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

8th 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 : 8th October 2018

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

Declaration i

Acknowledgements ii

List of Tables v

1 Introduction 1

1.1 Background 1

1.2 Objectives 2

1.3 Challenges 2

1.4 Report Structure 3

1.5 Relationship to Degree Programme 3

2 Literature Review 4

2.1 Open Source Security 4

2.2 Security Issues in Open Source Softwares 5

2.2.1 Injection 5

2.2.2 Broken Authentication 6

2.2.3 Using Componenets with Known Vulnerabilities 6

2.3 Vulnerabilities Classification Techniques 6

2.4 Related Work 7

3 Requirements and Analysis 8

3.1 Software Specification 8

3.2 Problems 9

3.3 Proposed Method 9

3.4 Tools 9

Bibliography **10**

List of Tables

3.1 Criteria to be met in this project 8

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 [5]. 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 [6].

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 [1] 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 numbers 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 [13] 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 [19], 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 [9] 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 [23] 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 [24] 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 [4] 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 [25] 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 numbers 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 Security Issues in Open Source Softwares

The Open Web Application Security Project (**OWASP**) is a worldwide nonprofit organization committed to improve and raise the awareness of software security [17]. 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 [18] will be analysed and the security risks listed in older versions of the top ten project will covered too.

2.2.1 Injection

An injection attack is the exploitation of a software vulnerability where the attacker injects malicious code into the software and perform harmful executions. Structured Query Language (**SQL**) injection is the most common type of injection attack in web applications [20]. In addition, **SQL** injection attacks required the most effort to fix [15]. Injection flaws are very widespread and easy to discover when attackers have access to the source code, where they could use a code scanner tool to find all possible ways of the injection attacks.

2.2.2 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 [10], 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.

2.2.3 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 [2], 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.

2.3 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 [12] 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 [8] adapted the ideas of Lindqvist and Jonsson and suggested the concept of dimensions, where they categorised the attacks into several layers.

2.4 Related Work

Matsushita, Sasaki, and Inoue [14] developed a system known as CoxR that is able to analyse open source repositories to search for code fragments, files and commit logs through the keywords provided. The CoxR system has an integrated analysis module, and the module consists of two functions: lexical analysis function and token comparing function. Although the main objective of CoxR system is not about finding security related commits in open source repositories, the methodology of CoxR system for searching the commit log is suitable to be used as an implementation reference.

Cadariu et al. [3] used **OWASP** Dependency Check tool [16] 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.

This project extends prior work on Reis and Abreu's [21] 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 [22] 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.

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 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
Repeatable: The results should be repeatable and reproducible.	Essential
Automated Evaluation The process of classifying and evaluating the commits into different vulnerabilities patch should be automated.	Desirable

Table 3.1: Criteria to be met in this project

3.2 Problems

As mentioned in **Section 1.2**, the repository mining tool must be able to detect commits that contain distinct patterns such as *fix*, *patch*, *vulnerability* etc. After extracting a possible list of commits, it should perform an evaluation process to identify the actual commits that fixed security vulnerability. This could be hard because not all open source software repositories are using the same programming language. Hence, it might be difficult to determine the actual lines of code that addressed the vulnerabilities.

One of the major challenges in this project is the classification or evaluation of the vulnerabilities.

3.3 Proposed Method

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 return the list of commits that might potentially contain a patch as a log file (JSON, HTML, etc.). The **evaluation** process takes a log file as input, and check the code difference of every commit in the log file to identify the real patches.

3.4 Tools

- PyGithub is a Python library build to access the GitHub API [11].
- GitPython is a Python library build to interact with Git repositories using a combination of python and git command implementation [7].
- Secbench Mining Tool is a repository mining tool build by The Quasar Research Group to mine vulnerability patterns from GitHub repositories [21].

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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 co-*

operative work, ACM, 11th Feb. 2012, pp. 1277–1286. DOI: 10.1145/2145204.2145396.

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] *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).

[17] *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).

[18] *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).

[19] 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`.

[20] 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`.

[21] S. Reis and R. Abreu, *Secbench mining tool*, The Quasar Research Group. [Online]. Available: `https://github.com/TQRG/secbench-mining-tool` (visited on 20/09/2018).

[22] 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.

[23] 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`.

[24] 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).

[25] 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`.