Windows Malware Binaries in C/C++ GitHub Repositories: Prevalence and Lessons Learned

William La Cholter, Matthew Elder and Antonius Stalick
Applied Physics Laboratory, Johns Hopkins University, U.S.A.

Keywords: Malware, GitHub, Open Source Software, Windows.

Abstract: Does malware lurking in GitHub pose a threat? GitHub is the most popular open source software website, having 188 million repositories. GitHub hosts malware-related projects for research and educational purposes and has also been used by malware to attack users. In this paper, we explore the prevalence of unencrypted, uncompressed binary code malware in Microsoft Windows compatible C and C++ GitHub repositories and characterize the threat. We mined 1,835 repositories for already-compiled malicious files and data suggesting whether the repository is malware-related. We focused on these repositories because Windows is frequently targeted by malware written in C or C++. These repositories are good resources for attackers and could target Windows users. We extracted all Portable Executable (PE) files from all commits and queried the malware resource VirusTotal for analysis from its 76 anti-virus engines. Of the 24,395 files, 4,335 are suspicious, with at least one detection; 440 could be considered malicious, with at least seven detections. We identify topic tags suggesting malware or offensive security content, to differentiate from seemingly benign repositories. 197 of 440 malicious executables were in 27 ostensibly benign repositories. This work illustrates risks in source code repositories and lessons learned in relating GitHub and VirusTotal data.

1 INTRODUCTION

GitHub is the most popular open source software website, with over 188 million repositories (GitHub.com, 2020a). GitHub is known to host malware-related projects for research and educational purposes—described as allowable in their "GitHub Community Guidelines" (GitHub.com, 2020c)—including source code examples of exploitation and generally nefarious functionality, such as keyboard logging. GitHub originally became popular as a service to host software source code repositories but has also become a popular hosting environment for non-source code information, such as raw data sets, including curated malware collections such as theZoo (ytisf, 2020). GitHub has also been used by malware for command and control, download infrastructure, or serving backdoored code (Avast Threat Intelligence Team, 2018), (Munoz, 2020). Given that malware resides on GitHub both legitimately and maliciously, we study whether malware lurking in GitHub repositories poses a threat to repository users and downstream consumers of these repositories.

Malware is a huge cybersecurity problem, with over 350,000 new malicious programs and potentially unwanted applications discovered every day (AV-Test, 2020). Malware developers target many platforms (e.g., desktop, mobile, servers, cloud, and deep learning systems), use many different programming languages (e.g., C, C++, Java, JavaScript, Assembly, Python, Ruby, C#, and Delphi), and produce many different forms of malware (e.g., Windows Portable Executable (PE), Linux Executable and Linkable Format (ELF), shell code injection, database injection, and raw malicious data). For this malware research, we focused on Windows Intel x86 binary files written in C and C++ because of their volume, reach, complexity, and potential for uniform analysis methods. It is therefore natural that our research started with ostensibly Windows C and C++ repositories.

In July 2019, we found 1,870 GitHub repositories using the search terms of "windows" and "c" or "cpp." Of those, 1,862 have source code that could be built using a modern Windows C++ compiler, and 1,835 were still online when we checked again in December 2019. Some related web UI searches, such as for Microsoft Visual C++ project files (.vcxproj), yielded repositories outside of this initial set. Additionally, keywords mined from these repositories suggest more repositories of interest beyond our search terms. Expanding the data set is future work.

Cholter, W., Elder, M. and Stalick, A.
Windows Malware Binaries in C/C++ GitHub Repositories: Prevalence and Lessons Learned.
DOI: 10.5220/0010237904750484
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 475-484
ISBN: 978-989-758-491-6

To determine whether a file might be malicious, we searched the VirusTotal malware information service that aggregates the detection results of 76 antivirus (AV) products (VirusTotal, 2020b). Any registered user can submit a sample to VirusTotal for analysis. The detection results and other file information are available to anyone for subsequent query, by submitting a cryptographic hash of the file. VirusTotal's Application Programming Interface (API) includes rescan requests for results from the most up-to-date AV products and much threat intelligence data related to malware (VirusTotal, 2020a).

The contribution of this paper is a methodology for investigating the presence of malware over all the commits in the lifetime of a GitHub repository. While it is straightforward to clone a repository to a specific point in time - e.g., the current head state or some arbitrary branch in the past - our approach investigates all of the commits throughout the history of the repository to identify files for analysis. We use the well-established method of VirusTotal anti-virus engine results to assess maliciousness of a particular file type (Windows portable executable binaries), and we apply our methodology to a subset of GitHub repositories (Windows C and C++ repositories) in this preliminary investigation. However, this methodology could be applied to additional populations of GitHub repositories, identifying other file types of interest through the repository lifetimes, and using other malware analysis methods.

In this paper, we present our preliminary investigation into the presence of malware files in Windows C/C++ GitHub repositories. Section 2 provides background on GitHub and related work in VirusTotal malware research. We describe our approach to mine Windows binary files from GitHub and then query VirusTotal for malware detection results in Section 3. Section 4 presents our initial VirusTotal analysis results for the Windows files that we mined from our GitHub repositories of interest. Section 5 provides a discussion and more detailed analysis of our results. We present our conclusions and directions for future research in Section 6.

2 BACKGROUND AND RELATED WORK

GitHub is known to host malware, both legitimately (i.e., in compliance with GitHub's terms of use) and illegitimately. GitHub prohibits content that "contains or installs any active malware or exploits, or uses our platform for exploit delivery" (GitHub.com, 2020b). An example of GitHub hosting malware in violation of this policy occurred in March 2018, when cybercriminals uploaded cryptocurrency mining malware to forked GitHub projects and used phishing ads to download and execute the malware (Avast Threat Intelligence Team, 2018). More recently, 26 open source projects were discovered to have backdoors inserted by the Octopus malware, which used the build process to spread to other NetBeans projects (Munoz, 2020). GitHub appears to allow executable malware in curated malware collections. A search for "malware samples" returns over 250 repositories. Although many repository descriptions suggest analysis tools or malware-related resources, some explicitly indicate that they include malware samples.

In terms of detecting malware or malicious repositories in GitHub, only recently have two efforts systematically studied this problem. Recent work by Rokon et al. developed a methodology for finding malware source code within GitHub projects and identified 7,504 malware source repositories (Rokon et al., 2020). While the findings from this work can be used to search for malware binaries in GitHub as well, our work seeks to find malicious binaries in GitHub repositories that are not necessarily purporting to contain malware. Zhang et al. developed a deep neural network approach to detect malicious GitHub repositories using content-based features from source code files, investigating a population of blockchain and crytocurrency repositories (Zhang et al., 2020). They used VirusTotal as part of their evaluation process for comparison purposes, ultimately labeling 1,492 repositories as malicious out of their population of 3,729 repositories, but again this work was more focused on malicious source code in GitHub.

Many previous research efforts have used VirusTotal to support malware detection and analysis in the domains of malware binaries run in dynamic analysis sandboxes (Graziano et al., 2015), signed malware binaries (Kim et al., 2018), and mobile applications (Hurier et al., 2017), (Pendlebury et al., 2019), (Salem et al., 2019), (Suciu et al., 2018), (Wang et al., 2019). VirusTotal can also be used for analysis of malicious web addresses, i.e., Uniform Resource Locators (URLs), such as those used in phishing campaigns (Peng et al., 2019). These research efforts and others each utilize VirusTotal in different ways, either using various thresholds for the number of VirusTotal engines needed to consider a sample as malicious (e.g., 1, 5, or 10), thresholds based on percentage of engines (e.g., 50%), or results from a subset of engines based on high reputation or market share. In short, there is little consensus on how to definitively interpret VirusTotal results to determine whether a sample is malicious.

Recently, Zhu et al. published a study on the behavior of the anti-virus engines within VirusTotal, which included a survey of 115 academic papers that used VirusTotal (Zhu et al., 2020). The most common approach to using VirusTotal was to set the threshold at one malicious engine detection for labeling a sample as malware (50 out of 115 papers). However, one key finding of their research was that the engines within VirusTotal "flip" detection results over time, sometimes oscillating between malicious and benign labels for the same sample over short periods of time. The authors recommended setting the threshold somewhere between 2 and 39 for stability of engine labels. Zhu et al. also found that the detection results from certain engines are highly correlated, which affects how one should set a threshold, with the largest cluster containing six engines using a hierarchical clustering algorithm with a threshold of 0.001 (Zhu et al., 2020).

3 APPROACH

We used GitHub to find and clone repositories, with the intent of compiling the code for binary analysis and getting data and metadata that provide insights into the software development process. In the course of that work, we discovered the presence of suspicious files and a paucity of rigorous research on them. We cloned all of our repositories of interest 9-July-2019. By picking a specific date, we eliminated the need to account for the variable of time in our analysis of GitHub data. Git repositories provide core ground truth through SHA-1 cryptographic hashes of files, commits (file versions, predecessors, and comments), and tags. GitHub provides ground truth of user-provided data and approval of commits by the repository maintainer. We performed as much mining as possible on local copies to avoid API limits.

3.1 Mining GitHub and Git

GitHub and Git present data management challenges: GitHub provides additional online context for the potentially offline Git commit activities, but it provides snapshot or event-driven data rather than historical information through its API. For example, to find candidate repositories, we used GitHub's GraphQL API, querying for languages "c" and "cpp" and the "windows" topic and cloned them locally. However, those topics associations can change over time.

To be thorough in analyzing all commits throughout a repository's history, it is necessary to scan all files ("blobs") in Git's local key-value store. It is very efficient, $O(b)$ for $b = |Blobs|$, to sweep the database for all file content that ever was in the history of commits and tags. But it is impossible to establish where and when they are referenced without walking the commit tree and tag graphs, naively $O(t \cdot c)$ for $t = |Blobs \cup TreeItemLists|$ and $c = |Commits \cup Tags|$.

We used pygit2, a wrapper of libgit2, which was anecdotally an order of magnitude faster than GitPython during our early prototyping. We used python-magic content type identification, which wraps libmagic. Because Git SHA-1 hashes are computed on file contents and additional metadata, we needed to compute pure cryptographic hashes for VirusTotal submission and used Python's hashlib, written in C. By performing operations in-memory using underlying C, performance was strong and we did not change repository file system state. For cross-repository analysis and structured ad hoc data we used PostgreSQL relations and JSON columns.

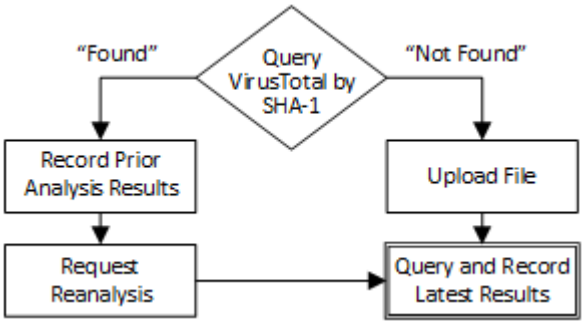

Figure 1: VirusTotal Query Flowchart.

3.2 Querying VirusTotal

VirusTotal supports queries by MD5, SHA-1, and SHA-256 cryptographic hashes. Although SHA-1 is generally deprecated because of collisions, it is fast and sufficient for file identification.

Figure 1 shows the flowchart of our VirusTotal query process. We started by querying VirusTotal using the file content's SHA-1 hash. If VirusTotal has previously received and analyzed the file, it returns JSON results that include the last analysis from its AV engines, labeled "prior" analysis in our results. That analysis could have occurred years ago, depending on the file's age, when it was first submitted, and when it was last analyzed. We saved those "prior" results to characterize the initial results and subsequent analysis. VirusTotal AV detections generally improve over time, as vendors improve algorithms and signatures, and as VirusTotal adds new engines. To establish results across contemporary engines, we requested reanalysis. We also uploaded all files that VirusTotal has not previously received and then queried those de-

Table 1: VirusTotal Detection Results - Suspicious Files, Previously Scanned and Unseen.

Binary Code Files	# Samples	# Prior Hits	# Latest Hits
Previously scanned by VT	10,413	1,353	1,090
Previously unseen by VT	13,982	N/A	3,245
Total	24,395	1,353	4,335

Table 2: VirusTotal Detection Results - Malicious Files, Previously Scanned and Unseen.

Binary Code Files	# Samples	# Prior Hits	# Latest Hits
Previously scanned by VT	10,413	226	240
Previously unseen by VT	13,982	N/A	200
Total	24,395	226	440

tection results. We downloaded "latest" results from 24-December-2019 to 7-January-2020.

VirusTotal provides four core file-related AV request APIs for non-premium users: the most recent scan results of a file, the request to rescan a file, the results of the request to scan a file, and the results from a specific non-public request identifier. The commercial/premium API service also offers users the ability to query the list of non-public request identifiers, necessary to obtain results from arbitrary past requests.

3.3 Threats to Validity

VirusTotal introduces inherent variability of results that challenge reproducibility: the accuracy of any given AV engine scan; the variability of available engines in VirusTotal at any given time; the success of individual engines processing the sample in a VirusTotal-managed processing window, the results from specific engines over time; the opacity, consistency, and provenance of details in reports; and the ability to obtain the most recent results without obtaining a paid premium account. It is not controversial to say that a given AV engine scanning a given file at a given time may report false positive or false negative results. We do not consider that a threat to our experiment's validity because of the well-understood caveats one may apply to an interpretation of AV results. In this research, the main threat is that data capture is not instantaneous and that the same file could garner different results at the beginning and end of a capture window.

We captured data in a two-week period, December 2019 – January 2020 to minimize the period of time that a change could have occurred. We provide results for any Windows binary that has at least one AV engine detection of "malicious", which indicates that the sample is "suspicious." We also report results using a threshold of seven AV engine detections of "malicious", based on recommendations and interpretation of the recent Zhu paper. Given the finding that certain engines' detection results are highly correlated, and the largest cluster consisted of six engines, a threshold of seven ensures that at least two independent engines are indicating "malicious."

4 RESULTS

In this section we present the VirusTotal detection results for the Windows binaries extracted from our 1,835 GitHub repositories of interest. We built a data set of 24,395 unique binary code files, mining all commits from all 1,835 GitHub repositories of interest. A file was included if its MIME type was "executable." (One 171 MB file was excluded because we were unable to upload it to VirusTotal.) The first subsection presents the results for the data set as a whole, and the second subsection provides results based on repository characteristics.

4.1 VirusTotal Results

Table 1 shows the results of VirusTotal scans for new and previously uploaded binary files when setting the threshold to at least one malicious detection, indicating that a file is "suspicious." Of the 24,395 files, 10,413 had been submitted previously, indicated by "Previously scanned by VT"; 1,353 of those had at least one malicious detection at the time of prior analysis in VirusTotal, labeled "# prior hits." When we requested reanalysis for these files, 1,090 files had at least one malicious engine detection, showing that detections decreased overall on rescan. Of the 13,982 files "Previously unseen by VT" that we uploaded for analysis, 3,245 had a malicious detection.

Table 2 shows the results of VirusTotal scans for new and previously uploaded binary files when at least seven engines provide a malicious detection, our threshold to determine that a file is "malicious." Setting the detection threshold higher results in far fewer hits, of course: only 440 out of the 24,395

Table 3: VirusTotal Detection Results - Suspicious Files, Previous Scan and Rescan Results.

Binary Code Files	# Samples	Detected	Not Detected
Previously submitted to VT	10,413	1,353	9,060
Resubmitted to VT	10,413	1,090	9,323

Table 4: VirusTotal Detection Change in Results - Suspicious Files.

Originally Benign	
# samples	9,060
# that became suspicious	289
% that became suspicious	3%
# AV engines	1 - 69

Originally Suspicious	
# samples	1,353
# that became benign	552
% that became benign	41%
# AV engines	1 - 3

have at least seven AV engines indicating malicious detections. Of the 10,413 files previously scanned by VirusTotal, 226 previously exceeded our malicious detection threshold and 240 are currently deemed malicious in the latest results. Of the 13,982 files previously unseen by VirusTotal, 200 are deemed malicious in the latest results.

Both tables of VirusTotal detection results demonstrate the change in engine detections over time. To highlight these changes in more detail for the suspicious file results (i.e., those with at least one malicious detection), Table 3 shows that some previously benign-seeming files were considered suspicious–and vice-versa–in the reports that we requested in the December 2019 – January 2020 timeframe. The overall decrease of 263 files—from 1,353 to 1,090—having at least one malicious detection is the net result of 289 files being detected as malicious that were not previously and 552 files previously being detected as malicious no longer having any AV engine detections.

Table 4 shows the relative change in results for the suspicious files. The substantial re-characterization of files as having detections vs. not having detections coincides with a relatively small number of initial positives results, with 1 to 3 AV engines previously indicating malicious. On the other hand, files only later getting malicious detections have a much larger range of 1 to 69 detecting engines.

Table 5 shows the breakdown of files within different categories of Windows executable binaries. The vast majority of binary code files are targeted to run on modern 32- or 64-bit Windows versions. There are also files targeting DOS and 16-bit Windows in the "Pre-Win32" category, which are ostensibly compatible with Windows. Finally, there are incompatible ELF and boot image files in the "Other" category (presumably misclassified by libmagic). As seen in the second column of Table 5, 4,280 Windows-compatible files were suspicious and 418 were malicious. Except for "Other" files, any standalone executable file poses an immediate risk to a repository user who runs it, while a dynamically linked library (DLL) on modern Windows poses a risk of incorporation into the repository's build outputs or execution as a system service or code injected into a process on a build host. Table 5 shows that of the 4,280 suspicious files, 1,074 are DLLs and 3,206 are standalone executable files. For the 418 malicious files, 28 are DLLs and 390 are standalone executable files.

Table 6 presents the number of files in weighted bins by the number of engines indicating "malicious." This shows the range of hits and the large proportion of samples with low hit counts.

The results above for all files represent the aggregate across all commits over the lifetime of the repository. For results at a single point in time, we also analyzed the files that were accessible from the head of the repository. A repository's head commit—the files accessible after cloning and updates—represents a public view of the repository at the time of cloning and analysis. Across all 1,835 of our repositories of interest, there are 7,772 unique binary files in the head commits on 9-July-2019, of which 939 were suspicious with at least one AV detection in VirusTotal, and 204 were malicious with at least seven AV detections. 5,512 files were already analyzed by VirusTotal, while 2,260 had to be uploaded for analysis.

4.2 Repository-based Results

Of the 1,835 repositories queried, 593 repositories contain binary files. 314 have at least one suspicious binary file, which is a significant subset. 52 repositories have at least one malicious binary with seven or more VirusTotal AV engine detections.

We examined the concentration of suspicious binaries across repositories, presented in Table 7. Of the 314 repositories having suspicious files, a majority, 182 repositories, have one (1) or two (2) suspicious files. Across the population, the mean file count is 7.03 and standard deviation is 20.67. Similarly, Table 8 presents the distribution of malicious file counts across the 52 repositories with malicious binaries and shows that most only have one or two.

Table 5: VirusTotal Detection Results – By File Type.

	All	Win 32/64	DLLs	EXEs	Pre-Win32	Other
Benign	20,060	19,385	12,331	7,054	431	244
Suspicious	4,335	4,280	1,074	3,206	14	41
Malicious	440	418	28	390	2	20
Total	24,395	23,665	13,405	10,260	445	285

Table 6: VirusTotal Hit Counts in Weighted Bins.

Hit Count	1	2	3	4	5	6	7-10	11-20	21-30	31-40	41-50	51-60	61+
# samples	2,491	722	298	161	135	88	100	92	92	71	59	20	6

Table 7: Suspicious File Count by Repository Count.

# suspicious files	1	2	3	4	5	6-10	11-617
# repos	131	54	20	24	16	22	47

Table 8: Malicious File Count by Repository Count.

# malicious files	1	2	3	4	5	6-10	11-617
# repos	20	10	5	3	2	1	8

Table 9: Top 10 Repositories by Files Having VirusTotal Detection - Suspicious Files.

Repository Name	**# Detected**	**# Binaries**	$\overline{\textbf{score}}$
papyrussolution/OpenPapyrus	617	1,259	1.19
lhmouse/mcfgthread	507	1,175	1.45
ffftp/ffftp	305	1,061	1.13
processhacker/processhacker	220	282	4.10
arjunae/myScite	205	1,762	1.99
RomaniukVadim/hack_scripts	198	313	21.45
arizvisa/windows-binary-tools	166	924	2.34
tomdaley92/kiwi-8	116	186	5.27
Twilight-Dream-Of-Magic/BackDoorProgram-EncryptOrDecryptFile	113	160	2.74
alexfru/SmallerC	109	300	20.79

Table 9 shows the top ten repositories by number of suspicious files and the mean score of those detections. The second column of Table 9 provides the number of overall binary files in these repositories for additional context, indicating how prevalent binary files are in each of these repositories and the ratio of suspicious binaries.

To assess the stated purpose of each GitHub repository, we extracted the user-provided repository tags and found 1,802 unique tags across the 1,835 repositories. We classified 70 tags as potentially related to malware or other offensive security topics. Each author identified candidate tags, and those receiving a majority of votes were selected. Our malware-related tags have overlap with the Malware Attribution Enumeration and Characterization (MAEC) structured language for malware information sharing (The Mitre Corporation, 2017), allowing for fuzzy matching and semantic equivalence. It is important to note that MAEC is a prescriptive taxonomy for documenting security incidents and not a computer security natural language topic model. There are other efforts towards defining cyber security ontologies (Syed et al., 2016), which could contribute to a characterization of malware-related purposes. This is an area of future exploration.

Of the 314 repositories that contain at least one suspicious binary, only 50 have at least one malware or offensive security-related tag. This leaves 259 repositories with suspicious/malicious binaries where users might not expect that risk. Of the 52 repositories that contain at least one malicious binary, 25 have at least one malware or offensive security-related tag. The 27 repositories not tagged as being related to malware or offensive security contain 197 malicious binaries, representing risk to unsuspecting repository users.

Table 10: Examples of Varying Scan Results over Time.

	Example 1	Example 2	Example 3	Example 4
Our Scan Requests				
scan date	12/24/2019	12/24/2019	12/24/2019	12/24/2019
# engines	74	73	75	75
# malicious	0	43	12	2
Previous Scans				
last analysis date	12/10/2015	9/29/2019	11/26/2019	2/1/2017
# engines	52	71	71	58
# malicious	0	43	11	0
Earlier Activity				
last modification date	1/8/2019	9/29/2019	12/4/2019	2/1/2017
first submission date	9/20/2013	5/12/2016	5/7/2011	1/4/2017
Submitter or Author-Reported Data				
PE file ”creation date”	9/11/2013	5/8/2016	5/7/2011	7/28/2014
”first seen itw date”	9/11/2013	5/8/2016	11/20/2010	12/31/2097

5 DISCUSSION

5.1 Risks Posed by Unhygienic Repositories

Without even considering the risk of malicious content, binary files in repositories should raise concerns. It is almost always a bad practice to store build outputs in any repository because they increase the repository size, are not amenable to editing or comparisons across versions, and may be accidentally updated when the repository is built–especially Windows PE files, which contain the build timestamp.

Including binaries, such as libraries, as build inputs or runtime dependencies violates the spirit of open source development. It may be unavoidable for a repository owner seeking to baseline specific build inputs while holding a software license that allows redistribution of binaries. In most cases, however, GitHub repository maintainers should provide prebuilt software in GitHub release bundles, outside the Git repositories.

The virus research community has adopted safe handling procedures, including packaging malware in encrypted archives (Zeltser, 2020), and sharing samples only after vetting interested researchers. Repositories that violate these rules expose non-malware research environments. Indeed, when we cloned repositories from our Linux environment onto a Windows server, we set off over 100 alerts in our enterprise AV sensors—and that was only in the file system copies from the head branches. Many malicious binaries lay dormant and unscanned while they rest in Git’s custom storage formats, likely unsupported for scanning by AV engines.

Finally, build files such as Makefiles, .vcxproj files, and continuous integration orchestration files are essentially executable scripts, which pose the risk that building a project can compromise a system. Non-malware repository researchers would also benefit from safe handling, such as processing as much as possible on less-targeted OSs and with repositories that are bare or mirrors without local file copies.

5.2 Not All Windows Malware Is in PE Files

Malware comes in many forms. We looked for binary files, but these repositories may have malware in other formats, such as documents and scripts. It is worth noting that in scanning repository head commits, we identified 761 archive files (WinZip, 7-Zip, and RAR), 33 of which are or could be encrypted. Perhaps the 33 represent responsibly encrypted malware samples. There are other forms of malware that we could mine from GitHub repositories beyond Windows binaries, such as Linux malware, mobile malware, malicious scripts, and malicious PDF documents.

5.3 Git-related Observations

In the course of this research we used many interfaces to Git-related data. While not necessarily critical to this immediate work, our experience provides some insights for future researchers. Online APIs such as GitHub REST v3, GitHub GraphQL,

GH Archive (gharchive.com, 2020), and Google BigQuery (Google Cloud, 2020a) are powerful for high-level data, but for compute-intensive file analysis, local execution may be the only option. While Git is the primary source for commit history and files, its data model is optimized for efficiency and extensibility of end-user file-based operations. The researcher is left to develop a new data model to manage the federation of Git and online APIs.

GitHub provides a rich online community and source of data, but does not provide direct temporal control over results comparable to the cryptographically stable Git commit log, which admittedly is coming under attack because of SHA-1's emerging weaknesses to hash collisions. So, while it may be straightforward to time-box commits up to a certain date, finding the GitHub topic associations at that date requires forethought to query all GitHub information, sifting through events from the beginning of the repository to that point in time (or in reverse from the present time), or queries using third-party services such as GH Archive and Google BigQuery. GitHub's 5,000 REST requests or GraphQL 5,000 points per hour (GitHub.com, 2020d) and BigQuery's 1 TB free per month API (Google Cloud, 2020b) quotas require considerable planning and data acquisition design, and therefore we attempted to maximize local analysis with Git. Moreover, a local checkout of Git provides groundtruth for what a developer would see from cloning the repository.

5.4 VirusTotal Observations

As previous research has shown (Zhu et al., 2020), (Pendlebury et al., 2019), (Peng et al., 2019), (Salem et al., 2019), VirusTotal engine data is subtle: results change based on when a query is run, and the non-premium API provides only the most recent results based on the time of the last requested scan, which could have been any arbitrary point in time in the past. It is possible that one or more engines within VirusTotal could provide a false positive detection for a file. VirusTotal's AV engines change over time and the results from the engines can change based on AV engine implementation and signature updates. While it may be tempting to use VirusTotal as a form of oracle for malware detection, there is no universally accepted threshold for the number of AV engines in VirusTotal that "guarantees" a file is malicious.

There are at least three interesting points in the lifetime of a file analyzed by VirusTotal: (1) initial analysis at the time of first submission to VirusTotal; (2) "prior" analysis relative to the current experimentation time, which will be whenever the file last had an analysis requested; and (3) "latest" analysis, requested at the current experimentation time. Tables 1 and 2 previously presented the change in detection results between points (2) and (3). Table 10 illustrates with four example files that metrics based on these points in time can be inconsistent within a small time window, across larger windows, and fabricated.

For example, before our rescan request ("Our Scan Results" in Table 10), the results for one "benign" binary named "curl.exe" (Example 1) were originally created when scanned on 20-September-2013, updated with scan results from 52 engines on 10-December-2015, and modified on 8-January-2019. Other dates in a report, such as first seen in the wild (the year 2097 in Example 4 in Table 10), and of course, the PE header timestamp have no assurance because they are subject to spoofing by the submitter or binary author (sometimes the same individual! (Zetter, 2014)).

Across all of our rescan requests started on 24-December-2019, we received results from 46 to 76 engines, with a mean of 73.4 engines and standard deviation of 1.21.

The VirusTotal terms of service do not allow sharing full reports that would reveal AV vendor capabilities. Therefore, experiments relying on precise scan details are not reproducible and the data cannot be broadly shared. One researcher could affect an unrelated researcher's work by requesting a rescan at a non-deterministic time during overall data capture, a significant risk with a public API rate limited to four per minute and with a potential for three requests for a single sample. Indeed, the footprints of our queries are all over the data. It is also possible that the footprints from the authors' IT department can be observed in the data, as the authors were contacted by them in the course of cloning repositories to explore build experimentation.

It is possible to get all scan history for a sample, by purchasing the premium service—but those results indicate which scans were requested, not whether a given file might have been considered malicious at a particular point in time, if only someone had requested a scan at that time. For example, it is infeasible to perform a post-mortem of an attack by asking, "Could all of the files in an intrusion have been identified as malware on 1-June-2015?" Although VirusTotal adds a very different dimension of data to software repository research, it does not offer the temporal control required in many studies and experiments.

6 CONCLUSIONS AND FUTURE WORK

Does the malware lurking in GitHub pose a threat? Yes, we found 4,335 suspicious Windows binary files with at least one malicious AV detections in VirusTotal across 314 of 1,835 repositories studied. We found 440 malicious binaries with at least seven AV detections across 52 repositories. Just as some researchers found hidden API keys in repositories (Meli et al., 2019), we found hidden malicious content, not easily queried because of the number of files and repositories, the cost of querying online services, and changing malware scan results. Users and researchers should be careful when downloading open source repositories, because it is difficult to be sure that the content is safe, especially binary content. Repository owners should be vigilant given their role in the open source software supply chain. We have submitted the hashes and repository URLs to GitHub, out of an abundance of due care in exercising responsible disclosure.

This study mined a particular slice of GitHub for malicious Windows binaries—we could obviously expand the population of GitHub repositories, beyond those tagged as Windows and C or C++, and expand the types of malware investigated. The substantial observed swing in VirusTotal results over time motivates more study to identify the controlling variables and ultimately to achieve a better understanding of how to assess confidence in a particular scan.

GitHub is a convenient platform for hosting source code and other user-provided content. GitHub users hosting malware should, at a minimum, apply basic safety measures, such as storing malware in encrypted archives (Zeltser, 2020). More troubling, though, is that the mere presence of binary content in a source code repository suggests a violation of best practices—mining the repository history can provide insights into a project's overall quality and maturity. The accidental presence of malicious binary content suggests a violation of trust—mining the contributors' history might provide insights into the kinds of people unwittingly compromised. The intentional and surreptitious insertion of malicious binary content is an attack on trust—mining the entire repository history might help identify future targets and enable attribution of the those willfully corrupting the open source software supply chain.

ACKNOWLEDGEMENTS

This work was funded by the Minerva Research Initiative and is sponsored by the Department of the Navy, Office of Naval Research under ONR award number N00014-18-1-2111. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

REFERENCES

AV-Test (2020). Malware statistics & trends report |av-test. https://www.av-test.org/en/statistics/malware/.

Avast Threat Intelligence Team (2018). Greedy cybercriminals host malware on github. https://blog.avast.com/greedy-cybercriminals-host-malware-on-github.

gharchive.com (2020). Gh archive. https://gharchive.org.

GitHub.com (2020a). Code search - github. https://github.com/search?q=&ref=simplesearch.

GitHub.com (2020b). Github acceptable use policies. https://help.github.com/en/github/site-policy/github-acceptable-use-policies.

GitHub.com (2020c). Github community guidelines. https://help.github.com/en/github/site-policy/github-community-guidelines.

GitHub.com (2020d). Graphql resource limitations | GitHub Developer Guide. https://developer.github.com/v4/guides/resource-limitations/.

Google Cloud (2020a). Bigquery: Cloud data warehouse — google cloud. https://cloud.google.com/bigquery.

Google Cloud (2020b). Estimating storage and query costs | BigQuery | Google Cloud. https://cloud.google.com/bigquery/docs/estimate-costs#estimating_query_costs_using_the_pricing_calculator.

Graziano, M., Canali, D., Bilge, L., Lanzi, A., and Balzarotti, D. (2015). Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 1057–1072, Washington, D.C. USENIX Association.

Hurier, M., Suarez-Tangil, G., Dash, S. K., Bissyandé, T. F., Le Traon, Y., Klein, J., and Cavallaro, L. (2017). Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 425–435.

Kim, D., Kwon, B. J., Kozák, K., Gates, C., and Dumitraş, T. (2018). The broken shield: Measuring revocation effectiveness in the windows code-signing pki. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 851–868, USA. USENIX Association.

Meli, M., McNiece, M. R., and Reaves, B. (2019). How bad can it git? characterizing secret leakage in public github repositories. In *NDSS*.

Munoz, A. (2020). The octopus scanner malware: Attacking the open source supply chain. https://securitylab.github.com/research/octopus-scanner-malware-open-source-supply-chain.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., and Cavallaro, L. (2019). Tesseract: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 729–746, USA. USENIX Association.

Peng, P., Yang, L., Song, L., and Wang, G. (2019). Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *Proceedings of the Internet Measurement Conference*, IMC '19, page 478–485, New York, NY, USA. Association for Computing Machinery.

Rokon, M. O. F., Islam, R., Darki, A., Papalexakis, V. E., and Faloutsos, M. (2020). Sourcefinder: Finding malware source-code from publicly available repositories.

Salem, A., Banescu, S., and Pretschner, A. (2019). Don't pick the cherry: An evaluation methodology for android malware detection methods. *CoRR*, abs/1903.10560.

Suciu, O., Mărginean, R., Kaya, Y., Daumé, H., and Dumitraş, T. (2018). When does machine learning fail? generalized transferability for evasion and poisoning attacks. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 1299–1316, USA. USENIX Association.

Syed, Z., Padia, A., Finin, T., Mathews, L., and Joshi, A. (2016). Uco: A unified cybersecurity ontology. https://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12574.

The Mitre Corporation (2017). Maec 5.0 specification – vocabularies. http://maecproject.github.io/releases/5.0/MAEC_Vocabularies_Specification.pdf.

VirusTotal (2020a). Getting started. https://developers.virustotal.com/reference.

VirusTotal (2020b). How it works – virustotal. https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works.

Wang, H., Si, J., Li, H., and Guo, Y. (2019). Rmvdroid: Towards a reliable android malware dataset with app metadata. In *Proceedings of the 16th International Conference on Mining Software Repositories*, MSR '19, page 404–408. IEEE Press.

ytisf (2020). Github - ytisf/thezoo: A repository of live malwares for your own joy and pleasure. thezoo is a project created to make the possibility of malware analysis open and available to the public. https://github.com/ytisf/theZoo.

Zeltser, L. (2020). How to share malware samples with other researchers. https://zeltser.com/share-malware-with-researchers/.

Zetter, K. (2014). A google site meant to protect you is helping hackers attack you. https://www.wired.com/2014/09/how-hackers-use-virustotal/.

Zhang, Y., Fan, Y., Hou, S., Ye, Y., Xiao, X., Li, P., Shi, C., Zhao, L., and Xu, S. (2020). Cyber-guided deep neural network for malicious repository detection in github. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 458–465.

Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L., and Wang, G. (2020). Measuring and modeling the label dynamics of online anti-malware engines. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2361–2378. USENIX Association.