Developer Companion: A Framework to Produce Secure Web Applications

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*—Software engineering and development is a very complex endeavor that contends with limited resources, potentially causing software to behave in an unexpected manner. Software developers often lack secure coding skills and its a major reason behind development of insecure web applications. In this work, we propose a developer companion as an integrated framework that can be integrated to any IDE to educate and help developers produce more secure code. This framework can be adopted and can be made more intelligent by focusing on historical security flaws in the development team. expert developers practices to overcome the security vulnerabilities.**

***Keywords*—web applications, source code, security, static analysis**

I. Introduction

Software development and engineering is a very complex endeavor that contends with limited resources [1], [2], potentially causing software to behave in an unexpected manner. Software developers often lack secure coding skills and its a major reason behind development of insecure web applications [3]. The most worrisome class of these faults can be exploited by attackers. These faults are considered security vulnerabilities [4] that are recurrent, causing companies to struggle to allocate resources for their management [5]. The practice of building secure software that functions properly under unwanted attacks is called software security.

Web applications nowadays have moved from static information about companies to a complete communication channel by providing numerous services on which clients can connect [6]. Introduction of web services and data over the web has increased the number of attacks on them thus a small security flaw in web application will have a bigger negative impact. Services that are offered by web application may range from normal purchase information to mission critical tasks or extremely sensitive information.

Development cost in terms of money and time increase whenever changes are required to be done in software at later stages [7]. Thus it is important to identify major vulnerabilities and fix them at earlier stage of development. 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 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 [8], [9].

From 1988 to 2016 total vulnerabilities recorded by National Vulnerability Database (NVD) are 72,855 out of which 6,488 are just reported in 2015. Probable security vulnerabilities can be detected using static analysis tools. These tools also provide details about each flaw like which line has flaw, type of flaw, what possible vulnerability can cause etc. Based on several studies, it was expected that 90 percent of security incidents are results of exploiting defects in the design or code of commonly used software [10]. 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. While many tools and research proposed recently have attempted to address several security exploits, we argue that a critical aspect to avoid these security problems is to target developers by educating and assisting them with developing secure code.

One of the responsibilities of software developers is to determine non-functional requirements, such as security and performance. Educating developers on security in order to help them build elasticity in applications to protect them against attacks and to prioritize security threats and handle them before designing applications. Constructing secure software needs a great deal of security education. Many software developers are not aware and equipped with enough security education. In addition, many programming books do not teach how to write secure programs [11].

Detecting vulnerabilities and finding precarious flaws in code can be classified in two main approaches: white-box analysis and black-box testing [12]. 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 [12]. 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 II discusses the related work, Section III discusses the collected data and the empirical study, Section IV explains the suggested framework, and Section V concludes the paper.

II. Related Work

Static analysis tools usually tend to produce false positive reports and it is one of the major critiques against these tools [13]. However, the vulnerabilities found by these tools are found to be reliable as reported by Walden and Doyle [14] as they reported that vulnerabilities reported by Fortify SCA tools are highly correlated to NVD vulnerabilities. Furthermore, Gegick et al. [15], [16] showed the correlation of actual vulnerabilities and warnings found by static analysis tools are highly correlated. A large scale study conducted by Zheng et al. [17] at industry showed the importance and effectiveness of using static analysis to find flaws that can lead to 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 [18], [13]. Manual code reviews and black box testing can be both used to find the vulnerabilities where manual code reviews can find more vulnerabilities but it will take a lot of time. Both techniques complement each other as explained by Finifter and Wagner [18]. Austin and Williams [13] found that there is no single technique that can detect all vulnerabilities. After exploring the techniques like manual and automated penetration testing and static analysis, they found that automated penetration testing are better than other two techniques in terms of vulnerabilities detected. Clark et al. [19] focused on effect of legacy code on vulnerabilities and found out that it is major player in terms of vulnerabilities found in software systems.

III. Empirical Study

To evaluate the current status of the security of several web applications, we conducted an empirical study on the source code of seven open source web software systems from different domains, namely, Crawler4j, Elasticsearch, WebGoat, Friki, Gestcv, Jfinal, and Jpetstore. We provide some information about these systems. Table I summarizes the collected 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 [1] 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 [2] 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 [3] 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 [4] 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 [5] 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 [6] 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. Its built on MVC architecture and require no configurations as uses XML. Java development and deployment doesnt need server to be restarted and is automatically loaded. Plugins can be scaled and provide struts support as well as supports multi-view.

JpetStore [7] 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. Summary of the Systems

Project	**Version**	**No. of 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	14	2379
JpetStore	6	116	25820

We report the results of running FindBugs on these applications. We report two types of bugs, namely Malicious

[1]https://github.com/yasserg/crawler4j

[2]https://github.com/elastic/elasticsearch

[3]https://github.com/WebGoat/WebGoat

[4]https://sourceforge.net/projects/friki/files/friki/

[5]https://sourceforge.net/projects/gestcv/

[6]http://www.jfinal.com/

[7]https://sourceforge.net/projects/ibatisjpetstore/

Code Vulnerability (MCV) and Security code. 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.

Figures 1 and 2 show the number of security issues found in these applications. These results clearly show that these web application have a lot security problems. These security issues can be addressed at early stages of these applications development. 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 dont 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. While many tools and research proposed recently have attempted to address several security exploits, we argue that a critical aspect to avoid these security problems is to target developers by educating and assisting them with developing secure code.

IV. Suggested Framework

Several organizations for example MITRE [20], SANS Institute [21] and OWASP [22] 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 dont 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.

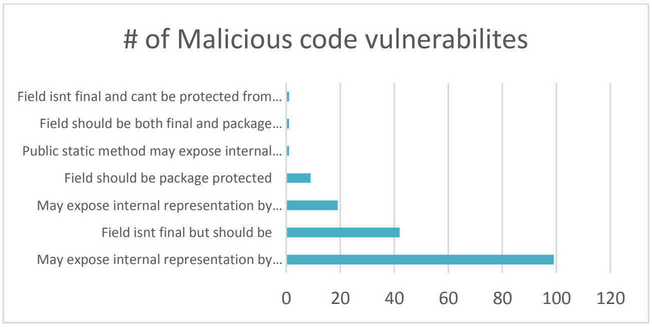

Fig. 1. MCV found in selected web applications.

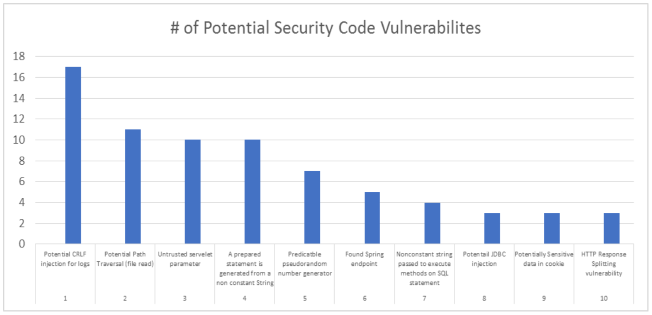

Fig. 2. Security gaps found in selected web applications.

Software security researchers have measured vulnerabilities using 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. We utilize the valuable knowledge in vulnerability repositories such as Common Weakness Enumeration (CWE) and National Vulnerability Database (NVD), which is contributed by software security experts around the world, and is available for public use for free. In Figure 3, 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. 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, CWE and 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.

For the static analysis tools, in the framework, several

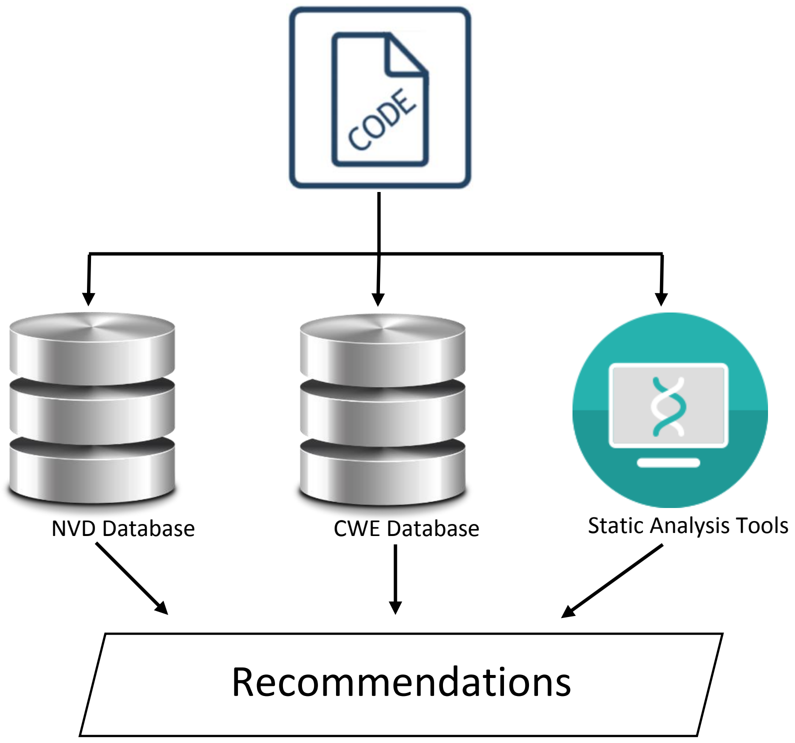

Fig. 3. The Suggested Framework.

tools can be integrated, namely FindBugs[8], PMD[9], Yasca[10], and LAPSE[11]. The Static Code Analysis Module contains static analysis tool(s) which scan the given code repository to find vulnerabilities. These tools are open source and can be integrated with several available IDEs. In the suggested framework, a filtered summary of these tools can be presented as recommendations for developers to educate them and to help them avoid such problems in their source code. The proposed framework is based on the static analysis of code written by software developers. Static analysis tools report suspicious security vulnerabilities found in source code. These results can be utilized for recommending issues and best practices to the software developers who contributed to writing those class and components, hence improving their software security skills.

Numerous kinds of security vulnerabilities can take place in code, design, or architecture. The security community uses Common Weakness Enumerations (CWE) to differentiate security vulnerabilities. For the Common Weakness Enumeration (CWE), the list is available in multiple machine-readable formats including html, XML, and PDF. The CWE Schema is also provided for processing the complete CWE List. The framework can check the developer code against these common problems in source code again to educate them and to help them avoid such problems in their source code. CWE is a software community project that targets creating and maintain a catalog of software weaknesses and vulnerabilities. The goal of the project is to identify, fix, and prevent those vulnerabilities.

NVD complete database about vulnerabilities can be downloaded from their website in unified XML format. Each XML document contains the security vulnerabilities found with a description about them. Our designed framework can check the developer code against these common problems in source code again to educate them and to help them avoid such problems in their source code.

These three source of recommendations will be aggregated and shown to the developers. Later on, some intelligence can be integrated to the framework as well. Based on the historical errors and flaws, the framework will adopt and focus on these specific security issues.

V. Conclusion

It is observed that selected projects have common vulnerabilities in all types of security flaws, malicious code and security code. These vulnerabilities are mostly inserted due to developers' non awareness or bad programming practice. The vulnerabilities from selected projects also reveals that they have security and malicious code vulnerabilities making them more prone to attacks. The suggested framework will allow developers to produce more secure code and help them learn about best practices in developing web applications. The framework will also tackle these issues at early stages before even going to testing which will reduce the cost of software development.

[8]http://find-sec-bugs.github.io/
[9]http://pmd.github.io/
[10]http://scovetta.github.io/yasca/
[11]https://code.google.com/archive/p/lapse-plus/

References

[1] M. Alenezi and F. Khellah, “Evolution impact on architecture stability in open-source projects,” *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 5, no. 4, pp. 24–35, 2015.

[2] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[3] I. Abunadi and M. Alenezi, “An empirical investigation of security vulnerabilities within web applications,” *Journal of Universal Computer Science*, vol. 22, no. 4, pp. 537–551, 2016.

[4] M. Bishop, *Introduction to computer security*. Addison-Wesley Boston, MA, 2005.

[5] M. Nyanchama, “Enterprise vulnerability management and its role in information security management.” *Information Systems Security*, vol. 14, no. 3, pp. 29–56, 2005.

[6] M. Alenezi and I. Abunadi, “Evaluating software metrics as predictors of software vulnerabilities,” *International Journal of Security and Its Applications*, vol. 9, no. 10, pp. 231–240, 2015.

[7] M. Alenezi and K. Magel, “Empirical evaluation of a new coupling metric: Combining structural and semantic coupling,” *International Journal of Computers and Applications*, vol. 36, no. 1, pp. 34–44, 2014.

[8] V. B. Livshits and M. S. Lam, “Finding security vulnerabilities in java applications with static analysis.” in *Usenix Security*, vol. 2013, 2005.

[9] T. Lee, G. Won, S. Cho, N. Park, and D. Won, “Detection and mitigation of web application vulnerabilities based on security testing,” in *IFIP International Conference on Network and Parallel Computing*. Springer, 2012, pp. 138–144.

[10] S. Chung, L. Hansel, Y. Bai, E. Moore, C. Taylor, M. Crosby, R. Heller, V. Popovsky, and B. Endicott-Popovsky, “What approaches work best for teaching secure coding practices,” in *Proceedings of the 2014 HUIC Education and STEM Conference*, 2014.

[11] A. S. Sodiya, S. A. Onashoga, and O. B. Ajayi, “Towards building secure software systems,” *Issues in Informing Science and Information Technology*, vol. 3, pp. 635–646, 2006.

[12] N. Antunes and M. Vieira, “Defending against web application vulnerabilities,” *Computer*, vol. 45, no. 2, pp. 0066–72, 2012.

[13] A. Austin and L. Williams, “One technique is not enough: A comparison of vulnerability discovery techniques,” in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2011, pp. 97–106.

[14] J. Walden and M. Doyle, “Savi: Static-analysis vulnerability indicator,” *IEEE Security & Privacy*, vol. 10, no. 3, pp. 32–39, 2012.

[15] M. Gegick, L. Williams, J. Osborne, and M. Vouk, “Prioritizing software security fortification throughcode-level metrics,” in *Proceedings of the 4th ACM workshop on Quality of protection*. ACM, 2008, pp. 31–38.

[16] M. Gegick, P. Rotella, and L. Williams, “Predicting attack-prone components,” in *2009 International Conference on Software Testing Verification and Validation*. IEEE, 2009, pp. 181–190.

[17] 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.

[18] M. Finifter and D. Wagner, “Exploring the relationship between web application development tools and security,” in *USENIX conference on Web application development*, 2011.

[19] S. Clark, S. Frei, M. Blaze, and J. Smith, “Familiarity breeds contempt: The honeymoon effect and the role of legacy code in zero-day vulnerabilities,” in *Proceedings of the 26th annual computer security applications conference*. ACM, 2010, pp. 251–260.

[20] B. Martin, M. Brown, A. Paller, D. Kirby, and S. Christey, “2011 cwe/sans top 25 most dangerous software errors,” *Common Weakness Enumeration*, vol. 7515, 2011.

[21] R. Dhamankar, M. Dausin, M. Eisenbarth, J. King, W. Kandek, J. Ullrich, E. Skoudis, and R. Lee, “The top cyber security risks,” *SANS Institute*, 2009.

[22] T. OWASP, “10: Ten most critical web application security risks,” 2013.