Open Source Web Application Security: A Static Analysis Approach

Mamdouh Alenezi
College of Computer & Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
malenezi@psu.edu.sa

Yasir Javed
College of Computer & Information Sciences
Prince Sultan University
Riyadh 11586, Saudi Arabia
yjaved@psu.edu.sa

***Abstract*— In this paper, we have tested several open source web applications against common security vulnerabilities. These vulnerabilities spans from unnecessary data member declaration to leaving gaps for SQL injection. The static security vulnerabilities testing was done in three categories (1) Dodgy code vulnerabilities (2) Malicious code vulnerabilities (3) Security code vulnerabilities on seven (7) different web applications built in Java. It is evident from the obtained results that almost all selected applications have similar kind of vulnerabilities that might have been introduced due to hasty programming or lack of developer knowledge against security vulnerabilities. We recommend to create an intelligent development framework that can provide suggestions for secure development by overcoming common vulnerabilities, can add missing code and can learn from expert developer's practices to overcome the security vulnerabilities.**

Keywords*— *Open source; Web application;, Source code; Security; Static analysis

I. INTRODUCTION

The open source software security is a major concern for organizations that implements them as part of their software solutions, predominantly if it will be a major component of their ecosystems. Open source is no inferior or better than proprietary software when it comes to security. Since open source code is open for the public to look at, its security will have been exposed to larger and additional worthwhile scrutiny. The practice of building secure software that functions properly under unwanted attacks is called software security.

Web applications are the most vital communication channels among different kinds of service providers and clients. As their importance increased, the negative impact of their security flaws has grown as well. These web applications accomplish mission-critical jobs and handle sensitive information. Vulnerabilities that might lead to the compromise of sensitive information are being reported continuously. The main reason for this phenomenon [22] is the lack of security awareness on part of the developers.

Security weaknesses found late in the software development cycle are more costly to rectify than the ones found early [24]. Consequently, developers have a duty to attempt to discover weaknesses as early as possible. However, the size and complexity of the code bases and shortage of [1]developers experience may complicate software weaknesses discoveries. Finding vulnerabilities in web applications can be done by code auditing (code inspection or reviews), static Analysis, dynamic analysis, and security testing [1],[4].

The National Vulnerability Database (NVD), in only 2015, recorded 6,488 new vulnerabilities, and the NVD reports a sum of 74,885 software vulnerabilities revealed during 1988-2016. Static analysis tools inspect code for faults which might lead to software security vulnerabilities, and generate warnings of the location of the purported flaw in the source code, the nature of the flaw, and often additional contextual information. The main purpose of static analysis tools is to find coding errors before they can be exploited. Static analysis is predominantly a good fit to security since several security issues happen in places that hard to reach and difficult to exercise by running the code.

Detecting vulnerabilities and finding precarious flaws in code can be classified in two main approaches: white-box analysis and black-box testing [2]. White-box analysis examines the code without the need of executing it. This can be done manually through code inspection and reviews or automatically through security static analysis [2]. Static analysis is an automated process to assess code without executing it. Code review methods, both manual and automated, try to find security issues before releasing the software. Black-box testing analyzes program execution externally. In other words, it compares the software execution outcome with expected results.

Code review needs knowledge of code as practitioners, with slight experience will not do a good job during a code review. The code review should be done by experienced senior developers while equipping them with modern source code analysis tools. There is no silver bullet solution to ensure secure coding. However, code review provides great insights in finding security irregularities. The remainder of the paper is organized as follows: Section 2 discusses the related work, Section 3 discusses the collected data, Section 4 explores the results and discussions, Section 5 explains the suggested framework, and Section 6 concludes the paper.

978-1-5090-5579-1/16/$31.00 ©2016 IEEE

II. Related Work

A collective criticism against static analysis tools the fact that they produce many false positives [6]. Nevertheless, several research results have demonstrated that statis analysis tools generate reliable warnings to some extent. Walden and Doyle [18] showed that Fortify SCA tool warnings are strongly correlated to NVD vulnerabilities. Gegick et al. [19],[20] showed statistically significant correlation between static analysis warnings and vulnerabilities. Zheng et al. [21] showed, based on an industrial large-scale study, that static analysis is an effective technique for checking faults that have the potential to cause security vulnerabilities. We conclude from previous studies that static analysis tool can be used to give some insights about the source code problems. The analysis results should be investigated in order to educate software developers and managers.

Previous research evaluated different techniques and their capabilities in detecting vulnerabilities [3],[6]. Finifter and Wagner [3] compared the effectiveness of black-box testing and manual code review for web applications, they found that they complement each other, and manual analysis found more vulnerabilities, but took much more time. Austin and Williams [6] compared the effectiveness of systematic and exploratory manual penetration testing, static analysis, and automated penetration testing. They reported that no one technique was capable of discovering every type of vulnerability. Their findings showed that very rare vulnerabilities are found by multiple techniques and automated penetration testing was found to be the most effective in terms of hours, followed by static analysis. Clark et al. [10] conducted a vulnerability study focusing on early existence of vulnerabilities in software products where the reused legacy code is a major player of these vulnerabilities.

III. Collected Data

We conducted an empirical study on the source code of seven open source software system, namely, Crawler4j, Elasticsearch, WebGoat, Friki, Gestcv, Jfinal, and Jpetstore. Here is some information about these systems. Find Security Bugs version 1.4.6 was used to find security problems. This plugin was integrated with NetBeans. It is a FindBugs plugin for security audits of Java web applications. It can detect 86 different vulnerability types with over 200 unique signatures with extensive references for each bug patterns with references to OWASP Top 10 and CWE.

Crawler4j [11] is an open source application for web-crawling that can crawl the web in few minutes using multi-threading. It is able to crawl almost 200 Wikipedia pages per second and waiting for 200 milliseconds between each steps. It is also possible to do resume-able crawling.

Elasticsearch [12] is a distributed search engine built for cloud using RESTful web services. It supports multiple indexing and multiple tenant cloud. It has real time search and analytical capabilities. It can allow full text search as well as persistent where each document changes are recorded. It has JSON based document store.

WebGoat [13] is a deliberately designed web application for security testing maintained by OWASP. It is also designed to teach security and penetration testing system and common security flaws. It can train in cross-site scripting, access control, parameter manipulation, blind SQL injection, web services, numeric SQL injection using realistic teaching environment. It is platform independent environment that uses Java VM. When you run the webgoat it is highly probable that your machine may be hacked.

Friki [14] is a wiki like application built using Java and can be deployed on any modern servlet. It has some common features like wiki and its common markup tag support. It offers an easy customizable solution that can be loaded dynamically without the need of restarting the server again.

Gestcv [15] is a java based application used to manage Curriculum Vitae. It allows creation of CV and allows searching of its contents. It is also based on Struts, Spring and Hibernate. It is built on MVC architecture. It uses MySQL database, and allows persistent development.

JFinal [16] is a complete framework written in Java language and it uses RESTful web services. It allows easy development without writing large amount of code for writing RESTful web services. It's built on MVC architecture and require no configurations as uses XML. Java development and deployment doesn't need server to be restarted and is automatically loaded. Plugins can be scaled and provide struts support as well as supports multi-view.

JpetStore [17] is completely re-written web application pet store that was originally made by Microsoft. It is written in Java and overcomes the shortcoming of its original version. It is based on Struts with color coding conventions to ease programmer for writing codes. Presentation later is based on MVC architecture and there is HTML in database making it completely independent.

TABLE I. Selected Projects

Name	Ver.	# Files	LOC
Crawler4j	4.2	43	7114
Elasticsearch	6.0.1	3865	616000
Webgoat	7.0.1	35	8474
Friki	2.1.1	21	1843
Gestcv	1.0.0	119	11524
Jfinal	2	24	2379
JpetStore	6	116	25820

The above table represents the projects selected along with the version that was used for evaluation. It is also shown how many files were evaluated along with the number of lines of code that is been evaluated. Only java files are been tested against code vulnerabilities. Tested vulnerabilities are categorized into following three categories.

- Dodgy code vulnerabilities

- Malicious code vulnerabilities
- Security code vulnerabilities

Dodgy code is a code that is confusing, unclear, irregular, or written in a way that leads to errors. These characteristics make the code less transparent and robust. Examples of some of the most occurring dodgy codes are (1) Loading the values that known to be null (2) public protected fields are defined but not read (3) Fields should be defined as protected or public but not defined, (4) Computation of values and storing it in local variable that are never used.

Malicious code vulnerability is a code that can be altered or exploited by other code. It can be in form of worms, viruses, Trojan horses or other programs that can exploit other security parameters. There are numerous Malicious code vulnerabilities like (1) exposing internal representation to reference object that pose a threat to security if that object is accessed through different purpose, (2) Usually the field that has last results should be declared is final but is missed and poses a threat of being used by malicious code to change the value. (3) Returning the mutable object as reference poses a serious security threat and can be used by malicious code, (4) A field is defined as static but not protected can be accessed by malicious code and can be changed.

Security code gaps means finding errors that might impact the application security by exploiting security vulnerabilities. It can be in form of malicious data injection or manipulating the applications using malicious data. There are couple of security categories that should be checked as these provide open threats to any web application. Most common security threats are (1) Carriage return and line feed or HTTP response splitting is a usual way programmers adapt to work on response returned but if hacker can plunge the response through injections it can be used to control how web functions will act. (2) Use of predictable random generator to calculate the random number may result in finding the predicted number and can be used to find the password sent or any other secret value, (3) Usually a file is opened to read or write where filename is sent as input and can result in revealing the full path of location of file (4) Usually programmer pass JDBC connection string as prepared statement unsafely can result in SQL injection attack, (5) Use of regular expression in a variable unprotected will result in plugging a big regular expression to compile and will result in Denial of service as program will get busy in parsing the variable for large amount of time.

IV. Results and Discussion

Most common dodgy code problem found in the selected packages are explained in following table that is a data is stored in a variable but is not used almost occurring 522 times in Elasticsearch project while a field is declared but contains a null value. These kind of vulnerabilities doesn't pose a threat but decrease the performance of application as well as consuming the memory on unnecessary data stores. Following figure shows the ten most common dodgy code vulnerabilities found in the selected seven projects.

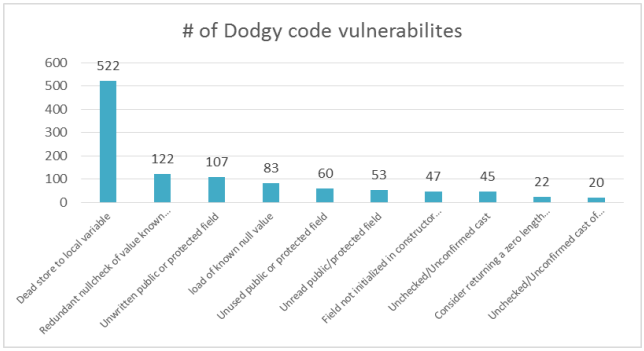

Fig. 1. Number of most common dodgy code vulnerabilities

Malicious code vulnerabilities poses a threat which make them interesting and may result in security violation. Like code containing a mutable object, which may be accessed by untrusted code, and if so will compromise the security or a field declared as static, which can be changed by mutable object or malicious code inserted and thus should be defined as protected. Among the project most common vulnerabilities are that field is used as static but not as protected or final almost 99 times in Elasticsearch project and it is a common problem in all projects.

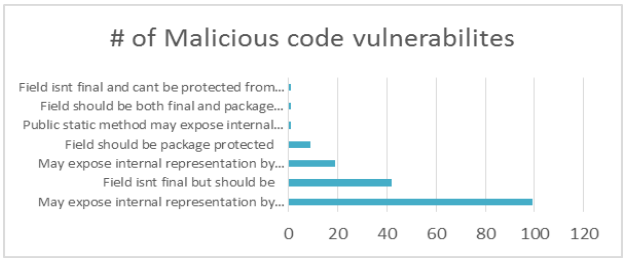

Fig. 2. Number of most common malicious code vulnerabilities

Security code vulnerabilities are common threats, which may result in data loss, application failure or account loss. Most common attack is CRLF injection that means carriage return and line feed, it is used at an end of file sequence or HTTP stream to identify the discrete elements or end of data. If a new CRLF is inserted in-between the original CRLF it will result in malicious code being inserted into to application and will compromise integrity, hijacking client sessions and web browser poisoning. CRLF injection attack is done at Application layer. HTTP request if contain CR and LF characters will be responded by two responses both as HTTP responses. It is possible that second response can be plugged into as attacker plunging the cross site scripting or cache poisoning attacks or cross user defacement. Also if a random number generator that is used and the number patterns can be easily detected, it will result in security exploitation like Cross site request forgery attack or account hack. The most common security attack done is potential CRLF attack possible 17 times in crawler4j application and potential path traversal attack.

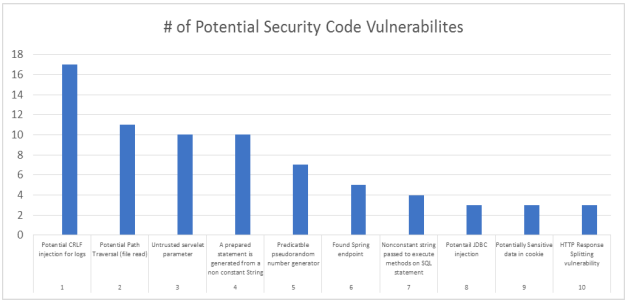

Fig. 3. Potential security code vulnerabilities

The following figures represents the results based on evaluation made on the selected projects. It is observed that most of projects offers hackers a free ride to hack into the systems. It also made possible for novice users with tools to abrupt the smooth execution of programs and a small vulnerability will result in a bigger threat. It doesn't means that smaller number of vulnerabilities pose less amount of threat but it shows that even a small number of vulnerabilities will result in application crash and these problems should be fixed immediately. Most problems of Dodgy code problems occurred in ElasticSearch, while malicious code problems also occurred in Elasticsearch. It is seen that Jfinal has got more security problems making an observation that number of problems in each project faced are independent of number of lines of code.

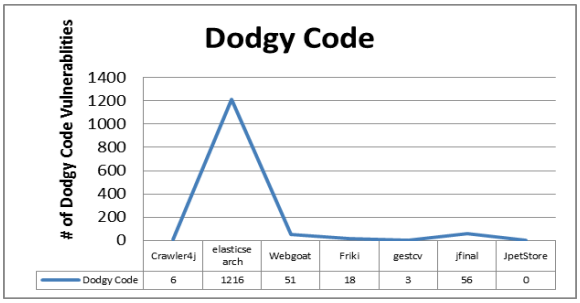

Fig. 1. Dodgy code vulnerabilities in selected projects

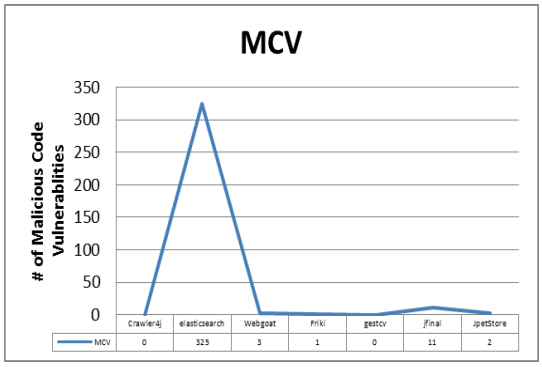

Fig. 2. Malicious code vulnerabilities in selected projects

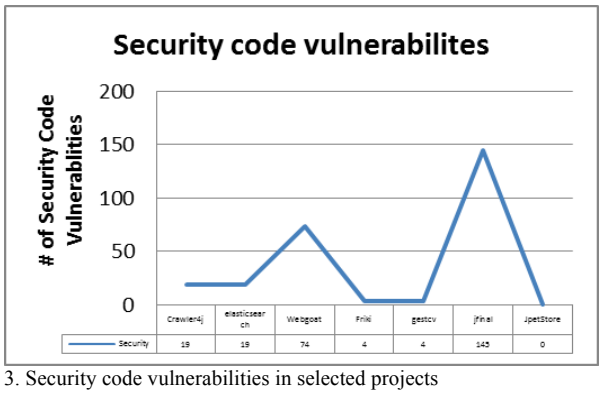

Fig. 3. Security code vulnerabilities in selected projects

V. SUGGESTED FRAMEWORK

Several organizations for example MITRE [7], SANS Institute [8] and OWASP [9] have highlighted the significance of educating students, developers, managers about security issues. These organizations do their part by frequently publishing common programming errors. Our study supports the intuition that web developers usually fail in securing their web applications. The outdated approach of testing applications after they are finished proved to be problematic. We believe that educating developers and giving them hints while they are developing the application will result in more secure applications. Developers and test mangers don't have to wait until they finish to find out if there is a security issue or not in the code. Learning from previous security errors can be a great aid in preventing them from happening in the future.

Software security researchers have relied in finding vulnerabilities on both databases of reported vulnerabilities such as the National Vulnerability Database (NVD) and static analysis results. In our suggested framework we make use of both approaches. In Figure 7, we explain our suggested framework. The framework can be integrated with any integrated development environment (IDE). The idea is to enable developers and testers to find security problems in the code while the system is in implementation [23]. After a piece of code has been written, the framework will run that code on several static analysis tools, check the code in two available databases, Common Weakness Enumeration (CWE) and National Vulnerability Database (NVD), and eventually give a recommendation based on the collected data from three different sources. This will give an instant feedback to the developer about the written code. It will make him/her confident about his code. It will also educate him/her in the go since these recommendations will help him/her learn a lot about code security problems.

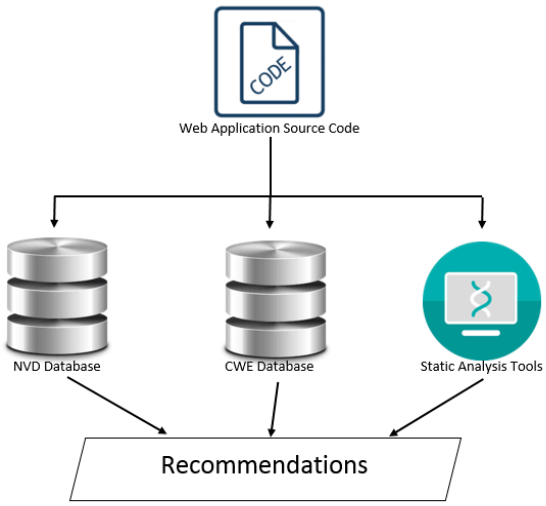

Fig. 3. Suggested framework for secure code design

VI. Conclusion

It is observed that selected projects have common vulnerabilities in all terms that are dodgy code, malicious code or security code. These common vulnerabilities includes from a field declared but unused to SQL injection attack, CSRF attack, cross site scripting attack and web cache poisoning. These all kind of vulnerabilities are mostly inserted due to developer's non awareness or bad programming practice. The vulnerabilities from selected projects also reveals that most of the open source code have security and malicious code vulnerabilities making it more prone to attacks, as open source projects are mostly used by organization trying to avoid costs but it also give attackers to look into to the code vulnerabilities enabling them to do sophistical and bulk attacks. Thus two suggestions are being made (1) before selection of any open source web application the analyst should look into types of web application, its size and attacks that can be done. It should be then selected with noting the developer expertise if the errors can be removed then it should be removed else not selecting the application before running into a bigger problem. (2) Creation of secure development framework that shouldn't allow developers to do hasty programming and adding the necessary security code to avoid any attacks. It should also suggest developers about good programming practice and possible script insertion to avoid potential threats. The framework should also allow the provision to select old projects and remove its vulnerabilities in all terms. The framework should be an intelligently updatable to allow inclusion of new threats that may arise.

References

[1] Livshits, V. Benjamin, and Monica S. Lam. "Finding Security Vulnerabilities in Java Applications with Static Analysis." Usenix Security. Vol. 2013. 2005.

[2] Antunes, Nuno, and Marco Vieira. "Defending against web application vulnerabilities." Computer 45.2 (2012): 0066-72.

[3] Finifter, Matthew, and David Wagner. "Exploring the relationship between Web application development tools and security." USENIX conference on Web application development. 2011.

[4] Lee, Taeseung, et al. "Detection and Mitigation of Web Application Vulnerabilities Based on Security Testing." Network and Parallel Computing. Springer Berlin Heidelberg, 2012. 138-144.

[5] Tripp, Omer, et al. "Andromeda: Accurate and scalable security analysis of web applications." Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2013. 210-225.

[6] Austin, Andrew, and Laurie Williams. "One technique is not enough: A comparison of vulnerability discovery techniques." Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on. IEEE, 2011.

[7] Martin, Bob, et al. "2011 CWE/SANS top 25 most dangerous software errors." Common Weakness Enumeration 7515 (2011).

[8] Dhamankar, Rohit, et al. "The top cyber security risks." TippingPoint, Qualys, the Internet Storm Center and the SANS Institute faculty, Tech. Rep(2009).

[9] OWASP, Top. "10: Ten Most Critical Web Application Security Risks." (2013).

[10] Clark, Sandy, et al. "Familiarity breeds contempt: The honeymoon effect and the role of legacy code in zero-day vulnerabilities." Proceedings of the 26th annual computer security applications conference. ACM, 2010.

[11] Ganjisaffar, Y. "Crawler4j–Open Source Web Crawler for Java." (2012).

[12] Gormley, C., & Tong, Z. (2015). Elasticsearch: The Definitive Guide. " O'Reilly Media, Inc.".

[13] https://www.owasp.org/index.php/WebGoat_Installation accessed in 5th May 2016..

[14] https://sourceforge.net/projects/friki/ accessed in 5th May 2016.

[15] http://gestcv.sourceforge.net/ accessed on 5th May 2016.

[16] http://www.jfinal.com/ accessed on 5th May 2016.

[17] https://sourceforge.net/projects/ibatisjpetstore/ accessed on 5th May 2016.

[18] J. Walden and M. Doyle, "SAVI: Static-analysis vulnerability indicator," IEEE Security & Privacy, vol. 10, no. 3, pp. 32–39, 2012.

[19] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Prioritizing software security fortification throughcode-level metrics," in ACM Workshop on Quality of Protection (QoP), 2008.

[20] M. Gegick, P. Rotella, and L. Williams, "Predicting attack-prone components," in International Conference on Software Testing Verification and Validation (ICST), 2009.

[21] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, "On the value of static analysis for fault detection in software," IEEE Transactions on Software Engineering, vol. 32, no. 4, pp. 240–253, 2006.

[22] Jovanovic, N., Kruegel, C., & Kirda, E. (2006, May). Pixy: A static analysis tool for detecting web application vulnerabilities. In IEEE Symposium on Security and Privacy, 2006 (pp. 6-pp).

[23] M. Alenezi and Y. Javed, "Developer Companion: A Framework to Produce Secure Web Applications," in International Journal of Computer Science and Information Security, vol 14, no. 7, pp. 12-16, 2016.

[24] Imran, Asif, Shadi Aljawarneh, and Kazi Sakib. "Web Data Amalgamation for Security Engineering: Digital Forensic Investigation of Open Source Cloud." Journal of Universal Computer Science 22.4 (2016): 494-520.