Libera, G., et al, *Security in a Web Services World A Proposed Architecture and Roadmap*, IBM Corp, Microsoft Corp, 7, Apr 2002, URL: http://msdn.microsoft.com/en-us/library/ms977312.aspx.

[13] Lawrence, K., C. Kaler, A, Nadalin, R. Monzillo, and P. Hallam-Baker, *Web Services Security: SOAP Message Security 1.1 (WS-Security 2006)*, OASIS, 2006.

[14] Ladan MI, *Web services: Security Challenges*, in Proceedings of the World Congress on Internet Security, 2011, WorldCIS'11, IEEE Press, Londres, Reino Unido, 21-23, Fev 2011.

[15] Valenti AW, and E. Martins, *Testes de Robustez em Web Services por Meio de Injeção de Falhas*, Thesis (Master in Computer Science), Institute of Computing, UNICAMP, State University of Campinas, Brazil, 29, Jun 2011.

[16] Rogan D., *OWASP WebScarabLite [software]*, Version 20070504-1631, Open Web Application Security Project 2011, URL: http://www.owasp.org/software/webscarab.html.

[17] Bartolini C., A. Bertolino, E. Marchetti, and A. Polini, *WS-TAXI: A WSDL-based Testing Tool for Web Services*, In Proceedings of the International Conference on Software Testing Verification and Validation, 2009, ICST '09, IEEE Computer Society, Denver, Colorado, 1-4 April, 2009.

[18] Zhou L, J. Ping, H. Xiao, Z. Wang, GeguangPu, and Z. Ding, *Automatically Testing Web Services Choreography with Assertions*, In Proceedings of the 12th international Conference on Formal Engineering Methods and Software Engineering. ICFEM'10. Springer-Verlag, Berlin, Heidelberg, 2010.

[19] Vieira M., N. Antunes, and H. Madeira, *Using Web Security Scanners to Detect Vulnerabilities in Web Services*, In Proceedings of the IEEE/IFIP International Conference on Dependable Systems & Networks, DSN '09, IEEE Computer Society, Lisbon, Porgugal, 2009.

[20] Morais A, and E. Martins, *Injeção de Ataques Baseados em Modelo para Teste de Protocolos de Segurança*, Thesis (Master in Computer Science), Institute of Computing, UNICAMP, State University of Campinas, Brazil, 15, May 2009.

[21] Dolev D., A. Yao, *On the Security of Public Key Protocols*, In IEEE Transactions on Information Theory, IEEE Computer Society Press, Mar 1983.