It Could Git Worse: Analysis of GitHub Repositories for Malicious Content

STEVEN CHEN HAO 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 Chen Hao Nyeo, Patrick Hogrell, Zubair Mukhi, and Chris Shorter. 2019. It Could Git Worse: Analysis of GitHub Repositories for Malicious Content. 1, 1 (December 2019), 8 pages.

Contents

Authors' addresses: Steven Chen Hao 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 programs 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 code repository sites such as GitHub to quickly develop code by collaborating with many people. We know that bad actors will take advantage of the goodwill of these people and hijack popular projects for malicious purposes, such as embedding a Trojan or a virus in the code. Our questions are: Can we find networks of users that have distributed malicious code? Can we figure out a way to prevent it happening more in the future?

[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

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

4.1 The Supreme Backdoor Factory [3]

This article outlines the process a single data forensics researcher took to identify a ring of malicious GitHub accounts. These accounts maintained projects that were forks of legitimate projects modified to download an e-commerce bot with remote command execution capabilities. By graphing a series of accounts starring and following each other, the researcher located an isolated ring of automated accounts used to host single files. We used similar logic, searching for malicious repositories via the followers of known malicious repositories out to two degrees of separation, for our large-scale analysis of malicious repositories.

4.2 How Bad Can It Git? Characterizing Secret Leakage in Public GitHub Repositories [4]

This article summarizes the results of a research group's findings regarding public disclosure of private API keys. The findings indicate that hundreds of thousands of API keys that ought to be private are publicly scrapable and locateable, including keys for commonly used platforms and services. This validated our belief that large-scale analysis of repositories was not only feasible, but could yield viable results.

4.3 GitHub Starts Scanning Millions of Projects for Insecure Components[5]

In 2017, GitHub announced that they have started scanning all of its repositories for repositories that are vulnerable and need security updates. They sent warnings to the owners of vulnerable repositories that their repositories needed security updates and recommending that they take steps to resolve the issue. This proves that GitHub does actually have the capability to do a full database scan of some capacity. Since GitHub has this capability, they should heavily consider doing so to protect their users and ensure their platform is not being used maliciously.

5 METHODOLOGY

5.1 Webscraper Design

To generate a list of repositories for analysis, we designed a webscraper in Python 3 using the BeautifulSoup, requests, and re (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

[3]https://dfir.it/blog/2019/02/26/the-supreme-backdoor-factory/
[4]https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_04B-3_Meli_paper.pdf
[5]https://nakedsecurity.sophos.com/2017/11/21/github-starts-scanning-millions-of-projects-for-insecure-components/

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** [6] Failing rate: 41/58 = 70.6%

6.2.2 swisskyrepo.

- **Project: PayloadsAllTheThings** [7] Failing rate: 39/59 = 66.1%

6.2.3 SecWiki.

- **Project: linux-kernel-exploits** [8]
  Failing rate: 31/60 = 51.7%
  This repository, uploaded by SecWiki, contains a list of exploits for the Linux Kernel throughout the years from 2004 to 2018. The repository contains 82.4% of code in C, indicating that the codebase is targeted towards exploiting lower-level aspects of the Linux system.
  The code from this repository is marked as malicious possibly due to its intention of bypassing the default user privileges of accessing the operating system kernel in order to demonstrate the kernel exploits. However, SecWiki declares that it is a sole list of identified kernel exploits so far, and anticipates more participants to continue adding newly found Linux kernel exploits to the list.
  In addition, the repository is licensed under the MIT License. SecWiki explicitly claims its exemption of responsibility on any attempts of using the codebase for malicious intents and purposes, and further informs the user of the risk of using the code provided by the repository.

- **Project: windows-kernel-exploits** [9]
  Failing rate: 24/54 = 44.4%
  This repository, uploaded by SecWiki, contains a list of exploits for the Windows NT Kernel throughout the years from 2003 to 2018.
  The repository contains 46.5% of code in C and 18.2% of code in C++. From this large proportion of low-level C/C++ composition, it is possible that the repository is marked as malicious for the same reason as the Linux kernel exploit repository mentioned above because of certain illegal access of Windows NT kernel privileges on the Windows system.

[6]https://github.com/redcanaryco/atomic-red-team
[7]https://github.com/swisskyrepo/PayloadsAllTheThings
[8]https://github.com/SecWiki/linux-kernel-exploits
[9]https://github.com/SecWiki/windows-kernel-exploits

Similar to the Linux kernel exploit repository, the Windows NT kernel exploit repository is also licensed under the MIT License. SecWiki also does not take responsibility for any damage and risks caused by the repository.

6.2.4 misterch0c.

- **Project: APT34** [10]
  Failing rate: 32/60 = 53.33%

- **Project: shadowbroker** [11]
  Failing rate: 17/48 = 35.4%

6.2.5 YSRC.

- **Project: xunfeng** [12]
  Failing rate: 18/60 = 30%

Xunfeng is an emergency response system for corporate internet systems. It's essentially a quick penetration test for companies to use to analyze their networks system quickly. It is set up to run in advance and periodically runs on the system to check for network vulnerabilities. It was put together by YSRC, which is an acronym that translates to the Tongcheng Safety Emergency Response Center. YSRC hosts other similar products on GitHub that handle risk prevention, detects when code is being run in an emulator, and runs other types of penetration tests.

6.2.6 grayddq.

- **Project: Gscan** [13]
  Failing rate: 22/60 = 36.7%

6.2.7 k8gege.

- **Project: K8tools** [14]
  Failing rate: 16 /47 = 34%

Like many of the other projects that flagged in our scan, k8tools is a penetration testing tool that is meant for research use only. K8gege, which translates to k8 brother, is a blog containing a number of different pages and posts about computer security topics and malware research. The K8gege GitHub page hosts a number of different tools and malware analyses that appeared on their blog, so that their users can test them out for themselves.

K8tools is essentially a compilation of bunch of different penetration tests that appear on the blog, put together to make one useful tool. One of these, Ladon, is named after a hundred headed dragon in Greek Mythology, because it is a heavily multi-threaded, comprehensive scanning service for penetration testing. It tests network assets, password blasting, and numerous other vulnerabilities that are commonly found in programs.

6.2.8 PowerShellMafia.

- **Project: PowerSploit**[15]
  Failing rate: 34/59 = 57.6%

[10]https://github.com/misterch0c/APT34
[11]https://github.com/misterch0c/shadowbroker
[12]https://github.com/ysrc/xunfeng
[13]https://github.com/TomANDJarry/Gscan
[14]https://github.com/k8gege/K8tools
[15]https://github.com/PowerShellMafia/PowerSploit

PowerSploit is an open source penetration testing tool created by the Mitre Att&ck program. Mitre is a corporation that works in a number of different fields to ensure people's safety and well-being. One of their main responsibilities is analyzing cybersecurity tactics, and their Att&ck program collects knowledge of cybersecurity attacks and makes them available for analysis and protective purposes. In this circumstance, their PowerSploit tool is being hosted by an independent user named PowerShellMafia, who also hosts a number of PowerShell based penetration testing tools.

PowerSploit essentially uses Microsoft PowerShell to run a number of different penetration tests on a project or a machine. It has tools to help it bypass anti-virus systems, extract data from compromised machines, monitor keystrokes, take screenshots, and record microphone audio. It even has tools designed to create general mayhem on an infected machine, without having a specific goal in mind. In order to help their platform grow, it allows its users to fork and add additional tests to their tool. That means that while this directory is not actually malicious, and is really meant for testing, in certain scenarios, the people who are forking and adding to their tools could theoretically use it to weaponize the product.

6.2.9 *b4rtik.*

- **Project: RedPeanut**[16]
  Failing rate (original): 14/58 = 24.1%
  Failing rate (RedTeamWing fork): 17/60 = 28.3%
  Failing rate (RunOnceEx fork): 18/59 = 30.5%

RedPeanut, developed by b4rtik, is an open-source .Net Remote Access Trojan (RAT) proof-of-concept currently under testing. RATs are malware that allow a remote actor control over a computer system without the system users' permission. This can often take the form of keylogging or using a system as part of a botnet. RedPeanut specifically is designed to allow remote users to "practice and experiment with various evasion techniques related to the dotnet environment, process management and injection." It also contains an execution agent capable of avoiding detection by spawning a new process to execute the commands and prevent the parent process from being detected by antivirus software. RedPeanut is also notable because the origin repository was not detected as malicious by our standards, missing the mark by one engine on VT, while forks from RedTeamWing and RunOnceEx were detected as malicious by VT.

6.2.10 *cobbr.*

- **Project: Covenant**[17]
  Failing rate: 25/59 = 42.3%

Covenant was directly referenced by language in RedPeanut's Readme.md file, and

- **Project: SharpSploit**[18]
  Failing Rate: 32/60 = 53.3%

This is a description of SharpSploit

6.2.11 *aainz.*

- **Project: TinyNuke**[19]

[16]https://github.com/b4rtik/RedPeanut
[17]https://github.com/cobbr/Covenant
[18]https://github.com/cobbr/SharpSploit
[19]https://github.com/rossja/TinyNuke

Failing rate: 18/59 = 30.5%

TinyNuke is a Zeus-like banking Trojan that was created and subsequently deleted by aainz to prevent it from being used maliciously, but it was forked and put back up by multiple people. The code itself was obfuscated to make it more difficult to detect. It utilizes Formgrabber and Webinjects for Firefox, Internet Explorer and Chrome to get into pages, and can inject on both x86 and x64 browsers. It is programmed in mainly C++ and C. While this is a minor project and player, it is a definitive malicious codebase that might have been developed for research, but was determined to be too significant of a threat by aainz, so they took it down.

7 COUNTERMEASURES

Although our scan did not find any malicious repositories that tried to pass as harmless, in order to ensure that people do not use malicious libraries when building projects, we have developed a few countermeasures.

7.1 GitHub Full Repository Scan

As noted in the literature review, GitHub has started scanning every repository in their database to clean out any malicious repositories. This is pretty much the only way to ensure that no malicious repositories exist. So, in order for GitHub to keep malware off of their platform, they need to scan their whole database and either continue scanning it or scan each new commit as it is pushed to the site. This is a massive undertaking and could take a great deal of time, but it is really the only way for them to truly fix the problem entirely.

7.2 GitHub Full Disclosure Requirement

On top of doing a full scan of all their repositories, GitHub should require any repositories that do contain malware to fully disclose all the vulnerabilities the software is made to exploit. Otherwise, that repository should be removed. That way, repositories that exist for research or penetration testing can still be hosted on GitHub, but repositories looking to weaponize malware will be booted off. It would probably even be smart for GitHub to put a special flag at the top of repositories that are deemed to be malicious and used for research purposes so that people looking to download and use them are fully aware of the risks when they do so.

7.3 Scan Before Use

In the end, the developer that is writing the code and choosing to use the outside library is responsible for making sure that the library they use is not malicious. Fortunately, the VT command-line scanning tool we created that takes GitHub URLs and automatically downloads and uploads them for scanning would be a great tool to do this. That way, developers can just run the repositories they plan to use through the system quickly before they add them to their projects.

8 CONCLUSION