Git Smart, Scan Before Use: An Analysis of Malicious GitHub Repositories

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. Git Smart, Scan Before Use: An Analysis of Malicious GitHub Repositories. 1, 1 (December 2019), 12 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

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 fully-functional malware. To resolve this, we built an automated webscraper and an analysis tool to locate and analyze GitHub projects and discover whether or not those projects are malicious.

By seeding our webscraper with a repository known to contain malware, we scanned 5768 repositories and found 252 that were malicious. Most of the repositories found were created either for penetration testing (pentesting) or research, so most of the repositories on GitHub that are meant to be malicious are likely buried deeper and have less attention. There are, without a doubt, more intentionally malicious repositories out there on GitHub, but without a good way to select random repositories and a great deal of time, it is very difficult to discover them. GitHub is currently working on clearing out malicious repositories, but it is a huge undertaking and in doing so they risk stifling important cybersecurity research. Because of this, our findings suggest that the only way to really be sure that a repository is not malicious is to scan it beforehand.

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

3 MOTIVATION AND BACKGROUND

A large number of people use code repository sites such as GitHub to quickly develop code by collaborating with many people. Developers often use third party libraries from GitHub so liberally that they barely even look into them before using them. As third-party library use continues to increase and GitHub remains one of the most popular places to host these libraries, the motivation for hackers to take advantage of more careless users increases. As time goes on, an increasing number of bad actors will take advantage of users' assumptions about code safety and hijack popular projects for malicious purposes, such as embedding a Trojan or a virus in the code[3]. Our guiding questions are:

- Can we find networks of users that contain malicious code?
- What types of users create malicious repositories?
- Can we figure out a way to prevent malicious code distribution?

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 only a single malicious repository located even when including topics such as "malware" and "exploit."

3.2 Disclosure of Malicious Links

In conducting this research, we located many repositories that VirusTotal flagged as malicious. In the spirit of transparent disclosure, we have included links to the flagged repositories and any auxiliary web-pages to allow for independent review and analysis of the documented users of interest.

4 LITERATURE SURVEY

4.1 The Supreme Backdoor Factory [4]

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 [5]

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 locatable, 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.

[3]as an example, see The Supreme Backdoor Factory from the literature review

[4]https://dfir.it/blog/2019/02/26/the-supreme-backdoor-factory/

[5]https://www.ndss-symposium.org/wp-content/uploads/2019/02/ndss2019_04B-3_Meli_paper.pdf

4.3 GitHub Starts Scanning Millions of Projects for Insecure Components[6]

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 BeautifulSoup4, 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 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" column 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

[6]https://nakedsecurity.sophos.com/2017/11/21/github-starts-scanning-millions-of-projects-for-insecure-components/

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 prevents the scanner from being able to successfully download or upload the repository, the file will also be considered an "Error". We then compiled the CSV files from multiple scans and sorted the values to locate failing repositories and explore their methods of dissemination. We manually examined the malicious repositories discovered by the scanner and 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 discover the origin point of the forked repositories. From there, we recorded that user's name and considered that user as a potential "user of interest." We compiled a list of the users who were most commonly referenced and recorded them 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

In our analysis, we discovered a considerable number of repositories that were flagged as failing. In 5768 repository scans, we found 252 malicious repositories, totalling a malicious rate of 4.37%. 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 and are not specifically meant to do harm.

Of the repositories we scanned, 207 returned some sort of error. Likely, the majority of these were simply too large for VT to scan, so they were rejected. We chose to still record the results in the CSV so that it was clear that the scanner at least downloaded each repository and got some sort of result. There were likely a few that just failed to download or caused some sort of error elsewhere in the program, but the vast majority that failed likely did so due to size.

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 vast majority of the malicious repositories we found included some sort of disclosure regarding the code's intended use, often research or penetration testing.

Component files within a repository marked as malicious are naturally part of the repository, so as long as the repository openly declares its contents as malicious, the repository itself should not immediately be considered malicious.

The intentions of the repositories that we have marked as failing are unknown, and determining the actual (unstated) intent of these users remains a challenge for future analysis.

6.2 Users of Interest

Through our analysis, we identified several users of interest through analysis of repositories, forks, and references by other users. They are listed below along with the failing repositories that the scanner detected and their respective percentage detection rates.

6.2.1 RedCanaryCo. RedCanaryCo is the overarching company behind Atomic Red Team, which hosts both the Atomic Red team blog,[7] and their own website[8] which promotes further security resources and talks. Along with other tools, they also include presentations and talks they have previously given in their GitHub repositories. They are a Colorado-based company that has a decent sized product base of security tools for other organizations to buy, and maintains Atomic Red Team and promotes it so that other users can test the security of their systems with these tests.

- **Project: Atomic Red Team** [9]
  Failing rate: 41/58 = 70.6%
  This project is a library of simple security attacks presented for security teams in order for them to test their controls. It is built with the premise of being able to test a wide range of attacks quickly. It is coded in a very wide variety of languages, including C#, Powershell, Java, and Ruby. They tell you how to get started, give you an automated framework, and have tests for Windows, macOS, and Linux. In addition to their website, they also have a link to their Discord to help build the project for more people to contribute. Additionally, they request that people adheres a code of conduct that they developed.

6.2.2 swisskyrepo. Swisskyrepo has a smaller presence of repositories on GitHub, with only eight repositories, but has a large number of followers. All of their projects are security based, and they state that they are a pentester on their user page. They additionally have a Twitter[10] that they use to talk about their repositories and other security and technology topics.

- **Project: PayloadsAllTheThings** [11]
  Failing rate: 39/59 = 66.1%
  PayloadsAllTheThings is a library of payloads and bypasses for web application security. The entire project is developed for pentesting, and how to use each included tool. It supports attacks for both Windows and Linux, along with general networks as well. Also included are a number of common vulnerabilities and exposures (CVE's) such as Heartbleed and Shellshock. Most of the tools are programmed in Python, with a few other languages such as Ruby and ASP being utilized. The project also contains a section that includes a number of books and YouTube videos to reference.

6.2.3 SecWiki. SecWiki [12] is an online Chinese platform dedicated to gathering and providing the most recent news and technology findings in information security and relevant topics. Their GitHub repository [13] contains known exploits and vulnerabilities of operating systems including but not limited to Windows, Linux, Android and MacOS.

- **Project: linux-kernel-exploits** [14]
  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. They propagate their vulnerability findings on their website: https://www.sec-wiki.com/

[7]https://atomicredteam.io/testing
[8]https://redcanary.com/
[9]https://github.com/redcanaryco/atomic-red-team
[10]https://twitter.com/pentest_swissky
[11]https://github.com/swisskyrepo/PayloadsAllTheThings
[12]https://www.sec-wiki.com/index.php
[13]https://github.com/SecWiki
[14]https://github.com/SecWiki/linux-kernel-exploits

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** [15]

  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.

  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. Misterch0c is self-described as "a hacker, not a slacker." Their Twitter was recently banned (within a week of 12/6/2019) and is only accessible via the Wayback Machine.[16] They began developing "Firminator," an open-source firmware decompilation and analysis tool, but development seems to have stalled on the tool. Misterch0c seems to host a couple of major leaks of state-created exploitation tools and malware, so they are a person of interest.

- **Project: APT34** [17]

  Failing rate: 32/60 = 53.33%

  APT34/OILRIG is an Iran-aligned hacking group focused on espionage. In April 2019, three tools were leaked over Telegram and GitHub. [18] Misterch0c hosts at least one of these tools, poisonfrog and the other two directories in the APT34 repository seem to be linked to other exploitation functionalities, specifically tools to update DNS records in the Webmask directory and remote access toolkits in the Webshells directory. The propagation method for this repository is unknown, but this repository is displayed as the repository hosting the APT34 leak as a screenshot in the article referenced above.

- **Project: shadowbroker** [19]

  Failing rate: 17/48 = 35.4% Shadowbroker is a repository that appears to be a compilation of one of the Shadow Brokers[20] dumps of NSA tools and exploits, specifically the one containing the Eternal series of Windows SMB exploits. The repository description specifically notes that certain files were deleted to allow the repository to stay online and links to an archive of the repository that probably contains the removed files. It also links to a post from "@theshadowbrokers" on the Steemit platform[21] that appears to link to an initial cache of leaks

[15]https://github.com/SecWiki/windows-kernel-exploits
[16]https://web.archive.org/web/20191206164109/twitter.com/misterch0c
[17]https://github.com/misterch0c/APT34
[18]https://www.bankinfosecurity.com/leak-exposes-oilrig-apt-groups-tools-a-12397
[19]https://github.com/misterch0c/shadowbroker
[20]https://www.theatlantic.com/technology/archive/2017/05/shadow-brokers/527778/
[21]https://steemit.com/shadowbrokers/@theshadowbrokers/lost-in-translation

via a URL in the post and a password. We did not follow this link, but comments on the Steemit page indicate that the URL worked on time of release. This repository is also linked in the Steemit comments of the initial Shadow Brokers post. This repository maintains a comprehensive list of its tools and exploits, including an as-yet-undetected Command and Control server called OddJob and the Eternal series exploits. This repository was starred 3700 times and has 293 forks. Because of the specific exploits included in this repository (EternalBlue is the exploit WannaCry and NotPetya ransomware leverage to infect vulnerable systems) and the note that the repository was already reported, there is a high chance that this repository is not hosted for entirely benign reasons.

6.2.5 YSRC. YSRC is an acronym that translates to the former TongCheng Safety Emergency Response Center. The repository is maintained by amateur security programmers who has left the organization. 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. At the moment, we are unable to locate a propagation method for YSRC.

- **Project: xunfeng** [22]
  Failing rate: 18/60 = 30%
  Xunfeng is an emergency response system for corporate internet systems. The program runs mainly on Python, and can be installed on either Windows, MacOS and Linux systems. It is also feasible for users to execute Xunfeng on a Docker container. The system essentially provides a quick penetration test for companies to use to analyze their networks systems. Xunfeng is set up to run in advance and periodically runs on the system to check for network vulnerabilities.

6.2.6 grayddq. The user grayddq owns mostly repositories that are used to check or monitor the safety of files, network traffic, as well as other more specific security measures. A number of grayddq's repositories are relevant to the development of Host-based Intrusion Detection System (HIDS), which is able to actively monitor whether a computer system has been infected by an attacker, since attackers will more or less leave a certain amount of traces of their intrusion. Many of the repositories are part of the main HIDS project. [23] Meanwhile, the WeChat ID provided by grayddq - 280495355 - is also reachable by WeChat. The above information about grayddq shows that she is putting effort in publicizing and improving her own security tools rather than uploading a malicious codebase to infiltrate other users' computer systems, yet it does not tell us whether grayddq concealed any functionality in the program.

- **Project: Gscan** [24]
  Failing rate: 22/60 = 36.7%
  Gscan is a scanning tool for Linux systems created by grayddq that is written in both Python 2 and 3, and has been tested on CentOS 6 and 7. The tool includes the security analysis of backdoor vulnerabilities, system configuration, file, network, and system processes. This repository has been marked as malicious possibly due to its attempt to access the system log files and other low level system configuration files in order to check for vulnerabilities. At the same time, the tool also checks for network security. Potential attackers are also able to combine these two features to get hold of the vulnerability information on the system and secretly send it back to the attacker in the background. Further investigation into the Python codebase itself is necessary to identify whether GScan is made solely for its claimed use.

[22] https://github.com/ysrc/xunfeng
[23] https://github.com/grayddq/HIDS
[24] https://github.com/TomANDJarry/Gscan

6.2.7 *k8gege.*

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. K8gege propogates their tools via their blog page.[25]

- **Project: K8tools** [26]
  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.
  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 various other common vulnerabilities.

6.2.8 *PowerShellMafia.*

It is unclear whether or not PowerShellMafia is connected to the Mitre Att&ck program, the creators of PowerSploit. PowerShellMafia also hosts a number of PowerShell-based penetration testing tools. Mitre is a corporation that works in a number of different fields to ensure people's safety and well-being. One of Mitre's main responsibilities is analyzing cybersecurity tactics, and the Att&ck program collects knowledge of cybersecurity attacks and makes their findings available for analysis and protective purposes. PowerSploit is one of the products that has come out of this program.

- **Project: PowerSploit**[27]
  Failing rate: 34/59 = 57.6%
  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, PowerSploit 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. Mitre is propogating their tool on the official Mitre Att&ck website.[28]

6.2.9 *b4rtik.*

B4rtik is active on Twitter as @b4rtik and has a blogpage[29] that they used to introduce RedPeanut and discuss another .NET exploitation method. They describe themselves as a sysadmin and pentester, which may indicate that any malware is designed for testing purposes.

- **Project: RedPeanut**[30]
  Failing rate (original): 14/58 = 24.1%
  Failing rate (RedTeamWing fork): 17/60 = 28.3%
  Failing rate (RunOnceEx fork): 18/59 = 30.5%

[25]https://www.cnblogs.com/k8gege
[26]https://github.com/k8gege/K8tools
[27]https://github.com/PowerShellMafia/PowerSploit
[28]https://attack.mitre.org/software/S0194/
[29]https://b4rtik.github.io/
[30]https://github.com/b4rtik/RedPeanut

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. RedPeanut is also listed as one of the top RATs on a software ranking website.[31]

6.2.10 cobbr. Cobbr developed two projects flagged as malware: Covenant and SharpSploit. Cobbr is a security researcher for a red teaming group and is actively developing these and other projects for use in red teaming. He is active on Twitter as @cobbr_io and maintains a blog page, posting about .NET and C# penetration testing techniques.[32]

- **Project: Covenant**[33]
  Failing rate: 25/59 = 42.3%
  Covenant was directly referenced by language in RedPeanut's Readme.md file. It is "Covenant is a collaborative .NET C2 framework for red teamers." This software is designed to interface with .NET connections on a network (from RATs, for example, in conjunction with RedPeanut instances) and is written entirely in C#. Covenant is multi-user and dynamically compiles any payloads used at runtime through obfuscation, ensuring that no static payloads are used.
- **Project: SharpSploit**[34]
  Failing Rate: 32/60 = 53.3%
  SharpSploit is a post-exploitation library designed to run on the .NET framework designed entirely in C#. It was inspired by PowerSploit and was written as part of what cobbr describes as a larger shift from PowerShell to C# in order to target .NET libraries, as PowerShell is becoming more secure and harder to exploit on target systems. SharpSploit contains four major classes of commands, "Credentials", "Enumeration", "Execution", and "Lateral Movement" designed for use in post-exploitation testing.[35] The Credentials section specifically references Mimikatz, a piece of testing software designed to steal Windows credentials.[36]

6.2.11 aainz. Aainz developed the TinyNuke botnet, a "Zeus-like banking Trojan" that was released as a nearly-functional program. A McAfee researcher isolated some discussions on hacker forums at the time of release where aainz said that he "wanted to stop [other hackers] using crap"[sic][37] in regard to the quality of botnets in use at the time of TinyNuke's release. Aainz later removed the source code of TinyNuke from GitHub, stipulating that access to the malware would only be allowed for research purposes.

- **Project: TinyNuke**[38]

[31] https://libs.garden/csharp/search?q=rat&sort=popular
[32] http://cobbr.io
[33] https://github.com/cobbr/Covenant
[34] https://github.com/cobbr/SharpSploit
[35] https://cobbr.io/SharpSploit.html
[36] https://www.varonis.com/blog/what-is-mimikatz/
[37] https://securingtomorrow.mcafee.com/blogs/enterprise/tinynuke-may-ticking-time-bomb/
[38] 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 TinyNuke is a relatively minor project and player, it is a definitively malicious codebase. Leaving the code open-source was determined to be too significant of a threat by aainz, so he took it down. TinyNuke, however, is, still hosted by other GitHub users (as per the failing repository above) with a note that the source repository was deleted.

7 COUNTERMEASURES

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

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 continually scan their whole database 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 removed from the site. 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 CONCLUSIONS

In general, the only way to be sure not to use malicious libraries on GitHub is to scan them before use. Our initial repository scans, which scanned over 2,000 trending repositories, found only one malicious repository. This shows that nearly all of the trending repositories on GitHub are clean. It makes sense that they would generally be clean, because in order to get to that point, thousands of people had to look at or use the code. However, malicious repositories almost certainly do still exist on GitHub. As our literature reviews found, rings of malicious accounts still exist, and while GitHub is working to clear them out, it is still important to be wary. These repositories are just buried deeper in the database and are more difficult to find. In reality, attackers would probably prefer that their projects stay under the radar, reaching just enough people to make their tactics profitable.

In future research, there are a couple different strategies that could yield much more interesting results. For one, it might be better to find a way of randomly selecting a "seed" repository and branching out to find more repositories from there, or finding a way to build a list of repositories that is completely random, rather than being based on trending repositories or a seed repository. That way, we would get a much more representative cross-section of GitHub, and we would increase our odds of finding black hat malicious code. Malicious repositories would still be difficult to find, but the randomness would almost certainly improve our odds somewhat. Once we had that list, we could scan through it to find which repositories were research and penetration testing and which had the potential to be truly malicious, and we could run a new "seeded" scan to find all the repositories they are connected to. Potentially, this could expose particular rings of malicious users. More likely though, malicious GitHub users will avoid being connected to too many people in order to prevent suspicion or association if one user gets caught. This future research would require a great deal of time, but could yield some incredibly interesting results.

Nearly every malicious repository we were able to find was either created for research use or penetration testing. In over 252 malicious repositories found, to our knowledge, not one of them was truly created to exploit its users. The repositories we found ranged from company penetration testing products to university malware research studies. The GitHub users hosting them were all either white hat hackers or were developers or companies promoting penetration testing software. Each of them either exposed vulnerabilities of specific programs or were designed to find known vulnerabilities for other programs. Nearly every malicious repository that we were able to find and investigate had disclaimers of some sort explaining that they were malicious and that they were only to be used legally to test code or authorized systems. Our findings show that GitHub has a strong circle of research and penetration testing users who are working to improve cybersecurity rather than weaken it. While any one of these open-source malicious repositories has the potential to be weaponized, the ability to share knowledge, make penetration testing tools readily available, and research vulnerabilities as proofs-of-concept is far more valuable than the risks of exposing known malware to hackers.

However, GitHub should be responsible for making it absolutely clear to its users which repositories contain malicious code. They should scan through every repository on their platform and tag every page that contains malicious code with a special malware tag that is easily visible to users. This won't hurt security researchers or penetration testers at all because the people using their software should already know that it is malicious. This could, however, save less security-aware users from downloading a "backdoored" application or using a malicious library. Next, GitHub should require any repository they find to be malicious to apply as a "malicious for research" repository. That way, they can screen malicious repositories to make sure they truly are research and not pretending to be benign. Once this policy is in place, GitHub should analyze each commit to determine if it contains malicious code. If so, GitHub should require the repository to apply to be malware.

Ultimately though, GitHub users will have to take some of the responsibility for the software they use. In reality, scanning a repository before use is the only surefire way to prevent the use of malicious repositories. GitHub should take whatever anti-malware scanning tool they develop and make it publicly available to run online against any repository. That way, all GitHub users will have immediate access to a tool like ours that can scan a single repository quickly. Developer, who should already understand the risks of using third-party libraries, should be encouraged to use libraries they find on GitHub responsibly and scan them before integration into their projects. In the end, although nearly all of the malware we found was for research purposes, both GitHub and its users should be more cautious about propagating malicious software.