- **Services level threats:** attacks against WSDL and UDDI, injection of malicious code, phishing, denial of service, spoofing XML schemas and kidnapping/ stealing session.
- **Message level threats:** injection attacks, forwarding messages, attacks of message validation, interception and loss of message confidentiality.

2.2 Vulnerabilities Detection Techniques

Following the best practices of software testing and standards, there have been developed a lot of tools, languages and techniques in order to analyze and detect vulnerabilities in systems [2]. The security validation for Web Services can be performed in two phases, static and dynamic phase.

The static phase tries to find faults inserted during the development phase introduced in the code by possible human errors in the project stage. This phase is analyzed as a state not reachable, i.e. it can always be found new faults. In this case, the methods used are Static Analyze (code inspection, static vulnerability analysis) or Theorem Proof, which do not need to run the system. These methods are early detection and carry many benefits such as reduced cost of testing.

On the other hand, the dynamic phase focuses on verification of the system during its running, i.e. the code of the system is tested with real entries to verify security mechanisms at runtime. The Security Testing is applied in this phase. This test looks for vulnerabilities in web applications by sending attack within request message. Among these security techniques, the Penetration Testing and Fault Injection are among them.

Penetration Testing emulate attacks, in order to reveal vulnerabilities. The tests are automated by the use of tools called vulnerability scanner (VS). There are a variety of vulnerabilities scanners, both commercial (e.g. HP Web Inspect, IBM Rational AppScan) and open source (e.g. WSDigger and WebScarab). The vulnerabilities detected differ from one tool to another. An evaluation of several commercial versions of vulnerabilities scanners showed that these tools are primarily limited to low coverage of existing vulnerabilities and the high percentage of false positives.

2.2.1 Fault Injection attack

In software testing, fault injection is a technique for improving the coverage of a test by introducing faults to test code paths, in particular error handling code paths that might otherwise rarely be followed. It is often used with stress testing and is widely considered to be an important part of developing robust software. Robustness testing (also known as Syntax Testing, Fuzzing or Fuzz testing) is a type of fault injection commonly used to test for vulnerabilities in communication interfaces such as protocols, command line parameters, or APIs.

The propagation of a fault through to an observable failure follows a well defined cycle. When executed, a fault may cause an error, which is an invalid state within a system boundary. An error may cause further errors within the system boundary, therefore each new error acts as a fault, or it may propagate to the system boundary and be observable. When error states are observed at the system boundary they are termed failures. This mechanism is termed the fault-error-failure cycle and is a key mechanism in dependability.

2.2.2 XML Injection Attack

In XML Injection, an attacker tries to inject various XML Tags in the SOAP message aiming at modifying the XML structure. Usually a successful XML injection results in the execution of a restricted operation. Depending on the executed operation various security objectives might get violated. Typical examples are:

- modification of payment data - violated security objective : Integrity
- unauthorized admin login - violated security objective : Access Control

3. Testing Methodology

3.1 Security Testing Methodology for Web Services

One of the challenges to find vulnerabilities in Web Services during the implementation phase is determine which attacks scenarios are appropriate to test for [1].

These scenarios can be obtained from various sources such as Internet, books and papers. 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, in part, the Security Testing Methodology [9].

In the following sub-sections, author briefly describe the results of each phase of Security Testing Methodology for Vulnerabilities Detection of XSS in Web Services the implementation of the Security Testing Methodology with XSS. This attack is emulated with WSInject and soapUI [1].

3.1.1 Identification of the Attacker Objectives

The Web Service 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.

3.1.2 Definition of the Attacker Capability

Based on the Dolev-Yao model [6], 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 WSDL5.

3.1.2 Attacks Modeling

In this step, author use the SecurITree version 3.4 in order to model XSS attack. This tool, used in several researches helped us to design the attack tree for injecting vulnerabilities in Web Services.

The attack tree was built and structured accordingly to the proposed steps in [1], composed of the following attributes: i) attacker capability; ii) possibility of emulating the attack by a fault injection tool; iii) the requirements of the attack to be run in the Web Service; and iv) the verification if the WS-Security protects the Web Services from XSS attack.

These four attributes were used to classify the Injection Attacks with boolean values, namely < Possible; Impossible >. The output is the creation of the attack tree, which is used by the attacker to look for vulnerabilities in the Web Services.

3.1.4 Attack Scenarios Generation

At this stage, the attack scenarios are produced automatically according to the criteria defined in step 3.1.3. The output of this step is the attack scenarios described in the same format of the tree leaves, each one representing the description of an attack.

The scenarios can be used to create a useful and reusable library of attacks to test protocols.

3.1.5 Attack Scenarios Implementation

The attack scenarios, generated in step 3.1.4, are described in text notation, i.e. at the same level of the attack tree abstraction. This type of description is useful for testing 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 executable script by WSInject tool.

4. Proposed System

It is necessary to have flexible testing approaches that allow creating multiple scenarios, which could help to determine robustness of the web services. There are several ways to analyze the existence of vulnerabilities in SOA (Service Oriented Architecture), e.g. compare server responses in the presence of attacks and absence of them, sensitive information exposure, XML schema modification request, among others. This step is crucial to reduce the number of false positives or false negatives.

4.1 Proposed System Architecture

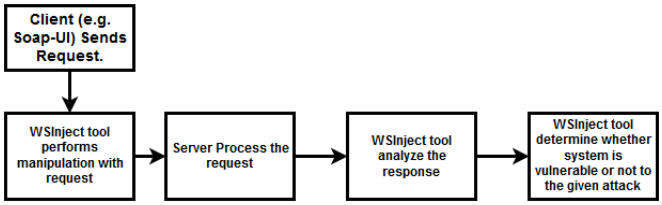

Figure 1: System architecture

Surveyed approach uses the HTTP status-code in the server response, which describes the behavior of the Web Service in a not robust environment. For example, when the request is processed by Web Services without detecting the attack, i.e. not generated a message describing the existence of error in the request, it allows to identify the existence of a possible vulnerability found with code 200 OK. If a code 400 Bad Request is received, it can be considered as robust response because the server detected the XSS attack.

4.2 Proposed System Algorithm

Step 1: Capture the message sent from client (e.g. Soap-UI).
Step 2: Identify the request and modify the content of the request to corrupt it, for given attacking scenario. (Using tool such as WS-Inject).
Step 3: Obtain response from server, observe the impact due to malformed request sent in step 2. This could be analyzed with help of HTTP status code, as mentioned in section 4.2.1.
Step 4: If
I. Given attack does not show any vulnerability in targeted environment then modify the test case,
Go to Step 2;
Else
I. Report the bug in the web service with given attack scenario.
Step 5: Stop.

4.2.1 Proposed System Algorithm rules:

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.

The fault injection can be performed with help of WSInject which is configured between client and server. This can be configured to WSDL of the service through proxy. Using this one can corrupt the request sent through the client (e.g. SoapUI) and observe its response [1]. This enables to conclude whether targeted system has vulnerabilities in it or not. The difference in behavior can be observed by varying workload and faultload. The evaluation can be performed with help of 8 rules developed as mentioned below [1]:

Rule 1. If the message header contains the code “200 OK” AND the server ran the SOAP message with any attack, THEN it could be 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).

Rule 2. If the message 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).

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

Rule 4. i) If in the absence of any attack, the message header contains the code "500 Internal Server Error" AND there are 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).

Rule 5. i) If in the absence of any attack, the header contains the code "500 Internal Server Error" AND there are 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).

Rule 6. i) If in the absence of any attack, the message header contains the code "500 Internal Server Error" AND there are 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).

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.

After performing various attacks made on selected web service, with security tokens (i.e. UsernameToken) and without security token results observed as, the use of UsernameTokens reduces the percentage of Web Services vulnerabilities from 92% to 72%, but not enough. 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 digested for protect against eavesdropping attacks, as well as other application-specific tracking mechanisms.

4.3 Mathematical Model

Let S be the whole system S= I, P, O I-Input P-Procedure O-Output.

Input I=REQ, MEC, S-Token Where,
REQ- Request that could be obtained from client (e.g. soap-UI),

MEC - Request which is obtained in the form of string could be corrupted with certain mechanism suitable to attack scenario,

S-Token - Security tokens may needs to be supplied with request to improve authentication.

Procedure (P), The REQ needs to be analyzed in the system, which could be corrupted by applying MEC suits to given attack scenario, forward the REQ to the web service and wait for the response. As mentioned in the algorithm steps, analyze the response message header information. It usually contains HTTP status code. Further security can be enhanced with S-Token mechanism add-on to the REQ.

Output (O) – As per the guideline provided in the algorithm, from the response output obtain as system is vulnerable to given attack scenario or not.

4.4 Module description

1) Fault injection: To test the robustness of the service, multiple attack scenarios needs to be performed by injecting the malformed code to the REQ made, through clients.
2) Attack scenario: various test cases can be generated to test the robustness of the service by varying the attack scenarios previously made. There are set of parameters need to pass to the service to perform desired operation, it's value or parameter itself can be modified to approach certain corrupting MEC.
3) Response analysis: The malformed request which has been sent to the service may provide in turn response from it. Its header information (e.g. HTTP status) could help to analyze the response.
4) Test case decision: For the given attack scenario based on the response observed, one can conclude whether system is vulnerable to the given attack or not.

5. Conclusion

This paper provides survey on new approach to analyze the robustness of Web Services by Fault Injection with WSInject. This tool allows emulation and generation of attacks, however, the process is delayed and often not automated. This survey focused on emulation of Cross-site Scripting (XSS) attack. This is a fairly frequent attack, according to the research cited, whose effects can be quite devastating for servers and users of Web Services.

The results of the Penetration Testing phase help to develop the rules for vulnerabilities analysis. However, the results obtained by soapUI show a large percentage of false positives and false negatives.

One advantage of the proposed approach is that it relies on the use of a fault injector of general purpose, which can be used to emulate several types of attacks and may generate variants of the same, which is usually limited in the tools commonly used for security testing, as the vulnerabilities scanners.

References

[1] Marcelo Invert Palma Salas, Paulo Lício de Geus, Eliane Martinsl, "Security Testing Methodology for Evaluation of Web Services Robustness - Case: XML Injection", Institute of Computing, UNICAMP, Campinas, Brazil, 2015.
[2] Cachin, C., and J. Camenisch, "Malicious and Accidental-Fault Tolerance in Internet Applications: Reference Model and Use Cases," LAAS, MAFTIA, 2000.
[3] Holgersson, J., and E. Soderstrom, "Web Service Security-Vulnerabilities and Threats within the Context of WS-Security", SIIT 2005, ITU.
[4] Williams J., and D. Wichers, OWASP Top 10, OWASP Foundation, 2010,URL: https://www.owasp.org/
[5] 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
[6] Dolev D., A. Yao, "On the Security of Public Key Protocols," In IEEE Transactions on Information Theory, IEEE Computer Society Press, Mar 1983.
[7] 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.
[8] 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.
[9] 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.

Author Profile

Prof. Vina M. Lomte is the HOD of Computer Dept. at RMD SSOE College, Pune, having more than 10+ years of experience in the field of teaching and research. The domains of her research are Software Testing, Software Engineering and Web Security.

Mr. Jaydeep Mangle is pursuing his Masters of Engineering in the Computer Science Department, RMD SSOE College, Savitribai Phule University, Pune. He has received Bachelor of Engineering degree in Computer Science from University Of Mumbai, Ratnagiri, India.