Detection and Mitigation of Web Application Vulnerabilities Based on Security Testing*

Taeseung Lee[1], Giyoun Won[2], Seongje Cho[2], Namje Park[3], and Dongho Won[1,**]

[1] College of Information and Communication Engineering, Sungkyunkwan University, 300 Cheoncheon-dong, Jangan-gu, Suwon-si, Gyeonggi-do, 440-746, Korea
tslee@kisa.or.kr, {tslee,dhwon}@security.re.kr
[2] Department of Computer Science & Engineering, Dankook University, Korea
kgyoun4@gmail.com, sjcho@dankook.ac.kr
[3] Department of Computer Education, Teachers College, Jeju National University, Jeju, Korea
namjepark@jejunu.ac.kr

Abstract. The paper proposes a security testing technique to detect known vulnerabilities of web applications using both static and dynamic analysis. We also present a process to improve the security of web applications by mitigating many of the vulnerabilities revealed in the testing phase, and address a new method for detecting unknown vulnerabilities by applying dynamic black-box testing based on a fuzzing technique. The fuzzing technique includes a structured fuzzing strategy that considers the input data format as well as misuse case generation to enhance the detection rate compared to general fuzzing techniques.

Keywords: web application, security testing, vulnerability, security.

1 Introduction

Software security testing analyzes the security of applications from the viewpoint of attackers and is not really concerned with the functionality of the software. Such testing consists of static testing and dynamic testing. The advantage of code-based static testing is its ability to efficiently analyze software in its entirety, but this testing often has a high false detection ratio, and can hardly be applied to commercial software whose source code is not given. Execution-based dynamic testing can be applied to commercial software and has a low false detection ratio, but analysis time is long and code coverage is limited [1].

* This research was supported by the KCC(Korea Communications Commission), Korea, under the R&D program supervised by the KCA(Korea Communications Agency) (KCA-2012-12-912-06-003).

** Corresponding author.

J.J. Park et al. (Eds.): NPC 2012, LNCS 7513, pp. 138–144, 2012.

This paper proposes a vulnerability detection and mitigation technique to increase the security of web applications. The vulnerability detection technique used in this paper is a software security test that combines both static and dynamic analysis from the viewpoint of the attacker, not the developer; and the vulnerability mitigation technique used here is the secure coding method. To this end, automated static analysis tools and dynamic analysis tools are applied to the web application to detect known vulnerabilities. Then a "fuzzing" technique for detecting new vulnerabilities is proposed. "Fuzzing" is a technique that generates/mutates input data either randomly or structurally and injects it into the application then monitors the results of application execution to detect vulnerabilities [2]. This paper proposes an abuse case generation and testing strategy for efficient fuzzing as well. Third, to mitigate or remove detected web application vulnerabilities, a process for applying the secure coding technique is shown.

2 Security Testing Integration and Vulnerability Mitigation

2.1 Integration of Static and Dynamic Testing

Static analysis and dynamic analysis are complementary. The advantages of integrating them are as follows:

- Expansion of analysis scope and analysis targets: The code coverage of the code review of static analysis is high, while the code coverage of dynamic analysis is low. On the contrary, source codes are given in case of code review, but dynamic analysis can be applied to commercial software as well.
- Increased accuracy: Static analysis has a high false detection ratio, while dynamic analysis is accurate. So if they are integrated, the accuracy of vulnerability detection can be increased.
- Increased detection ratio: It is very difficult for fuzzing to detect vulnerabilities. It can be supplemented by vulnerability scanning or code analysis that has a high detection ratio.
- Expansion of detection areas: Code review and vulnerability scanning can detect known vulnerabilities. If fuzzing is applied, new unknown vulnerabilities can be detected as well.

2.2 Fuzzing

Vulnerabilities of web applications are caused mostly by wrongly implemented source codes, but sometimes by web application servers. This paper proposes the fuzzing process for finding vulnerabilities of web applications and servers.

2.2.1 Input Format Considerations

To create misuse cases used for fuzzing, the input format of web applications must be analyzed. The request message, which is the input of web applications, can be divided into the "URI of the Request Line," and the "message header and message body."

The first part, the URI, is a standard command system indicating the location of documents and resources on the Internet. Its format is "protocol name://domain name/path name (parameter)." It is used as the input for the browser address window or for the Request Line of the request message. If components like the protocol, domain name and path name are invalid, the input cannot reach the web applications. As attacks are made by changing the input for the URI parameter in general, fuzzing can be done for the values of the parameters.

The second part of the request message is the message header and body. The header includes additional information of the HTTP message, and the body describes the data necessary for the request. Fuzzing can also be attempted by altering the parameter values of the header and body of the request message.

2.2.2 Fuzzing Strategy

There are four fuzzing techniques for discovering vulnerabilities: simple random, simple mutation, structured random and structured mutation. For more information on these techniques, please see a previous study of the authors [3].

- Simple random: This method randomly generates input data without taking the input format into consideration. As fuzzing is done with completely random data without a basic input format, and most web applications process errors unawares, no exception will occur.
- Simple mutation: This method brings a valid URL or request message and changes it as much as desired in units of single bytes. If the field perceiving the path of the URL or request message in web applications is mutated, it will not be perceived as a correct input, thereby lowering efficiency.
- Structured random: This method adds the path of the URL or the header of the request message to the randomly generated data. If a header with a certain level of format is applied to the simple random data, web applications will perceive it as a normal format.
- Structured mutation: This method gets actual input data, grasps the data format structure, and mutates desired parts. For example, the data of the request message is mutated, or the header size may be mutated, etc.

2.2.3 Technique for Generating Abuse Cases for Fuzzing

It is important to generate abuse cases when using the fuzzing technique. This study collected and analyzed information on existing known vulnerabilities to generate abuse cases, and generates abuse cases in consideration of the fuzzing technique (See Figure 1) [4].

The information on disclosed vulnerabilities of JEUS Web Application Server (WAS) for generation of abuse cases was searched. If JEUS vulnerabilities are searched through CVE (Common Vulnerabilities and Exposure), vulnerabilities related to the Alternate Data Stream can be found as shown in Figure 2.

The Alternate Data Stream is a function of the Windows NT File System that prevents users from seeing a file through Windows Explorer by loading this file onto another file. This method is also used by attackers to prevent the attack file from

being detected by hiding it in another file. If this is used to make a request like "test.jsp::$DATA," JEUS will perceive it as a general file, not a jsp file, thereby exposing the source [5].

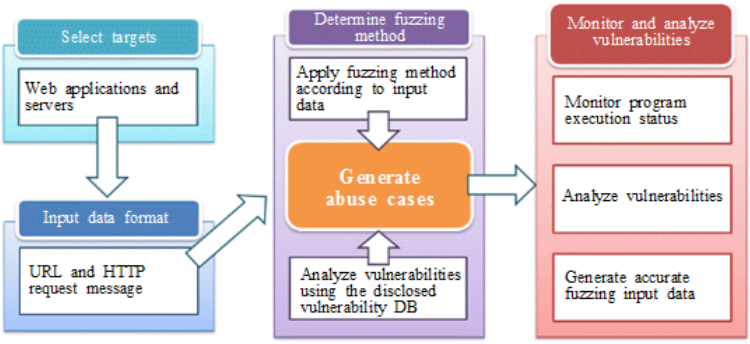


Fig. 1. Abuse case generation

Name	Description
CVE-2008-6528	NTFS TmaxSoft JEUS 5 before Fix 26 allows remote attackers to read the source code for scripts by appending ::$DATA to the URL, which accesses the alternate data stream.

Fig. 2. JEUS vulnerabilities

To collect and analyze information on more vulnerabilities, vulnerabilities regarding the Alternate Data Stream of other products similar to the disclosed vulnerabilities of JEUS were investigated. <Table 1> shows that web servers are the main targets of attacks, and as for types of abuses, most of the input data is divided into filename and data stream.

Table 1. Results of an Alternate Data Stream vulnerability search

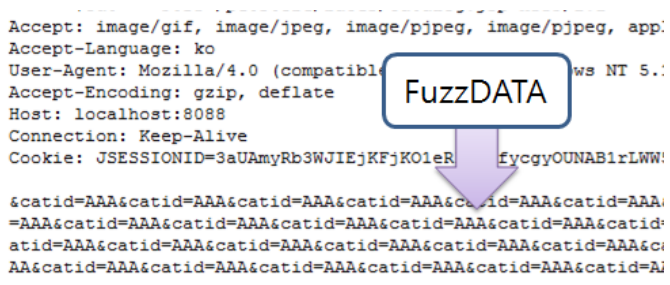
CVE-ID	Types of abuses	Targets of attack
CVE-2209-2445	test.jsp::$DATA	Sun ONE Web Server
CVE-2008-6528	test.jsp::$DATA	JEUS
CVE-2006-5715	HTTP GET request::$DATA	EFS Easy Address Book Web Server
CVE-2006-5714	HTTP GET request::$DATA	EFS Easy File Sharing Web Server
CVE-2006-1475	filename:stream syntax	Windows XP Firewall

Other vulnerabilities similar to data stream vulnerabilities that were also analyzed are shown in Table 2. The analysis result showed that they were the buffer overflow of the HTTP request message due to the input value of Post: or Host:. Accordingly, the association between the input value of each field in the HTTP request message and the occurrence of vulnerabilities may be considered, and abuse cases can be generated in consideration of this.

Table 2. Results of searching vulnerabilities related to the HTTP request message

CVE-ID	Types of abuses	Targets of attack
CVE-2008-4678	Long HTTP Host Header	WebSphere
CVE-2008-3257	POST /.jsp	Oracle WebLogic Server

Table 3. Abuse case generation method

Applied to	Abuse case
URL filename URL data stream HTTP request message field	*http://...../bugTrack/**FuzzDATA**.jsp* *http://...../bugTrack/Default.jsp**FuzzDATA*** 

As for the method of generating abuse cases, the types of abuses in <Table 1> were analyzed, and the structured mutation method was applied to the filename of the URL and the data stream, which are normal input data, as shown in <Table 3>. At this time, the types of abuses in <Table 2> were analyzed, and test cases were generated for each field of the header and body of a normal HTTP request message.

2.2.4 Monitoring and Vulnerability Analysis

Fuzzing can discover abnormal terminations or unique bugs only, and it is impossible to accurately analyze vulnerabilities with this information alone. Accordingly, the next step is to monitor the input process of the fuzz data, and analyze exceptions to classify vulnerabilities.

As for web applications, the response message of the client can be used to check if there was any exception. As for web servers, process monitoring tools like Process Explorer can be used to look at the status of processes. It is possible to use register information, stack information and exception information at the time when vulnerabilities occurred through debugging using OllyDbg.

2.3 Vulnerability Mitigation

For starters, if there are source codes, a code review tool can be used to detect source code implementation errors, and then vulnerability scanning and fuzzing conducted for the complete application. Vulnerability scanning detects known vulnerabilities based on the inspection rules made through analyzing information on existing vulnerabilities. To detect unknown vulnerabilities, fuzzing can be applied.

Mitigation measures should be implemented for detected vulnerabilities. For example, as there is a limit in blocking Cross Site Scripting (XSS) and injection vulnerability of the OWASP TOP 10 (2010) with a web firewall, removing or mitigating it can be a more fundamental solution. To mitigate vulnerabilities detected through security testing, vulnerabilities will be searched in the list of vulnerabilities that can be mitigated through secure coding. Vulnerabilities detected by applying the mitigation technique to the vulnerabilities found in the list can be mitigated.

3 Conclusion

In this paper, static testing and dynamic testing were applied to web applications to detect security vulnerabilities which were then removed. In other words, code review, vulnerability scanning and fuzzing were conducted to detect vulnerabilities, and a process of using secure coding to mitigate vulnerabilities was proposed. Also, this paper proposed a fuzzing technique for discovering new vulnerabilities as well as an abuse case generation technique, which is the key to efficient fuzzing.

References

1. Ernst, M.D.: Static and dynamic analysis: synergy and duality. In: Proc. of WODA 2003 (ICSE Workshop on Dynamic Analysis) (2003)
2. Godefroid, P., Levin, M.Y., Molnar, D.: Automated Whitebox Fuzz Testing. NDSS (2008)
3. Kim, D.J., Cho, S.J.: Fuzzing-based Vulnerability Analysis for Multimedia Players. Journal of KIISE: Computing Practices and Letters 17(2) (2011)
4. Kim, G., Cho, S.: Fuzzing of Web Application Server Using Known Vulnerability Information and Its Verification. Proc. of the KIISE Korea Computer Congress 2011 38(1-B), 181–184 (2011)
5. Security Focus Vulnerability Database: Vulnerability Summary for BID: 32804, Security Focus (2008)
6. Park, N., Kwak, J., Kim, S., Won, D., Kim, H.: WIPI Mobile Platform with Secure Service for Mobile RFID Network Environment. In: Shen, H.T., Li, J., Li, M., Ni, J., Wang, W. (eds.) APWeb Workshops 2006. LNCS, vol. 3842, pp. 741–748. Springer, Heidelberg (2006)
7. Park, N.: Security Scheme for Managing a Large Quantity of Individual Information in RFID Environment. In: Zhu, R., Zhang, Y., Liu, B., Liu, C. (eds.) ICICA 2010. CCIS, vol. 106, pp. 72–79. Springer, Heidelberg (2010)
8. Park, N.: Secure UHF/HF Dual-Band RFID: Strategic Framework Approaches and Application Solutions. In: Jędrzejowicz, P., Nguyen, N.T., Hoang, K. (eds.) ICCCI 2011, Part I. LNCS, vol. 6922, pp. 488–496. Springer, Heidelberg (2011)
9. Park, N.: Implementation of Terminal Middleware Platform for Mobile RFID computing. International Journal of Ad Hoc and Ubiquitous Computing 8(4), 205–219 (2011)
10. Park, N., Kim, Y.: Harmful Adult Multimedia Contents Filtering Method in Mobile RFID Service Environment. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) ICCCI 2010, Part II. LNCS (LNAI), vol. 6422, pp. 193–202. Springer, Heidelberg (2010)

11. Park, N., Song, Y.: AONT Encryption Based Application Data Management in Mobile RFID Environment. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) ICCCI 2010, Part II. LNCS (LNAI), vol. 6422, pp. 142–152. Springer, Heidelberg (2010)
12. Park, N., Song, Y.: Secure RFID Application Data Management Using All-Or-Nothing Transform Encryption. In: Pandurangan, G., Anil Kumar, V.S., Ming, G., Liu, Y., Li, Y. (eds.) WASA 2010. LNCS, vol. 6221, pp. 245–252. Springer, Heidelberg (2010)
13. Park, N.: The Implementation of Open Embedded S/W Platform for Secure Mobile RFID Reader. The Journal of Korea Information and Communications Society 35(5), 785–793 (2010)