The University Of Sheffield.

Finding Security Issues in (Open Source) Software Repositories

Zer Jun Eng

supervised by

Dr. Achim Brucker

This report is submitted in partial fulfilment of the requirement for the degree of MEng Software Enginnering by Zer Jun Eng

COM3610

21st October 2018

Declaration

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name : Zer Jun Eng

Date : 21st October 2018

Abstract

Acknowledgements

I would firstly like to thank my parents for their unconditional love and the full financial supoprt throughout my university life. It would not be possible for me to finish this project and my course without them.

I would also like to thank my supervisor, Dr. Achim Brucker, who are continuously providing constructive advice for my project. I am honoured to work with you, and I look forward to working with you in the future.

Contents

List of Tables

Chapter 1

Introduction

1.1 Background

Free/Libre and Open Source Software (**FLOSS**) is a type of software which its license allows the users to inspect, use, modify and redistribute the software's source code [6]. Since the introduction of Git, and later the Git repositories hosting site such as GitHub, many users have started to make their softwares open source by storing them as public repositories on GitHub. As a result, the participation of global communities into FLOSS projects has started to grow and different contributions were made to improve the softwares quality, which included fixing the software vulnerabilities [10].

Building a secure software is expensive, difficult, and time-consuming. In FLOSS projects, it is necessary to know when and how a security vulnerability is fixed. Therefore, having a list of changelogs or informative git commit messages that record the fixed security vulnerabilities is helpful. However, Arora and Telang [2] stated that some open source developers believe that public disclosure of security vulnerabilities patch is dangerous, thus vulnerability fixing commits are not commonly identified in some open source software repositories to prevent malicious exploits. Hence, a repository mining tool that investigate vulnerability patterns and identify vulnerable software components can be developed to reduce the time and cost required to mitigate the vulnerabilities.

1.2 Objectives

- Identify the security patterns of the most popular security issues in OWASP Top Ten Project. The patterns should be expressed using regular expressions.
- Develop a repository mining tool to search through the commit history of a repository and find a list of commit messages that match the patterns. The list should be produced in a suitable file format such as JSON, XML, or CSV.
- Extend the mining tool which checks the code difference in the commits found to obtain the actual commits fixing the security vulnerabilities. This extension should separate from the mining process to make the mining results easier to verify and debug.

1.3 Challenges

- **Data**: There are a large number of open source repositories available on GitHub. However, it is challenging to find a set of sample repositories that can produce accurate and consistent results.
- **Misclassification**: Commit messages for a same vulnerability patch are not always the same, thus misclassification of commit messages is inevitable. Using regular expressions to match the patterns in the mining process does not guarantee correctness of the result.
- **Evaluation**: After mining a list of commits that contain the identified patterns in its message, the evaluation process might not correctly locate the lines of code that addressed the security vulnerability. It might be required to perform manual evaluation to correctly identify some of the results.
- **Time**: Large repository such as Linux which has more than 780,000 commits in total [23] could be extremely time-consuming for the repository mining tool to complete the search and evaluation process.

1.4 Report Structure

Chapter 2 reviews a range of academic articles, theories and previous studies that is related to this project, as well as investigating the techniques and tools to be used.

Chapter 3 is a list of detailed requirements and a thorough analysis for design, implementation and testing stage. Some core decisions are reviewed in the analysis part to ensure the feasibility of the project.

Chapter 4 is a comparison between different design concepts, where the advantages and disadvantages of difference approach are stated. The chosen design is justified with suitable diagrams provided including wireframes and UML.

Chapter 5 describes the implementation process by highlighting novel aspects to the algorithms used. Testing are performed by following a suitable model to evaluate the implementation.

Chapter 6 presents all the results along with critical discussions about the main findings, and outlines the possible improvements that could be made in the future work.

Chapter 7 summarises the main points of previous chapters and emphasise the results found.

1.5 Relationship to Degree Programme

This project focuses on the reserach of real-world software security problems, and offers a valuable insights into the computer security. By studying the patterns of security vulnerabilities patch, practical knowledge for building and ensuring a secure system could be gained. Moreover, the difficulty of improving software security could be experienced during the evaluation process in this project. This relates to the Software Engineering degree as it requires a good understanding in version control system and it aims to improve softwares quality by reducing the time and effort needed to find security vulnerabilities in the source code.

Chapter 2

Literature Review

This chapter will start with the background contents of the project, and then focus on discussing the security aspect of open source softwares. Lastly, previous and existing relevant work are reviewed and a critical analysis is provided for the comparison of these resources and this project.

2.1 Open Source Security

There are currently two approaches to the license distribution of software: open source and closed source. The users of closed source software are limited to accept the level of security provided by their chosen vendor. In contrast, open source softwares provide more flexibility and freedom over the security option to their users [31], where the users can decide to wait for a patch from the vendor or collaborate with the community to develop their own.

Hoepman and Jacobs [16] suggested that open source softwares will have better security and reliability than closed source softwares through the power of open data and crowdsourcing. Conversely, Schryen [37] has shown that open source and closed source softwares do not have significant security dfference in his experiment and concluded that the policy of the developers is the main factor that determines the security. Wheeler [42] agrees with Schryen's conclusion, but he argued that Schryen's experiment had a small sample size, so the results may not be accurate.

While both statements might be true, Cowan [5] indicated that the security of a software is determined by many factors, and the source availability model is not the primary driver. Witten, Landwehr, and Caloyannides [44] stated that code review is the most effective method to improve system security and deduced that closed source systems could reach the same security level as open source systems if there were sufficient review processes being carried out. In recent years, the growing popularity of open source softwares has attracted a large number of people to join the community. Some open source softwares are now believed to have better quality as compared to their respective closed source softwares. However, this argument has yet to be proven in a formal way.

2.2 Common Vulnerabilities and Exposures

The Common Vulnerabilities and Exposures (**CVE**) is a project launched by Mitre Corporation and sponsored by the National Cyber Security Division of the United States Department of Homeland Security [9]. The CVE system provides the computer security community with a complete list of publicly known security vulnerabilities, and each vulnerability is identified by a unique CVE ID number. It is now the standardised solution and industry-recognised standard for identifying vulnerabilities and exposures.

2.3 Common Weakness Enumeration

The Common Weakness Enumeration (**CWE**) is another project of Mitre Corporation [7] that organises the software weaknesses into a list of different categories, known as the CWE list. Software weaknesses are defined as the errors that can lead to software vulnerabilities, which includes buffer overflows, authentication errors, code injection, etc. [8]. The CWE is now a formal standard for representing software weaknesses. Each entry in the CWE are identified by a unique ID number, and contained detailed information about the specific weakness.

2.4 Security Issues in Open Source Softwares

The Open Web Application Security Project (**OWASP**) is a worldwide non-profit organization committed to improve and raise the awareness of software security [29]. The project members of OWASP have worked together to produce a list of the most critical web application security risks based on the community feedback and comprehensive data contributed by different organizations. The list consists of ten categories of security attacks which are considered to be the most dangerous and popular in the recent years. The vulnerabilities listed by OWASP in 2017 [30] will be analysed and the security risks listed in older versions of the top ten project will covered too.

2.4.1 Injection

An injection attack is the exploitation of a software vulnerability where the attacker injects malicious code into the software applications and perform harmful executions. The most common types of injection attacks are Structured Query Language (**SQL**) injection, cross-site scripting, and shell injection [32]. Among these attacks, SQL injection attacks required the most effort to fix [27]. This is because a successful SQL injection attack could delete an entire database that could cause a massive data loss in extreme situation. SQL injection attacks are very widespread and easy to discover especially when attackers have access to the source code, or they could use a code scanner tool such as sqlmap [11] to find all possible ways of the injection attacks.

Vast majority of web applications make use of JavaScript to add automation and interactivity to their web pages to enhances the user experience. However, the growing usage of JavaScript has also increased the frequency and popularity of cross-site scripting (**XSS**) attacks. There three types of XSS attacks are: *reflected XSS*, *stored XSS*, and *DOM XSS* [21]. While the attack approach of different XSS methods may vary, their end goal is to steal or access the sensitive information of the victims [40]. XSS vulnerabilities could be easily detected and exploited at the same time, for example, the open source tool XSStrike [36] can scan a web page for possible XSS vulnerabilities. A promising approach to tackle this problem would be adding an extra security mechanism for handling the user inputs on the web pages.

2.4.2 Broken Authentication

Broken authentication happens when the attackers are allowed to perform malicious actions such as brute force dictionary attack on the authentication system. It might also be one of the outcome of a successful injection attack. According to Huluka and Popov [17], this vulnerability are very prevalent and has various causes, in which the lack of attention to security details is the most critial because developers often overlook certain scenarios which are likely to be exploited by attackers.

2.4.3 Using Componenets with Known Vulnerabilities

Components such as plugins, libraries, and modules can often be found in different parts of a software. Third-party components are increasingly being integrated into softwares to reduce the amount of time and effort required for development [3], but they also increase the risk of vulnerabilities being introduced into the softwares. These components are mostly maintened by different developers or organisations hence it is unable to guarantee that all the components used will be provided with the latest security patch.

Cadariu et al. [4] used OWASP Dependency Check tool [28] to find all known vulnerabilities in proprietary softwares written in Java. However, their results contained a considerable amount of both false positives and false negatives, which directly affected the precision of their technique. Therefore, it is also expected that the final results produced in this project might encounter the same problem.

2.5 Vulnerabilities Classification Techniques

Many classification techniques of computer attacks have been introduced before open source became popular, and these early works have significant influence on the later work of finding and classifying security issues in open source software repositories. An early work of Lindqvist and Jonsson [22] implied that the *location* of a computer flaw is a determining factor in the classification process, in which their results are obtained through real penetration attacks. Hansman and Hunt [14] adapted the ideas of Lindqvist and Jonsson and suggested the concept of dimensions, where they categorised the attacks into several layers.

2.6 Mining Software Repositories

The term 'Mining Software Repositories' (**MSR**) is a process of collecting and analysing big data from repositories, which includes version control repositories, mailing list repositories, and bug tracking repositories. The purpose of mining software repositories is to extract practical information from rich metadata and discover hidden trends about a specific evolutionary characteristic [20]. The information collected could be used in various development process. For example, some developers could gain insight by mining repositories, which may help them to enhance their software quality based on previous implementation evidence of other developers [15].

In order to identify both hidden and publicly disclosed patches, it is required to make effective use of MSR technique. A MSR process is normally carried out using tools or scripts made by the researchers themselves. Although there are many researches in the MSR field in the recent years, the majority of the tools or scrips used are not published publicly [35]. As a result, it is not possible to fully replicate the previous research methods and make improvements based on that. Despite the undisclosed information of research methods in many papers, Shang [38] suggested that the MSR process should be splitted into several stages, with each stage focusing on specific topic of the problem to achieve the optimal efficiency.

2.6.1 Keywords Search

Keywords search is the core procedure of retrieving information from a repository. The prerequisite is that the repository must have sufficient amount of valuable information, which can be estimated by judging the history of the repository. To correctly and precisely retrieve the information from a query, it is required to integrate some algorithms and modules into the search function. Matsushita, Sasaki, and Inoue [24] developed a repository search system that make use of two functions: lexical analysis function and token comparing function. The system produced very detailed results by deploying recursive search strategy into every commit. On the contrary, Mockus and Votta [25] designed an automated program that

2.6.2 Vulnerability Patch Prediction

Williams and Hollingsworth [43] developed a source code analysis tool that searches for bug fixes and combines with information mined from repositories to improve the results. It is stated that the most efficient way to utilise the historical information is to ignore the commit messages and focus on mining the code changes. In order to locate the actual code changes for the bug fix, a function return value checker was implemented to compares the number of warnings produced by a same function across different versions. Williams and Hollingsworth assumed that a bug is fixed if the warnings produced by the same function have decreased between two versions, and the final result produced is a list of functions that are related to a potential bug fix in the commit history.

2.6.3 Mining Vulnerabilities

This project extends prior work on Reis and Abreu's [33] Secbench Mining Tool. The tool aims to find vulnerabilities patch in GitHub repositories by using specific regular expressions for each vulnerability pattern. Then it creates a test case for every vulnerability found and these test cases are evaluated manually. Reis and Abreu [34] discussed the procedure of the evaluation and explained that human errors could occur due to source code complexity and similarity of vulnerability pattern. The approach of Secbench Mining Tool is similar to the concept of this project. However, it is not practical to perform manual evaluation on every result. In this project, the tool developed should be able to automate the evaulation process to some extent, while preserving the accuracy of the results.

Chapter 3

Requirements and Analysis

The purpose of this chapter is to express the aims in more details, and discuss the problems to be solved. This chapter will outline the requirements of the project and list the criteria to be met. The analysis part will cover every aspect of the design, implementation and testing stage to ensure that the project is feasible.

3.1 Project Objectives

Initially, the objectives set in **Section 1.2** are an ideal concept of this project. Having completed the background research and literature review, it is now possible to provide a detailed description and more clearly defined objectives that improve the feasibility of this project.

1. **Vulnerability patterns**: Identifying the each of the vulnerability pattern independently and individually is a time-comsuming process. There is also no guarantee that the patterns will produce accurate results during the testing stage. Therefore, it is decided to reuse and improve the patterns provided in previous related work.

2. **Mining the commits**: This task involves creating a repository mining tool that makes extensive use of the pre-defined regular expressions to search for the relative commits. It will be necessary to consider how

closely a commit needs to match with the patterns for it to be included in the result. The file format of the result will be discussed in the later section.

3. **Evaluating the mined commits**: The tool can be extended to include a separate function that evaluates the commits mined to find the actual code commit addressing the security vulnerabilities. However, all previous related work performed did not use automated techinques for the evaluation. This project will consider to automate the evaluation process to some extent while maintaining the accuracy of the results at the standard level.

3.2 Software Specification

Criteria	Importance
Compatibility: The mining tool should be able to run on all machines that meet the system requirements.	Essential
Completeness: The mining tool should be able to find all commits that match the regular expressions.	Essential
Lightweight: The mining tool should have a compact design and require the minimum amount of dependencies to run.	Essential
Repeatable: The results should be repeatable and reproducible.	Essential
Robustness: The mining tool should be able to handle all possible errors without terminating the mining process.	Essential
Automated Evaluation: The process of classifying and evaluating the commits into different vulnerabilities patch should be automated to a certain extent.	Desirable

Table 3.1: Specification of the mining tool

3.3 Considerations

The aim of this section is to contemplate the options available for this project and review some of the fundamental decisions to be made before the implementation.

3.3.1 Programming Language

Python 3 [41] is chosen to be the main programming language for the repository mining tool. While other programming languages may be more suitable for tackling specific problems of this project, Python 3 provides sufficient coverage over every aspect with its comprehensive functionality. The greatest advantage of Python 3 is that it has a wide range of libraries that facilitate the development environment, which fully justified that a complete working solution can be produced using Python 3.

3.3.2 Libraries and Tools

Since the mining tool is decided to be programmed in Python 3, a wide range of libraries could be integrated into the tool to enhance its functionality.

- PyGithub is a Python library build to access the GitHub API [18].
- GitPython is a Python library build to interact with Git repositories using a combination of python and git command implementation [13].

3.3.3 File Format of Result

The JavaScript Object Notation (**JSON**) [19] has been chosen as the file format for storing the results in this project. This is because JSON is supported in Python and it does not require complicated operations in Python to access the data. While various alternative data interchange formats such as the Extensible Markup Language (**XML**) [12] has its unique advantages, it is important to choose a data interchange format that consumes less resource and have lower processing time for large amount of data. Since it has been proved that JSON has better performance than XML in terms of processing time and resource utilisation [26], it is considered that JSON would be the best option for this project.

3.4 Proposed Method

This project strongly emphasises the need for finding security issues in open source repositories through MSR. While it might be impossible to discover the security patches in a repository through a single search, the problem could be solved using divide and conquer. The ideal concept of this project is to build a command-line interface program that is able to run two separate process: the **mining** process and the **evaluation** process. The **mining** process takes a Git repository as input, searches through the commit log, and stores the list of commits that might potentially contain a patch in a JSON file. The **evaluation** process takes a JSON file as input, and check the code difference of every commit in the log file to identify the real patches.

3.5 Problems and Constraints

As mentioned in **Section 1.3**, the main challenges of this project are **data**, **misclassification**, **evaluation** and **time**. This section will discuss the problems in detail and review several ways of mitigating them, as well as analysing the possible constraints that might affect the progress of the project.

It is expected that some problems might be solved and new problems could emerge in the course of the project. There are several ways of mitigating the problems to reduce the risk, provided that the problems are clearly identified and they are under the project scope. It is estimated that the **evaluation** process would be the biggest challenge of this project since it was regarded as a complicated and difficult area in previous researches. Moreover, this project plans to implement an automated version of the evaluation process, which will further increases the difficulty level.

Automated evaulation is hard and does not guarantee to provide a good result. It is also extremely challenging for the tool to work across repositories programmed in different programming languages. The constraint is that the tool has to be exhaustively tested to find the optimal threshold value and for it to be automated and produce good results. Although the tool might produce good results on some repositories, it does not indicate that the tool will produce consistent results on all repositories. To ensure the minimum quality of the results, one of the solutions might be using both automated method for basic filtering and manual method for advance refinement.

3.6 Testing

This section covers a brief overview of the testing stage. It will be necessary to consider some of the test cases and scenarios in advance to ensure that the requirements and criteria listed are practical and feasible.

3.6.1 Unit Testing

Python provides an unit testing framework as part of its standard library, known as unittest [39], which offers a complete set of functions suffice to cover the testing stage of this project.

3.6.2 System Testing

After completing the unit testing, the mining tool has to be tested for its completeness and robustness, as mentioned in **Table 3.1**. It is expected that the program would not be able to handle complicated errors during the early implementation, and the project schedule would become a iterative process between implementation and testing. It is assumed that the testing stage would be the most time-consuming process in the whole project, thus it might be required to allocate more time and effort into this stage

3.7 Ethical Issues

In this project, it has to be clearly declared that any known or unknown vulnerabilities found by the tool in any repositories will not be publicly disclosed without the permission of the original authors. The reason is that there are no guarantee that the respective software vendors will respond to any of the vulnerabilities found, and publishing the vulnerabilities publicly would make the softwares highly vulnerable to attackers [1].

Bibliography

[1] A. Arora, R. Krishnan, R. Telang and Y. Yang, 'An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure', *Information Systems Research*, vol. 21, no. 1, pp. 115–132, 1st Mar. 2010. DOI: 10.1287/isre.1080.0226.

[2] A. Arora and R. Telang, 'Economics of software vulnerability disclosure', *IEEE security & privacy*, vol. 3, no. 1, pp. 20–25, 14th Feb. 2005, ISSN: 1540-7993. DOI: 10.1109/MSP.2005.12.

[3] D. Balzarotti, M. Monga and S. Sicari, 'Assessing the risk of using vulnerable components', in *Quality of Protection*, D. Gollmann, F. Massacci and A. Yautsiukhin, Eds., Springer, 2006, pp. 65–77, ISBN: 978-0-387-36584-8. DOI: 10.1007/978-0-387-36584-8_6.

[4] M. Cadariu, E. Bouwers, J. Visser and A. van Deursen, 'Tracking known security vulnerabilities in proprietary software systems', in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, IEEE, Montreal, QC, Canada, Mar. 2015, pp. 516–519, ISBN: 978-1-4799-8469-5. DOI: 10.1109/SANER.2015.7081868.

[5] C. Cowan, 'Software security for open-source systems', *IEEE Security & Privacy*, vol. 99, no. 1, pp. 38–45, 19th Feb. 2003, ISSN: 1540-7993. DOI: 10.1109/MSECP.2003.1176994.

[6] K. Crowston, K. Wei, J. Howison and A. Wiggins, 'Free/libre open-source software development: What we know and what we do not know', *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, p. 7, 1st Feb. 2012, ISSN: 0360-0300. DOI: 10.1145/2089125.2089127.

[7] *Cve – about cwe*, Mitre Corporation, 30th Mar. 2018. [Online]. Available: https://cwe.mitre.org/about/index.html (visited on 09/10/2018).

[8] *Cve – frequently asked question (faq)*, Mitre Corporation, 30th Mar. 2018. [Online]. Available: https://cwe.mitre.org/about/faq.html#A.1 (visited on 09/10/2018).

[9] *Cve – home*, Mitre Corporation, 17th Jan. 2018. [Online]. Available: https://cve.mitre.org/about/index.html (visited on 09/10/2018).

[10] L. Dabbish, C. Stuart, J. Tsay and J. Herbsleb, 'Social coding in github: Transparency and collaboration in an open software repository', in *Proceedings of the ACM 2012 conference on computer supported cooperative work*, ACM, 11th Feb. 2012, pp. 1277–1286. DOI: 10.1145/2145204.2145396.

[11] B. Damele and M. Stampar, *Sqlmap: Automatic sql injection and database takeover tool.* [Online]. Available: http://sqlmap.org/ (visited on 11/10/2018).

[12] *Extensible markup language (xml).* [Online]. Available: https://www.w3.org/XML/ (visited on 14/10/2018).

[13] *Gitpython.* [Online]. Available: https://github.com/gitpython-developers/GitPython (visited on 20/09/2018).

[14] S. Hansman and R. Hunt, 'A taxonomy of network and computer attacks', *Computers & Security*, vol. 24, no. 1, pp. 31–43, 2005-01-28, ISSN: 0167-4048. DOI: 10.1016/j.cose.2004.06.011.

[15] A. E. Hassan, 'The road ahead for mining software repositories', in *2008 Frontiers of Software Maintenance*, IEEE, Sep. 2008, pp. 48–57. DOI: 10.1109/FOSM.2008.4659248.

[16] J.-H. Hoepman and B. Jacobs, 'Increased security through open source', *Communications of the ACM*, vol. 50, no. 1, pp. 79–83, 1st Jan. 2007, ISSN: 0001-0782. DOI: 10.1145/1188913.1188921.

[17] D. Huluka and O. Popov, 'Root cause analysis of session management and broken authentication vulnerabilities', in *World Congress on Internet Security (WorldCIS-2012)*, IEEE, 12th Jun. 2012, pp. 82–86, ISBN: 978-1-908320-04-9.

[18] V. Jacques, *Pygithub*, PyGithub. [Online]. Available: `https://github.com/PyGithub/PyGithub` (visited on 20/09/2018).

[19] *Json*, ecma International. [Online]. Available: `https://www.json.org/` (visited on 13/10/2018).

[20] H. Kagdi, M. L. Collard and J. I. Maletic, 'A survey and taxonomy of approaches for mining software repositories in the context of software evolution', *Journal of software maintenance and evolution: Research and practice*, vol. 19, no. 2, pp. 77–131, 29th Mar. 2007. DOI: `10.1002/smr.344`.

[21] A. Kieyzun, P. J. Guo, K. Jayaraman and M. D. Ernst, 'Automatic creation of sql injection and cross-site scripting attacks', in *Proceedings of the 31st International Conference on Software Engineering*, ser. ICSE '09, Washington, DC, USA: IEEE Computer Society, 2009, pp. 199–209, ISBN: 978-1-4244-3453-4. DOI: `10.1109/ICSE.2009.5070521`.

[22] U. Lindqvist and E. Jonsson, 'How to systematically classify computer security intrusions', in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, IEEE, May 1997, pp. 154–163. DOI: `10.1109/SECPRI.1997.601330`.

[23] *Linux kernel source tree*. [Online]. Available: `https://github.com/torvalds/linux` (visited on 20/09/2018).

[24] M. Matsushita, K. Sasaki and K. Inoue, 'Coxr: Open source development history search system', in *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, IEEE, Dec. 2005. DOI: `10.1109/APSEC.2005.56`.

[25] A. Mockus and L. G. Votta, 'Identifying reasons for software changes using historic databases', in *Proceedings 2000 International Conference on Software Maintenance*, IEEE, 2000, pp. 120–130, ISBN: 0-7695-0753-0. DOI: `10.1109/ICSM.2000.883028`.

[26] N. Nurseitov, M. Paulson, R. Reynolds and C. Izurieta, ‘Comparison of json and xml data interchange formats: A case study’, vol. 9, Jan. 2009, pp. 157–162.

[27] L. B. Othmane, G. Chehrazi, E. Bodden, P. Tsalovski, A. D. Brucker and P. Miseldine, ‘Factors impacting the effort required to fix security vulnerabilities’, in *Information Security*, J. Lopez and C. J. Mitchell, Eds., Cham: Springer International Publishing, 2015, pp. 102–119, ISBN: 978-3-319-23318-5. DOI: `10.1007/978-3-319-23318-5_6`.

[28] *Owasp dependency check*, The Open Web Application Security Project (OWASP), 16th Sep. 2018. [Online]. Available: `https://www.owasp.org/index.php/OWASP_Dependency_Check` (visited on 06/10/2018).

[29] *Owasp home*, The Open Web Application Security Project (OWASP), 18th Sep. 2018. [Online]. Available: `https://www.owasp.org/index.php/Main_Page` (visited on 29/09/2018).

[30] *Owasp top ten 2017 project*, The Open Web Application Security Project (OWASP), 20th Oct. 2017. [Online]. Available: `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project` (visited on 26/09/2018).

[31] C. Payne, ‘On the security of open source software’, *Information Systems Journal*, vol. 12, no. 1, pp. 61–78, 8th Feb. 2002, ISSN: 1350-1917. DOI: `10.1046/j.1365-2575.2002.00118.x`.

[32] T. Pietraszek and C. V. Berghe, ‘Defending against injection attacks through context-sensitive string evaluation’, in *Recent Advances in Intrusion Detection*, A. Valdes and D. Zamboni, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 124–145, ISBN: 978-3-540-31779-1. DOI: `10.1007/11663812_7`.

[33] S. Reis and R. Abreu, *Secbench mining tool*, The Quasar Research Group. [Online]. Available: `https://github.com/TQRG/secbench-mining-tool` (visited on 11/10/2018).

[34] S. Reis and R. Abreu, ‘Secbench: A database of real security vulnerabilities’, *Secure Software Engineering in DevOps and Agile Development*, M. G. Jaatun and D. S. Cruzes, Eds., pp. 69–85, 31st Oct. 2017.

[35] G. Robles, 'Replicating msr: A study of the potential replicability of papers published in the mining software repositories proceedings', in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, IEEE, May 2010, pp. 171–180, ISBN: 978-1-4244-6803-4. DOI: 10.1109/MSR.2010.5463348.

[36] S. Sangwan, *Xsstrike: Intelligent xss detection & exploitation suitel.* [Online]. Available: http://xsstrike.tk/ (visited on 12/10/2018).

[37] G. Schryen, 'Is open source security a myth?', *Communications of the ACM*, vol. 54, no. 5, pp. 130–140, 1st May 2011, ISSN: 0001-0782. DOI: 10.1145/1941487.1941516.

[38] W. Shang, Z. M. Jiang, B. Adams and A. E. Hassan, 'Mapreduce as a general framework to support research in mining software repositories (msr)', in *2009 6th IEEE International Working Conference on Mining Software Repositories*, IEEE, May 2009, pp. 21–30, ISBN: 978-1-4244-3493-0. DOI: 10.1109/MSR.2009.5069477.

[39] *Unittest - unit testing framework*, Python Software Foundation. [Online]. Available: https://docs.python.org/3/library/unittest.html (visited on 14/10/2018).

[40] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel and G. Vigna, 'Cross site scripting prevention with dynamic data tainting and static analysis.', in *NDSS*, vol. 2007, 2007, p. 12.

[41] *Welcome to python.org*, Python Software Foundation. [Online]. Available: https://www.python.org/ (visited on 13/10/2018).

[42] D. A. Wheeler, *Why open source software/free software (oss/fs, floss, or foss)? look at the numbers*, 2015. [Online]. Available: https://www.dwheeler.com/oss_fs_why.html (visited on 18/09/2018).

[43] C. C. Williams and J. K. Hollingsworth, 'Automatic mining of source code repositories to improve bug finding techniques', *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 466–480, Jun. 2005, ISSN: 0098-5589. DOI: 10.1109/TSE.2005.63.

[44] B. Witten, C. Landwehr and M. Caloyannides, 'Does open source improve system security?', *IEEE Software*, vol. 18, no. 5, pp. 57–61, Sep. 2001, ISSN: 0740-7459. DOI: 10.1109/52.951496.