Analysis of GitHub and Stack Overflow Code for Malicious Content

STEVEN NYEO, Case Western Reserve University, USA
PATRICK HOGRELL, Case Western Reserve University, USA
ZUBAIR MUKHI, Case Western Reserve University, USA
CHRIS SHORTER, Case Western Reserve University, USA

ACM Reference Format:
Steven Nyeo, Patrick Hogrell, Zubair Mukhi, and Chris Shorter. 2019. Analysis of GitHub and Stack Overflow Code for Malicious Content. 1, 1 (December 2019), 5 pages.

Contents

Authors' addresses: Steven Nyeo, Case Western Reserve University, Department of Electrical, Computer, and Systems Engineering, Cleveland, Ohio, 44106, USA, cxn152@case.edu; Patrick Hogrell, Case Western Reserve University, Department of Computer and Data Sciences, Cleveland, Ohio, 44106, USA, pjh96@case.edu; Zubair Mukhi, Case Western Reserve University, Department of Computer and Data Sciences, Cleveland, Ohio, 44106, USA, zxm132@case.edu; Chris Shorter, Case Western Reserve University, College of Arts and Sciences, Cleveland, Ohio, 44106, USA, cws68@case.edu.

1 ACKNOWLEDGEMENTS

We would like to acknowledge Professor Yanfang (Fanny) Ye from the Department of Computer and Data Sciences at Case Western Reserve University, as well as VirusTotal.com for their assistance in this project.

2 ABSTRACT

The Internet and specifically code repositories serve as a space for the spread of solutions to complex software problems, computing research, and software libraries and executables that decrease reproduction of labor. One of the most popular repository sites is GitHub. GitHub, however, has limited protection for end-users, as GitHub only recently started adding code screening this year[1]. For example, a malicious GitHub repository posing as a benign solution to a common but time-intensive coding problem could easily disseminate malicious code or poor programming practices that allow for bad actors to compromise otherwise safe systems. Additionally, malware and exploits are occasionally hosted on GitHub[2]. While these files are often made public for the purpose of full disclosure, there is still a potential that a popular fork of the project can contain malware. To resolve this, we built an automated webscraper and an analysis tool to locate and analyze GitHub projects and ensure their security.

3 MOTIVATION AND BACKGROUND

A large number of people and programmers use sites such as GitHub to quickly develop code with many people. We know that some people will take advantage of the good will of these people and use it for malicious purposes, such as embedding a Trojan or a virus in the code. Our questions are, can we find people who have done this and figure out the length they have distributed their code already? Can we figure out a way to prevent it happening more in the future?

3.1 Initial Webscraper Design

We initially approached the challenge of discovering malicious repositories by scraping the most popular repositories in selected and popular topics. We generated negligible data (2355 total repositories) with a single malicious repository located even when including topics such as "malware" and "exploit."

4 LITERATURE SURVEY

5 METHODOLOGY

For this project, we divided the work into three specific tasks: Repository Scraping, Repository Scanning, and Repository Analysis. All tasks were run on a Raspberry Pi 3B+ running Raspbian 9.11.

[1] See Literature Review, The Supreme Backdoor Factory
[2] see https://github.com/frohoff/ysoserial as an example of a proof-of-concept exploit hosted on GitHub

5.1 Repository Scraping

5.1.1 Webscraper Design. To generate a list of repositories for analysis, we designed a webscraper in Python 3 using the BeautifulSoup and regex libraries. This scraper utilizes a recursive depth-first search (DFS) starting with a single GitHub repository URL as its starting point. The tool takes two arguments: seed URL and graph depth (TTL). For any given repository URL, the scraper locates the username from the seed URL, locates all repositories the user has created, and generates a list of the user's followers before executing the search on each follower. By including a decrementing TTL value in each function call, the scraper avoids infinite loop conditions that could be caused by rings of users following each other. After the scraper instance generates its list of repositories, it compiles all projects from child processes and returns it to the parent process. The scraper then ensures that there are no duplicate repositories by re-generating the list through Python's inbuilt dictionary function before parsing all repositories to a Comma-Separated Values (CSV) file. This scraper regularly generated tens of thousands of results at a TTL value of 2 (3 layers of DFS).

5.2 Repository Scanning

In order to scan the list of repositories we scraped, we created a repository scanning command-line tool, written in Java, that uses the VirusTotal (VT) API to test repositories for malware. The tool has two modes: single repo (-s argument), which scans a single repository URL provided as a parameter and repo list (-a argument), which takes the file-path of a CSV file of repository URLs generated by the above scraper as a parameter and scans each of them consecutively. In order to scan each repo, the scanner first uses the Apache Commons IO API to download either the tarball (.tar.gz) or zip file from GitHub. It is then placed in a temporary folder while the scanner uploads it to VT. There are either two or three steps needed to execute a VT file scan depending on the size of the file. If the file is less than 32MB, it uploads it straight to the regular VT file scan API endpoint. It receives a VT scan results page URL and waits for five minutes to call the results page and retrieve the results to ensure that scanning completes. For files between 32MB and 220MB, it requests a special URL for larger file scans. From there, it uploads the file, and follows the same delayed retrieval process. VT will not allow us to upload files larger than 220MB, so the scanner flags the scan as an error. In single scan mode, the scanner writes the repository scan results to the console. In list scanning mode, the scanner creates a new CSV file with the results of the scan in the tool's results folder.

5.3 Repository Analysis

The results provided in the results CSV generated by the scanner include a "Scan Result" to indicate the outcome of each scan. There are three possible outcomes to a scan: "Pass", "Fail", and "Error". A repository is considered malicious (a "Fail") if it is uploaded successfully and fails over 25% of VT's anti-malware scans are failures. A repository "Passes" if it is uploaded successfully and fails less than or equal to 25% of its anti-malware scans. The repository state is an "Error" if the scanner is unable to download the file or if the file is too big to upload to VT. If any other error that prevents the scanner from being able to successfully download or upload the repository, it will also be considered an "Error". From there, we compiled the CSV files from multiple scans and sorted the values to locate failing repositories and explore their methods of dissemination. We then manually examined the malicious repositories discovered by the scanner and we selected repositories and users we determined to be "Users of Interest" based on several criteria. First, we sorted the results to see which repositories were being forked the most. Then, we chose one of those repositories to view where that repository was originally forked from. From there, we recorded that user's name and considered that user as a potential "user of interest." The users whose names came up the most, we recorded as our final list of "Users of Interest."

6 RESULTS

From the results we have analyzed, we have findings related to the methodology and results of our scans and have additionally identified 10 users of interest.

6.1 Overall Findings

We have discovered a considerable number of repositories that have been marked as "Fails." However, many of these repositories are in fact platforms for penetration tests or tools to address other security measures. Therefore, these repositories containing suspicious content are intentionally made public for non-malicious purposes. They are readily available for modification from everyone.

A repository publicly declared as malware can often be considered less harmful than a repository that does not declare its dangerous nature, as explicit disclosure indicates that the owner is aware of the threat their repository can pose.

The parts marked as malicious are naturally part of the project, and so long as the repository description has declared its intention of publicly releasing malicious repositories, the repository itself should not be considered malicious.

The intentions of the repositories that we have marked as failing are unknown, and remains a challenge to be analyzed automatically in the future.

6.2 Users of Interest

6.2.1 RedCanaryCo. Project: Atomic Red Team[3]

Failing rate is 41 / 58 = 70.6 %

6.2.2 swisskyrepo. Project: PayloadsAllTheThings [4]

Failing rate is 39 / 59 = 66.1 %

6.2.3 SecWiki. Project: linux-kernel-exploits[5]

Failing rate is 31 / 60 = 51.7 %

Project: windows-kernel-exploits[6]

Failing rate is 24 / 54 = 44.4 %

6.2.4 misterch0c. Project: APT34[7]

Failing rate is 32 / 60 = 53.33 %

Project: shadowbroker[8]

Failing rate is 17 / 48 = 35.4 %

6.2.5 Player 5.

6.2.6 Player 6.

6.2.7 Player 7.

[3]https://github.com/redcanaryco/atomic-red-team
[4]https://github.com/swisskyrepo/PayloadsAllTheThings
[5]https://github.com/SecWiki/linux-kernel-exploits
[6]https://github.com/SecWiki/windows-kernel-exploits
[7]https://github.com/misterch0c/APT34
[8]https://github.com/misterch0c/shadowbroker

6.2.8 Player 8.

6.2.9 Player 9.

6.2.10 Player 10.

7 CONCLUSION