Detecting Vulnerabilities in Web Applications Using Automated Black Box and Manual Penetration Testing

Nor Fatimah Awang and Azizah Abd Manaf

Advanced Informatics School (UTM AIS),
UTM International Campus,
Kuala Lumpur, Malaysia
norfatimah@upnm.edu.my, azizah07@ic.utm.my

Abstract. Today, web applications are becoming the most popular tool that offers a collection of various services to users. However, previous research and study showed that many web applications are deployed with critical vulnerabilities. Penetration testing is one of the well-known techniques that is frequently used for the detection of security vulnerabilities in web application. This technique can be performed either manually or by using automated tools. However, according to previous study, automated black box tools have detected more vulnerability with high false positive rate. Therefore, this paper proposed a framework which combines both automated black box testing and manual penetration testing to achieve the accuracy in vulnerability detecting in web application.

1 Introduction

Today, Internet has become the most ever powerful tool for user throughout the world. The internet offers a collection of various services and resources, not only email and World Wide Web as the principle constituents of internet. According to the World Internet Usage and Population Statistics as of June 2012, the number of current Internet user has raised to 2.4 billion, which is estimated to be equal to 34.3% of the world population [1]. This number is a 566.4% increase since the year 2000. Due to the simplicity of its use and its high accessibility, the Web has become the dominant way for people to search information, online banking, job seeking, purchasing tickets, hotel reservations and social networking. Nowadays, popular networking sites such as MySpace and Facebook, with exciting and interactive user driven content such as blog and YouTube videos, are becoming the norm for web content. The growth of these sites gave a high impact and business opportunity to the organization. Several recent studies [2], [3] indicate that this popularity of web applications has unfortunately also attracted attention of attackers. This is reflected by various statistics. For example, Fig. 1, taken from the 2012 IBM Internet Security X-Force 2012 Trend and Risk Report, presents web application vulnerabilities versus total vulnerabilities [2]. It shows the increase in the number of web application vulnerability from 2011 to 2012. Web application vulnerabilities increase to 14%

A.I. Awad, A.E. Hassanien, and K. Baba (Eds.): SecNet 2013, CCIS 381, pp. 230–239, 2013.

from 2,921 vulnerabilities in 2011 to 3,551 vulnerabilities in 2012. Similarly, another study conducted by Positive Technology in 2010 and 2011, reported that from 123 websites chosen for study, all the sites contained vulnerabilities whereby 60% of sites are vulnerable to high-risk, 98% medium and 37% are vulnerable to low-risk [3]. All the above studies show the potential serious vulnerabilities which exist in web applications.

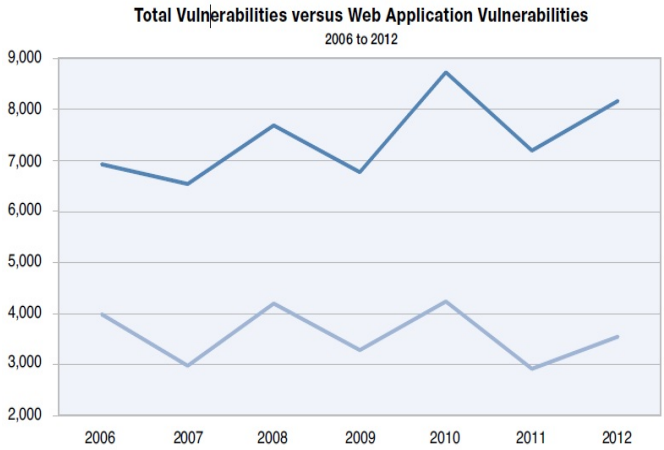

Fig. 1. Total Vulnerabilities versus Web Application [2]

In information security, there are no accepted standard regarding the word vulnerability. We are trying to define the word vulnerability related to web application. There are many definitions proposed, here are some of them:

- According to Wang et al., vulnerability is security flaw, defect or mistake in software that can be directly used by a hacker to gain access to system or network [4].
- Weakness in the security system which might be exploited by malicious users causing loss or harm [5].

Here we can consider the term vulnerability is applied to one weakness in a system, which allows an attacker to violate the integrity of that system. Based on the above scenario, security is a big issue that should be seriously considered by the system administrator as well as top management in order to protect the potential software and web application systems.

The main objective of this paper is to propose a framework for detecting vulnerability in web applications using both techniques automated black box and manual penetration testing.

The structure of this paper is as follows. Section 2 briefly describes background of web application vulnerability and discusses on more related techniques that are commonly used to detect the vulnerabilities which exist in web application. Section 3

describes more detail on the proposed framework. Section 4 discusses the results and finally, section 5 presents the conclusion.

2 Background and Related Work

In order to understand the variety of vulnerabilities in web application, this section will give some explanations on the architecture and processes of web application, web application vulnerability and several techniques that can be used to discover vulnerability.

2.1 Web Application Architecture

Web application architecture is often structured as a three-tiered application [6],[7]. The architecture of web applications consists of web browser, web server, web application and database server. Tier 1 architecture consists of web browser and web server, Tier 2 for web application and Tier 3 for database.

- Tier 1 – web browser and web server architecture

  A web browser is also known as web client, functions as the user interface to web server to get input from web application or database server. Web server receives input and interacts with client through web browser by using Hyper Text Transfer Protocol (HTTP) or via secure protocol HTTPS. There are many type of web servers where Apache and Internet Information Server (IIS) are the most popular web server in the world.

- Tier 2 – web application architecture

  Web application consists of a collection of scripts such as Javascript, VBscript, which reside on a web server and interact with databases or other sources of dynamic content. The common example, the data input using a web browser is processed and stored into database. Java Server Pages (JSP), PHP, Active Server Pages (ASP), Perl and Common Gateway Interface (CGI) are among the technology used to build web based application. Using the infrastructure of the Internet, web applications allow service providers and clients to share and manipulate information in a platform-independent manner. Normally web application server is attached on top of web server and works as interface from web client and database server. Web server will manage the page requested from web client by sending to application server and application server constructs code dynamically and passed back to web server. The flow of data among tiers gives rise to the input validation problem for the web application server; it must check and/or modify incoming input before processing them further or incorporating them into output that it passes to other tiers to execute. Failure to check or sanitize input appropriately can compromise the web application's security [8].

- Tier 3 – database architecture

  Stores and manages all the processed users input data. The database tier is responsible for the access of authenticated users and the rejection of malicious users from the database.

2.2 Web Application Vulnerability

Because web applications are open to the world, they are more vulnerable to attacks and prone to a great variety of vulnerabilities. In this section, we describe some of the most common and well-known web application vulnerabilities based on OWASP Top Ten lists 2010 [9]. OWASP stands for Open Web Applications Security Project, and is an open-source collaboration of web based security tools, technologies and methodologies from industry leaders, educational organizations and individuals from around the world. The OWASP Top Ten is a valuable document for developers and testers because its focus on web applications. The OWASP Top 10 2010 has listed the ten most critical web application security vulnerabilities as shown in Table 1. The OWASP Top 10 2010 refers to the top 10 web attacks as seen over the year by security experts, and community contributors to the project.

Table 1. OWASP Top Ten Vulnerability 2010 [9]

Ranking	Vulnerability
A1	Injection
A2	Cross site scripting (XSS)
A3	Broken Authentication / Session Management
A4	Insecure Direct Object References
A5	Cross Site Request Forgery
A6	Security Misconfiguration Sensitive Data Exposure
A7	Insecure Cryptographic Storage
A8	Failure to Restrict URL Access
A9	Insufficient Transport Layer Protection
A10	Unvalidated Redirects and Forwards

Injection and Cross Site Scripting attacks are the most common popular vulnerabilities exploited by attackers. In this paper, we focus on SQL Injection and Cross Site Scripting due to the common vulnerabilities that have evolved in the last decade [10]. These attacks are triggered where an attacker intentionally sends a malicious input or script to the application to get some valuable information. The detail attacks are as follows [4]:

- SQL Injection Attack

  This kind of attack occurs when an attacker uses some special SQL queries as an input, which can open up a database [11], [12], [13]. Online forms such as login prompts, search enquiries, guest books and feedback forms are always targeted. The simple test to check for SQL injection attack is to append "or+1=1" to the URL and shows the data returned by the server.

- Cross Site Scripting (XSS) Attack

  Many XSS attacks happen because vulnerable applications fail to sanitize malicious input at either server side or browser side, allowing them to be injected into response pages. XSS attacks exploit through inputs that might contain HTML tags and Java Script code [14], [15]. For example, an attacker injects a malicious JavaScript program into a trusted site's web page. If a victim user visits the affected web page, then the web browser executes the malicious JavaScript program as though it came from the trusted site. By using this vulnerability, an attacker can force a client, such as a user web browser, to execute attacker-supplied code. As a result, the attacker's code is granted access critical information that was issued by the trusted site.

2.3 Techniques to Detect Vulnerability

Penetration testing and code analysis are two (2) well-known techniques frequently used by web developers and testers for detecting vulnerability [16],[17]. Penetration testing involves in stressing the application from the point of view of an attacker by using specific malicious inputs. This technique can be performed either manually or by using automated tools. Automated tool is also known as automated black box tools and most of these automated black box tools are commercial tools such as IBM Rational AppScan [19], Acunetix Web Vulnerability Scanner [20] and HP WebInspect [21] whereas, static code analysis is a technique that analyzes the source code of the application without trying to execute it, whilst searching for potential vulnerabilities. Similarly with penetration testing, this technique can be done manually or by using code analysis tools like Pixy [22], FORTIFY [23] or Ounce [24]. According to OWASP [4], the most efficient way of finding vulnerabilities in web application is manual code analysis. This technique is very time consuming. It requires expert skills and is prone to be overlooked error of a system. Therefore, due to time constraints or resource limitations, developers or testers frequently have to choose automated tools to search for vulnerability [18]. However, according to previous study conducted by [17], [18], automated black box tools have detected more vulnerabilities with high false positive rate (vulnerability detected that did not exist in web application). Therefore, this paper presents a framework that combines both automated black box testing and manual penetration testing to achieve better accuracy of vulnerability detecting in web application.

3 The Proposed Framework

Our proposed framework consists of four phases, refer to Fig. 2:

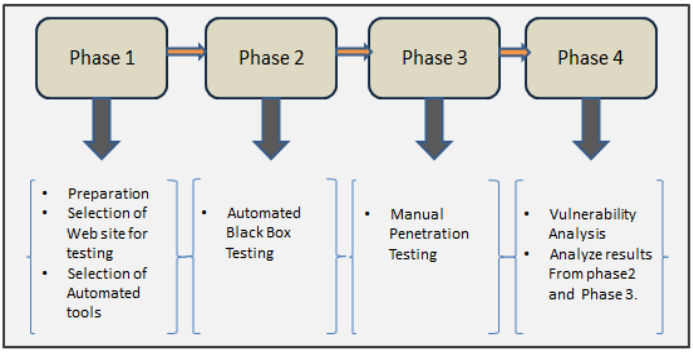

Fig. 2. Framework for detecting vulnerability in web application

- Phase 1

  The Web Application Security Scanner Evaluation Criteria (WASSEC) has set a guideline to evaluate web application scanners on their ability to effectively test web applications and identify vulnerabilities [25]. Other studies for comparing the effectiveness of web application scanning tools conducted by researchers [26, 27], was also used as a guideline for selecting the scanning tools. In this framework, we have considered and selected well known commercial automated black box tools namely IBM Rational AppScan [19] and we have chosen an anonymous online web application website as shown detail in Fig. 3.

Target IP/Hostname	Operating System	Web Server	Web Application
www.	Unix	Apache 2.2.3 Redhat	PHP

Fig. 3. Target Testing Website

- Phase 2

  This phase uses the automated black box tool to scan the services to identify potential vulnerabilities in web application. The tool used in this phase in order to scan and detect the variety of known vulnerabilities in web application as listed in Table 1. To begin a scanning session, the tester must enter the entry URL of the web application. The tester then must specify options for the scanner's page crawler, in order to maximize page scanning coverage. In this phase, we always set the scanner to run in automated mode to maximize vulnerability detection capability.

- Phase 3

  This stage performs manual penetration testing to confirm vulnerabilities that have been detected through second phase (to check false positive of the vulnerabilities).

- Phase 4

  The goal of this phase is to analyze the result of the target system after conducting testing. Vulnerability analysis result will be based on two activity's results from two different phases, phase 2 and phase 3. This activity will conclude and validate the vulnerability whether the vulnerability reported in phase 2 is actual vulnerability and to ensure no false positive exist in the test result.

4 Results and Discussion

This section presents the results that have been performed as described in section 3. The testing was done in actual anonymous online web application as shown detail in Fig. 3. As you can see in Fig. 4, a total of 27 vulnerabilities have been detected by automated black box tools of which the medium and high severity level share the same amount with the remainder categorized as low. Meanwhile, Fig. 5 shows five different types of vulnerabilities namely:

1) SQL Injection : it is possible to alter and steal the information stored in database
2) Content Spoofing: it is possible to trick a user to believe that certain content appearing on a Web site is legitimate and not from an external source.
3) Directory Indexing: it is possible to allow the contents of unintended directory listings to be disclosed to the user
4) Information Leakage: it is possible to reveal sensitive information, such as from developer comments or error messages
5) Abuse of functionality: it is possible to use a web site's own features and functionality to attack itself or others.

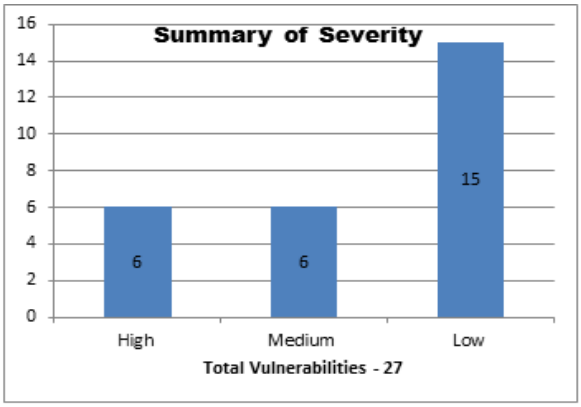

Fig. 4. Number of Vulnerabilities by severity level

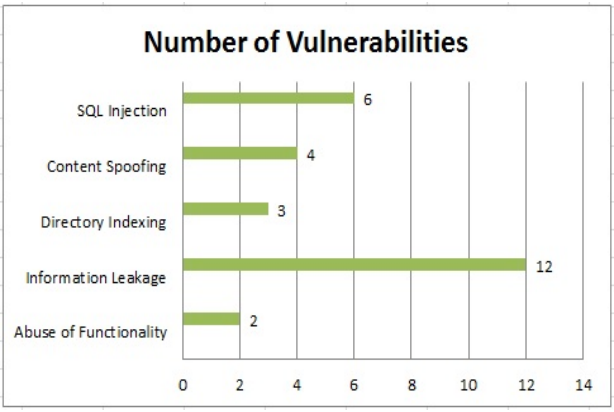

Fig. 5. Number of Vulnerabilities by type

The result shows that SQL Injection is classified as high severity; content spoofing and abuse of functionality are categorized as medium severity. Meanwhile, information leakage and directory indexing are categorized as low severity.

Possible vulnerabilities that have been reported in Fig. 5, is validated again by using manual penetration testing to ensure that false positive does not exist in this testing phase. We consider that vulnerability exists, if any malicious patterns or errors were found as shown detail in Fig. 6. This work is still in progress to validate all vulnerabilities detected by automated tool. In this paper, we have chosen SQL Injection vulnerabilities due to high severity level to validate whether false positive or not. As you can see in Table 2, results so far do not have false positives. It means that the potential vulnerability detected by automated black box tool is considered as actual vulnerability after successfully being reconfirmed by manual penetration testing.

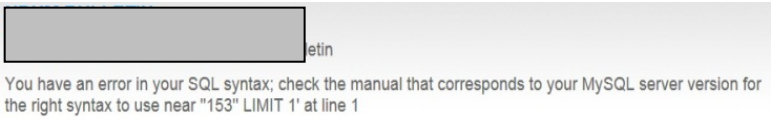

Fig. 6. Example of error after conducting manual testing

Table 2. Reconfirm False Positive

Type of Vulnerability	# of Vulnerability	# of False Positive
SQL Injection	6	0

5 Conclusion

This paper presents a framework for detecting vulnerability in web application. One automated black box tool was selected to detect various vulnerabilities in web application. In this work, the automated tool pointed out five different type vulnerabilities as shown in Fig. 5. After detecting vulnerability process was completed in phase 2, manual penetration testing was performed in order to ensure there are no false positive exist in the test result.

References

1. Internet World Stats, Usage and Population Statistics (2013), http://www.internetworldstats.com/stats.htm
2. X-Force Research and Development Team, IBM X-Force 2012 Trend and Risk Report, Technical Report (March 2012)
3. Web Application Vulnerability Statistics for 2011-2012, Positive Technology, Technical Report (2012)
4. Wang, J.A., Guo, M., Wang, H., Xia, M., Zhou, L.: Environmental metrics for software security based on a vulnerability ontology. In: Third IEEE International Conference on Secure Software Integration and Reliability Improvement, pp. 159–168 (2009)
5. Pfleeger, C.P., Pfleeger, S.L.: Security in Computing, 3rd edn. Prentice Hall PTR (2003)
6. Kim, J.: Injection Attack Detection Using the Removal of SQL Query Attribute Values. In: 2011 International Conference on Information Science and Applications, ICISA, April 26-29, pp. 1–7 (2011)
7. Zhendong, S., Wassermann, G.: The Essence of Command Injection Attacks in Web Applications. In: Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 372–382 (2006)
8. Shklar, L., Rosen, R.: Web Application Architecture: Principles, Protocols and Practices, 2nd edn. John Wiley & Sons (2009)
9. The Open Web Application Security Project: The Ten Most Critical Web Application Security Vulnerabilities, https://www.owasp.org/index.php/Main_Page:OWASP_Top_Ten_Project
10. Theodoor, S., Davide, B., Engin K.: Have things changed now? An Empirical Study on Input Validation Vulnerabilities in Web Applications (2012), http://iseclab.org/papers/theo-journal.pdf
11. Ezumalai, R., Aghila, G.: Combinatorial Approach for Preventing SQL Injection Attacks, Advance Computing Conference. IEEE International, IACC (2009)
12. Justin, C.: SQL Injection Attacks and Defense. Syngress Publishing (2009) ISBN 13: 978-1-59749-424-3
13. Huang, Y., Yu, F., Hang, C., Tsai, C.H., Lee, D.T., Kuo, S.Y.: Securing Web Application Code by Static Analysis and Runtime Protection. In: Proceedings of the 12th International World Wide Web Conference, WWW 2004 (May 2004)
14. Shahriar, H., Zulkernine, M.: Taxonomy and classification of automatic monitoring of program security vulnerability exploitations. Journal of Systems and Software 84, 250–269 (2011) ISSN 0164-1212, 10.1016/j.jss.2010.09.020
15. Avancini, A.: Security testing of web applications: A research plan. In: 2012 34th International Conference on Software Engineering, ICSE, June 2-9, pp. 1491–1494 (2012)

16. Bacudio, A.G., Yuan, X., Chu, B.B., Jones, M.: An Overview of Penetration Testing. International Journal of Network Security & Its Applications (IJNSA) (November 2011)
17. Vieira, M., Antunes, N., Madeira, H.: Using Web Security Scanners to Detect Vulnerabilities in Web Services. In: IEEE/IFIP Intl Conf. on Dependable Systems and Networks, DSN (2009)
18. Nuno, A., Marco, V.: Comparing of Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services. In: 15th IEEE Pacific Rim International Symposium on Dependable Computing (2009)
19. IBM Security Appscan, http://www-01.ibm.com/software/awdtools/appscan/
20. Acunetic, http://www.acunetix.com/
21. HP WebInspect, http://www8.hp.com/my/en/software-solutions/software.html?compURI=1341991
22. Jovanovic, N., Kruegel, C., Kirda, E.: Pixy: a static analysis tool for detecting Web application vulnerabilities. In: 2006 IEEE Symposium on Security and Privacy, May 21-24, p. 6 p. 263 (2006)
23. FORTIFY, http://www.fortifysoftware.com/
24. Ounce, http://www.ouncelabs.com/
25. Web Application Security Scanner Evaluation Criteria Version 1.0, http://projects.webappsec.org/w/page/13246986/Web%20Application%20Security%20Scanner%20Evaluation%20Criteria
26. Doupé, A., Cova, M., Vigna, G.: Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. In: Kreibich, C., Jahnke, M. (eds.) DIMVA 2010. LNCS, vol. 6201, pp. 111–131. Springer, Heidelberg (2010)
27. Fonseca, J., Vieira, M., Madeira, H.: Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. In: The 13th IEEE Pacific Rim International Symposium on Dependable Computing (December 2007)