What the Fork?
Finding and Analyzing Malware in GitHub Forks

Alan Cao
NYU
alan.cao@nyu.edu

Brendan Dolan-Gavitt
NYU
brendandg@nyu.edu

***Abstract*—On GitHub, open-source developers use the *fork* feature to create server-side clones and implement code changes separately before creating pull requests. However, such fork repositories can be abused to store and distribute malware, particularly malware that stealthily mines cryptocurrencies.**

In this paper, we present an analysis of this emerging attack vector and a system for catching malware in GitHub fork repositories with minimal human effort called *Fork Integrity Analysis*, implemented through a detection infrastructure called Fork Sentry. By automatically detecting and reverse engineering interesting artifacts extracted from a given repository's forks, we can generate alerts for suspicious artifacts, and provide a means for takedown by GitHub Trust & Safety. We demonstrate the efficacy of our techniques by scanning 68,879 forks of 35 popular cryptocurrency repositories, leading to the discovery of 26 forked repositories that were hosting malware, and report them to GitHub with seven successful takedowns so far. Our detection infrastructure allows not only for the triaging and alerting of suspicious forks, but also provides continuous monitoring for later potential malicious forks. The code and collected data from Fork Sentry will be released as an open-source project.

I. Introduction

As GitHub has evolved into the largest collaborative development platform, it has seen a rise in abuse of the platform's functionality itself, such as recent incidents of cryptomining in its Actions continuous integration (CI) offering [1]. In this paper, we demonstrate how GitHub's forking and artifact release features have given rise to a novel breed of supply chain attack, where threat actors fork popular repositories to serve malicious artifacts.

These threat actors can use their forks to trick users to pull and compile their source code, download malicious artifact releases, or use it as a storage solution for a multi-stage attack chain. Creating malicious forks is an attractive technique for malware propagation since forks are not indexed for public search by GitHub unless they have more stars than the parent repository [2]. This compounds the severity of these attacks as malicious forks can evade detection efforts by static analysis engines.

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2022
28 April 2022, San Diego, CA, USA
ISBN 1-891562-78-9
https://dx.doi.org/10.14722/madweb.2022.23001
www.ndss-symposium.org

The most popular example of this attack was seen with the Electrum Wallet (`spesmilo/electrum`), where threat actors forked the repository under official-looking accounts and tricked users into downloading compromised versions of the application that can exfiltrate users' wallet seeds to an attacker-controlled server [3]. This behavior led to the widespread theft of a substantial amount of Bitcoin from users, leading the original maintainers to get the repository taken down and launch their own counterattack against the attackers' servers. Aside from the use of forks to imitate known benign software, threat actors also have used forks as a storage medium to hide payloads as part of a larger attack chain [4]. This allows attackers to launch phishing and adware campaigns against victims, and pull second-stage payloads from GitHub rather than a suspicious domain that can be detected by an on-device antivirus or endpoint security solution.

It is important to note that there currently exists no mitigation for these types of attacks other than manual investigation and reporting to both maintainers and GitHub. Furthermore, threat actors can repeatedly fork and create malicious repositories [4], making fork malware difficult to catch and remediate before actual damage has been done.

In this paper, we focus on the automated discovery and alerting of such malicious fork repositories by implementing a detection infrastructure called Fork Sentry, which integrates with an open-sourced project to perform *fork integrity analysis*. During this process, forks are actively checked for malicious executables, and instances of abuse are reported back to maintainers for takedown by GitHub Trust & Safety. Fork Sentry uses typosquatting detection, well-established static malware detection engines to catch adversarial capabilities, and binary similarity hashing to reduce repeat analysis and connect threat actors. In addition, Fork Sentry can be deployed to monitor on a schedule to detect future attacks.

Given the number of actively maintained and used repositories on GitHub, we restrict the scope of our initial work to analyzing popular cryptocurrency-related repositories. As seen in the previously-mentioned campaigns and by Pastrana et al. [5], cryptocurrency-related projects are a popular target for attackers because of the large amount of money at stake. Additionally, we focus only on the detection of compiled executables in forks and their releases, with future work planned to augment this detection with source-based static

analysis tuned towards adversarial capabilities.

The main contributions of our work are:

1) **Introduce Fork Integrity Analysis** as a viable technique for the detection of malicious artifacts in forks of GitHub repositories.
2) **Analyze malicious behavior** of malware we detected in cryptocurrency-related forks, and present how they are being used and propagated against victims.

Our results from this work are as follows:

1) We identify and report **26 malicious forks** in 35 cryptocurrency repositories, causing seven of them to be taken down so far.
2) We **discover a persistent threat actor** who was actively propagating novel cryptocurrency-based malware, who appears to have made over $200,000 through illicit mining.
3) We use the results to provide **recommendations to GitHub** for mitigations to help prevent future supply chain attacks.

II. Background and Related Work

GitHub is the most popular code hosting platform, housing over 271 million repositories at the time of writing [6]. The platform includes a feature called *forking*, which allows developers to create public snapshot clones of a repository. This allows developers to introduce new changes separately before getting them merged and shipped, or to create a spinoff variant of the software. GitHub has also enabled a feature called *releases*, which complements repositories and allows maintainers to release pre-built versions of their software that users can download instead of having to build the project themselves.

A. Software Supply Chain Attacks

Our work falls into the area of *supply chain attacks*, where threat actors introduce malware backdoors into software components that are used by developers. Kaplan et al. [7] examined the attack vectors that can be carried out against package dependency registries, such as typosquatting package names and exploiting outdated dependencies. Duan et al. [8] take this further by implementing an analysis pipeline that found 278 malicious dependencies across three registries. Enriksen [9] also attacked this problem in Go dependencies by generating typosquat permutations of popular packages, uncovering a malicious fork of the commonly used `urfave/cli` package.

While this work is important in highlighting the severity of attacks against the open-source ecosystem, it is limited in only looking at threats that hide in software dependencies, rather than directly in code repositories themselves.

B. Finding GitHub Malware

One avenue of approach to mitigate against *any* form of supply chain attack is to automate the detection of malicious source code in repositories. Measurements done by Rokon et al. [10] demonstrated the prevalence of malware source code on GitHub, with Gonzalez et al. [11] going further by building a system that applies a decision model for detecting malicious commits as they enter a codebase.

While these works overlap somewhat with our own in *malware detection on GitHub*, they consider only malicious *source code* and assume a threat model of attackers that want to push malicious code directly to the parent repository, rather than hide compiled artifacts in separate forks. Work done by Cholter et al. [12] is more closely related to ours, as they conduct large-scale mining and scanning of Windows and C/C++-based malware in GitHub repositories. However, they do not focus on fork repositories, as these are not directly indexed for analysis by independent researchers. Thus, prior work involved in the detection and remediation of malware on GitHub is not directly applicable for detecting fork-based malware.

C. Fork Analysis

Work has been done to examine the interactions between fork repositories and their parents [13], [14], [15]. Notably, Ren et al. [13] introduce a similar system to ours that gives insight into forks and the behaviors that they are introducing to the parent repository. While efforts like these demonstrate the necessity of understanding how forks interact with their parents, it does not directly address the security gap that exists in finding threats in forks.

III. Malicious Indicators

Before presenting the design of Fork Sentry, we gathered several heuristics that have been previously observed in open-source malware attacks that should be detected when scanning a repository's forks.

A. Typosquatting

As pointed out by Taylor et al. [16], a suspicious indicator that a piece of software is malicious is the presence of *typosquatting*, where the name is intentionally misspelled in the hope that a user will accidentally use it instead of the original. This is an attractive technique to maximize the spread of malware, and it is important for Fork Sentry to be able to recognize it. A common detection technique is to compute the *Levenshtein distance* [17]. We can apply this in Fork Sentry by looking at the distance between the parent repository name and its forks, flagging any forks with a small distance score for further investigation.

B. Presence of Compiled Malicious Executables

GitHub repository trees typically do not contain compiled executables, since it is designed for distributing source code files. As seen in the Avast report [4], the presence of executable artifacts committed directly to the repository tree is a suspicious indicator, since it can cause unsuspecting users to execute the executable or be used as a place to store artifacts in a larger attack. Furthermore, shell scripts can also complement these artifacts to help bootstrap the payload properly when executing on a victim's host, e.g. by setting input flags for a cryptominer. Fork Sentry needs to be aware of the presence of

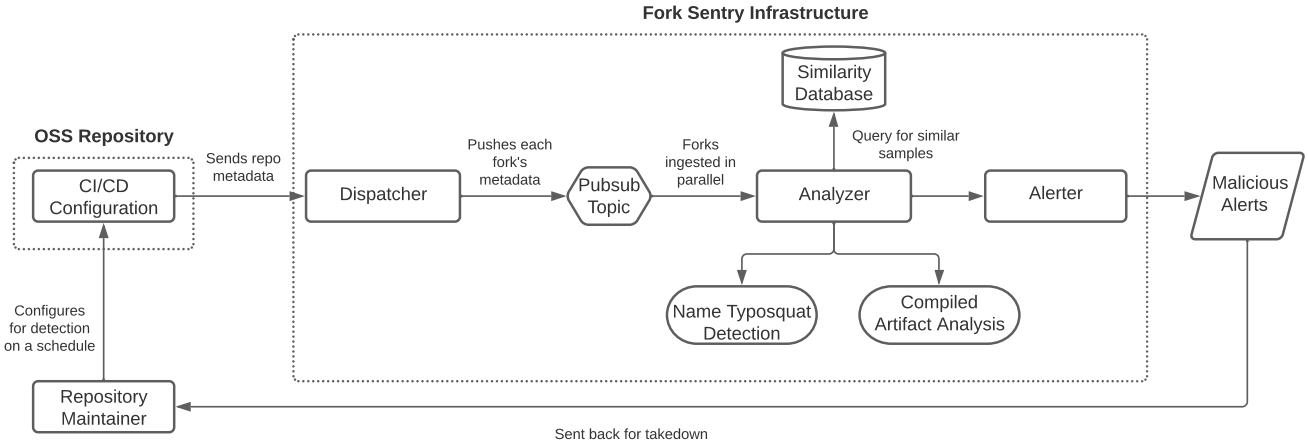

Fig. 1. A high-level overview of Fork Sentry's detection architecture.

binaries and shell scripts directly committed to the repository. However, our infrastructure should not indiscriminately label a fork as malicious based only on the presence of these files, since there are many benign forks that may have *accidentally* committed compiled executables of the source code. Thus, it is important to additionally check for *malicious capabilities* such as packing, obfuscation, and signatures of known malware families.

Forks can have malware in their GitHub releases, since it is a natural location for pre-built executables, and GitHub does not perform malicious artifact scanning. By not storing payloads directly in the repository tree, attackers can phish more effectively by alleviating suspicions from potential victims. As such, our compiled executable detection must also include files in artifact releases associated with forks.

IV. Design

A. Goals

While we want Fork Sentry to be able to detect the malicious indicators laid out earlier, it should also fulfill several implementation goals to maximize its usefulness to both security researchers and repository maintainers.

Automation. Conducting fork integrity analysis needs to be *automated* so that security researchers and maintainers can recover alerts and request takedowns quickly, rather than spending time on manual investigation. In addition, our analysis should be *continuous* in order to detect newly emerging fork threats.

Robustness. One obstacle that needs to be overcome is the GitHub API rate limit that is imposed on authenticated users [12]. This rate limit unfortunately imposes restrictions on how we can gather critical metadata about forks. Thus, our infrastructure must attempt to minimize the number of API calls used while scraping, and also recover gracefully and continue fork analysis if one of our components fails due to the rate limit.

Scalability. Popular target repositories can contain thousands of forks by other GitHub users, which can further aid in hiding their repository from showing up in the public fork tree. Thus, the analysis time for a large number of fork repositories may become too high for timely detection and alerting. Therefore, Fork Sentry needs to be able to efficiently *scale* analysis to millions of repositories and their forks, processing inputs quickly and asynchronously and allowing the workload to be distributed across as many workers as are available.

B. Architecture

Figure 1 shows the internal components that make up the Fork Sentry infrastructure, which consists of the Dispatcher, Analyzer, and Alerter. Figure 2 depicts the analysis workflow that gets carried out for an individual fork repository and its artifacts, not including typosquatting detection. We will discuss these components and explain the implementation details that help fulfill our design goals.

Dispatcher. The dispatcher offers an API endpoint that can be invoked *ad-hoc* by a user or *on a schedule* in a CI/CD workflow to kick off the fork integrity analysis. The API endpoint consumes a target parent repository and uses the GitHub API to recover all forks. To be comprehensive in our scanning efforts, we do a depth-first traversal on the parent's fork tree to recover additional children of forks themselves.

Analyzer. Conforming to a publish-subscribe model, forks recovered are all pushed to a message topic. Analyzers are then instantiated as subscribers to perform malware analysis on each fork, as highlighted by the workflow in Figure 2. To avoid overloading the GitHub API, the fork repository is cloned to disk, and relevant file types are filtered for malware analysis by traversing all newly made changes in the commit history

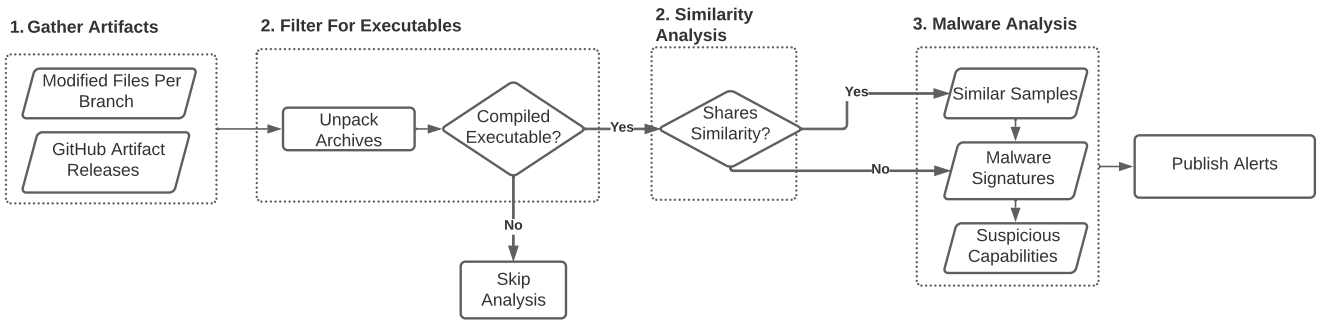

Fig. 2. Compiled artifact analysis workflow employed for an individual target fork repository.

of each branch. Artifacts are also downloaded and filtered for relevant compiled executables. These filtered samples are then checked against the similarity database with their computed *locality-sensitive hash* to determine if they have been seen previously.

Next, *signature-based* detection using ClamAV [18] and *capability-based* detection with `capa` [19] will be employed to do the actual malware analysis. Checking for *signatures* helps determine if artifacts resemble known malware, and checking for *capabilities* helps determine suspicious actions statically for potentially novel samples. If suspicious artifacts match a known signature, capability, or have similar samples, alerts are generated and sent to the alerter for output.

To fulfill our goal of being *robust* and to mitigate GitHub's imposed rate limit, the analyzer *re-publishes* the fork for analysis in the next hour if the rate limit is reached. This ensures that we are able to continue comprehensive analysis of forks without missing any repositories.

Similarity Database. It is important to detect similar samples, as it helps both reduce the analysis workload for samples that have already been ingested, and to highlight clusters of similar malware that may indicate longer-running campaigns in the fork ecosystem.

To detect similar samples, we compute and store *locality-sensitive hashes* produced by the `ssdeep` algorithm [20], which uses context-triggered piecewise hashing to create a "fingerprint" for each file. `ssdeep` can then compute a similarity score between two hashes; according to its author, any score greater than 0 indicates that the files are similar [21]. However, a straightforward implementation of this similarity search requires a full scan over the hash database for each sample.

To help achieve our goal of scalability, we employ an optimization from Wallace [22] that indexes the hashes in the database by their *chunk size*; a hash with chunk size n can only match hashes with chunk sizes $\frac{n}{2}$, n, or $2n$, so this allows the search to be performed over a much smaller set of candidates. Although `ssdeep` is robust and well-understood, there are several newer similarity hashes intended for malware analysis (e.g., Lempel-Ziv Jaccard Distance [23] and TLSH [24]) that may offer better performance; we hope to explore these in future research. The online malware analysis service VirusTotal also supports both `ssdeep` and TLSH, which could potentially be used to expand the scope of our similarity checking.

TABLE I
CRYPTOCURRENCY REPOSITORY SCAN SUMMARY

Category	Repos	Total Forks	Malicious Forks
Core Implementation	5	47,723	0
Cryptominer	22	13,964	24
Wallet	8	7,192	2
Total	35	68,879	26

TABLE II
LOCATION OF DISCOVERED MALWARE IN FORKS

Location	# of Malicious Forks
Repository Tree	14
Releases	12

Alerter. Finally, suspicious forks that are detected are passed to the alerter for storage and to output back to parent repository maintainers through the GitHub issues tracker for the project.

V. EVALUATION

We scanned 35 popular cryptocurrency repositories (a total of 68,879 forks) with Fork Sentry and detected 26 malicious forks. Table I breaks down the categorization of the target repositories that we chose to scan, and the number of malicious forks found for each. According to Table II, we noticed that

TABLE III
TARGETED OPERATING SYSTEM

Operating System	# of Malicious Forks
Windows	3
Linux	22
Android	1

TABLE IV
SUSPICIOUS INDICATORS

Category	# of Malicious Forks
Typosquatting/Confusion	2
Packing	6
Malicious Cryptomining	26

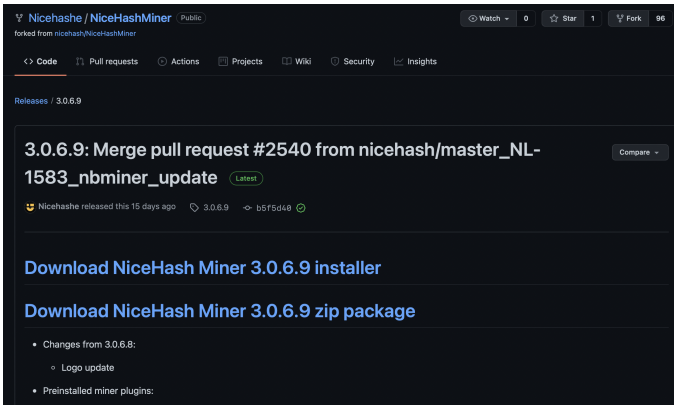

Fig. 3. Typosquatted fork of `nicehash/NiceHashMiner` serving a malicious release.

a near equal amount of forks stored malware artifacts in their repository trees and releases, and as per Table III, a majority of malware discovered is compiled for Linux. We have reported every malicious fork that has been detected, resulting in seven takedowns by GitHub Trust & Safety, with others pending review.

From our pool of recovered malicious forks, we will highlight malicious capabilities prevalent in those samples, and examine in more detail a specific instance of a persistent threat actor that spread novel malware across multiple detected forks.

A. Malicious Behaviors

Name Typosquatting and Confusion. From our analysis, we discovered one instance of a fork repository typosquatting `nicehash/NiceHashMiner`, where an account with the name of `nicehashe` was created with malicious releases to attract new victims (Figure 3). Furthermore, we detected another suspicious fork of `spesmilo/electrum` that employed *confusion* by naming itself `electrummoneroclassic/electrummoneroclassic` and serving its own suspicious releases.

Packing. One common malicious behavior is using different packing routines to both obfuscate and make files smaller. We noticed an instance of malware using UPX as a packer, a Windows malware that employed the VMProtect packer, and an Android-based miner that employed the Chinese-based *Jiagu* packer. Furthermore, as detailed in the next section, we also uncovered malware that used a custom packing and anti-analysis workflow.

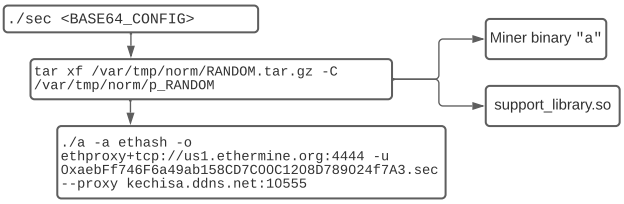

Fig. 4. Unpacking and execution workflow for `sec` malware.

Malicious Cryptomining. All of the malware samples detected by Fork Sentry are tagged as malicious cryptominers. We recognize that many malware detection engines often flag cryptominers, even if created to be legitimately used for participating in proof-of-work. In order to limit the amount of false positives, we ensured that samples detected as cryptominers needed to have *additional* tags that signified malicious behaviors.

B. Persistent Threat Actor Case Study

We highlight a case study of a recurring threat actor and the discovery of their novel malware inside the forks of several cryptominer repositories, and how it was potentially being propagated amongst victims.

Discovery. Fork Sentry detected this particular piece of malware first when evaluating `pooler/cpuminer`, where the fork `cambel183/cpuminer` and a transitive fork, `michaelmusto26/cpuminer`, were found serving malicious miners that used packing and anti-analysis techniques in their repository trees. Furthermore, the samples created by this threat actor showed up again in another cryptominer repository `ethereum-mining/ethminer` as `89565004/ethminer`, due to high similarity scores between the artifacts in the codebase. These forks and their malware were active from June 2021 until their time of takedown in November 2021; here we will specifically discuss the malware found in `89565004`.

Runtime. Fork Sentry reported on the presence of several malicious bootstrap scripts that complemented these samples, which are run post-exploitation and are used by the malware to establish persistence and stealth on a compromised server. The script found in `89565004/ethminer` is seen in Listing 1. An inconspicuously named systemd unit file, `gpu1.service`, is created to run another malicious script, `covi2.sh` (Listing 2) after networking is enabled, which executes the actual cryptominer.

Packing & Anti-Analysis. The main Golang-based malware, `sec`, implements a runtime-based packing routine, demonstrated by the workflow in Figure 3. During execution, the malware untars and writes to disk the actual cryptominer, a copy of the `nheqminer` cryptominer, and an additional `support_library.so` shared object that is decoded from Base64. The shared object appears to be dynamically loaded

by the initial sample, which then executes the actual cryptominer. Here, anti-debugging is employed additionally to check for `ptrace` interactions by a debugger, and the contents of the `LD_PRELOAD` environment variable.

```
cd /home
sudo wget https://raw.githubusercontent.com
    /89505004/ces/master/scripts/covi2.sh
sudo wget https://raw.githubusercontent.com
    /89505004/ces/master/a
sudo wget https://raw.githubusercontent.com
    /89505004/ces/master/sec
sudo chmod +x sec
sudo chmod +x a
sudo chmod +x covi2.sh
sudo rm -rf /lib/systemd/system/gpu1.service
sudo rm -rf /lib/systemd/system/gpu.service
sudo rm -rf /lib/systemd/system/eth.service
sudo rm -rf /var/crash
sleep 86400
bash -c 'cat <<EOT >>/lib/systemd/system/gpu1.
    service
[Unit]
Description=gpu1
After=network.target
[Service]
ExecStart= /home/covi2.sh
WatchdogSec=18000
Restart=always
RestartSec=60
User=root
[Install]
WantedBy=multi-user.target
EOT
' &&
systemctl daemon-reload &&
systemctl enable gpu1.service &&
service gpu1 stop  &&
service gpu1 restart
```

Listing 1. Script bootstrapping malicious cryptominers on a compromised server.

```
/home/sec huZU1vijNr6S6G1f766z//
    ocy7gAa4Md1YWTvhCl234IDNVZVKImiFO23hiajKuAyKu+79
    zCdW0oU2fdd7a9hARp9Yppff4jFIwse2rA85zxwVV...
```

Listing 2. Malware execution with Base64-encoded configuration.

From the given address used to mine Ethereum, we noted that this attacker seemed to have already made approximately 76.46 ETH [25], worth approximately $234,555 in USD at the time of this writing)[1]. After validating our automated analysis, we submitted takedown requests to GitHub Trust & Safety, which were all fulfilled in the span of a week.

VI. Discussion

Our work showcases that using fork repositories to host malware is indeed a rising problem in open-source security, based on several observations we can make from our results:

1) **Fork-based malware is quite prevalent in cryptomining repositories.** This is concordant with observations made by Pastrana and Suarez-Tangil [5]; given the high economic incentives associated with the growth of proof-of-work cryptocurrencies, attackers will naturally gravitate toward this area.
2) **Fork-based malware is mostly built for Linux.** We can speculate that many of these attacks carried out are against Linux containers, servers, and IoT (Internet-of-Things) devices, and used post-intrusion to capitalize off of compute and memory capacity for revenue generation.
3) **Malware in releases are enticing for phishing.** The typosquatting/confusion malware introduced in Section I and found by us show how easy it is for threat actors to exploit misspelling errors propagate malware.

Given these observations, should GitHub be responsible for detection of malicious forks (or any malware in repositories), or defer that effort to the community? We believe that GitHub should take action given the *implicit trust* users naturally place on the platform, and the pervasive effects of these attacks on them when that trust is exploited by threat actors. Moreover, first-party malware scanning by GitHub would not be affected by the API rate limits that hinder independent researchers, and they have access to other metadata, such as IP and email addresses, that can be used to more effectively detect and link threat actors across campaigns. Therefore, although we believe Fork Sentry is a valuable interim solution, we also make several recommendations to GitHub to enhance defense against this new attack vector in the long-term.

A. Recommendations

Account typosquatting prevention. GitHub should apply edit distance detection to block users from creating accounts/organizations that are too similar to one that is existing, such that they do not have the opportunity to take advantage of misspelling accidents by users to serve them malware.

Comprehensive malware scanning. Filtering and malware analysis should be conducted on executable artifacts committed to any repository and their releases. Such work has been proposed for hosting vendors like GitLab [26], but has not come to fruition. Given the presence we've seen of packing and cryptomining, such detections should incorporate heuristics for catching these capabilities.

Code provenance for releases. Releases should only be public if it can be demonstrated that they were created from a specific codebase. Efforts such as applying program analysis [27] and compiler and attribute identification [28] can help verification, but are difficult to make perfectly accurate given the diversity of compilation runtimes. Thus, a more practical technique is to require that releases be created through publicly verifiable continuous integration builds, rather than allowing users to directly upload assets on their own.

Improved indexing and search for detection. Forks that have changes from their parents should be indexed for static analysis efforts, and researchers should also be able to query for non-source file types to better catch malicious artifacts.

[1]The malware runs a miner and directs the mining reward to Ethereum address 0xaebFf746F6a49ab158CD7C00C1208D789024f7A3; note that we cannot confirm what portion of this revenue from mining may have been gained through *illicit* mining.

VII. Future Work

Given the diversity of threats and the variety of analysis techniques, we highlight future work that can help bolster our accomplishments.

A. Analyzing Other Modified Artifacts

Fork Sentry does not account for adversarial changes in source code, where injected source can discreetly conduct malicious behaviors. Catching such attacks will either involve incorporating prior work done in identifying malicious source code and commits or applying source-based diff-aware static analysis.

Another area of detection can be in *compilation manifests*, which establish metadata, dependencies, and runtime behavior during installation. The Octopus Scanner malware [29] is an example of this, where the Java build process for a popular project is backdoored to install malware. Parsers for different manifest formats can be incorporated to statically analyze these configurations to recognize if malicious capabilities are introduced.

B. Improving Similarity Analysis

Previous studies [30] have showcased how similarity hashes like `ssdeep` that employ a *sliding window* technique on the entirety of a file lack context in binary features that may be critical to associating similar samples. As such, false negatives may become prevalent in identifying samples that are compiled and released, particularly if malware authors actively attempt to evade similarity analysis.

When examining the effectiveness of our similarity analysis with the weaponized cryptominer propagated by the persistent threat actor against all other samples detected in `pooler/cpuminer`'s forks, we found that Fork Sentry fails to recognize its similarity to original `cpuminer` binaries, even though it is only lightly modified.

In addition, because our scope is currently limited to cryptocurrency repositories, obtaining reliable ground truth about whether a given repository is malicious can be difficult. Although we require that cryptominers have *additional* suspicious indicators (described in Section IV-B) to be deemed actually malicious and worth reporting to GitHub, we lack sufficient context to determine if any of the unreported repositories were, in fact, malicious.

To address this, we plan to investigate other similarity clustering techniques, perform deeper analysis on the *meaning* of the changes made in a fork repository, and broaden the scope of our work to incorporate other repositories and source code.

C. Expanding Target Scope

In this paper, we focused on analyzing cryptocurrency-based repositories, a very small subset of all projects on GitHub. This scope can be widely expanded for many other open-sourced ecosystems. For instance, as demonstrated by Enriksen's [9] work, Go-based packages have been targets of malware infection through typosquatted forks accidentally used as dependencies. Popular offensive security projects are also good candidates, since there may be malicious forks hidden amongst projects meant for pedagogy or research. Finally, we could further expand our scope to include *all* popular repositories (using, e.g., Google's BigQuery dataset for GitHub repositories [31]).

Availability

To help others build on our work, we have released the code of our fork scanning infrastructure under an open source license at:

https://github.com/ex0dus-0x/fork-sentry

Datasets containing the results of our initial fork scans, binary similarity database, and analyses are available at:

https://zenodo.org/record/6391341

https://www.virustotal.com/gui/collection/f15433215537bc3dea2e71718778ca70f8241228cd2418b5b7f21a3a729a34da

Acknowledgements

We would like to thank our anonymous reviewers for their time and effort and for the valuable feedback they provided. We also thank the NYU OSIRIS lab for the their support by lending us computing infrastructure, which we used to run our analyses.

References

[1] C. Patterson. (2021) Github actions update: Helping maintainers combat bad actors. [Online]. Available: https://github.blog/2021-04-22-github-actions-update-helping-maintainers-combat-bad-actors/

[2] GitHub. (2021) Searching in forks. [Online]. Available: https://docs.github.com/en/search-github/searching-on-github/searching-in-forks

[3] Malwarebytes Lab. (2019) Electrum bitcoin wallets under siege. [Online]. Available: https://blog.malwarebytes.com/cybercrime/2019/04/electrum-bitcoin-wallets-under-siege/

[4] Avast Threat Intelligence Team. (2021) Greedy cybercriminals host malware on github. [Online]. Available: https://blog.avast.com/greedy-cybercriminals-host-malware-on-github

[5] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 73–86.

[6] GitHub. (2021) Github search. [Online]. Available: https://github.com/search

[7] B. Kaplan and J. Qian, "A survey on common threats in npm and PyPI registries," in *International Workshop on Deployable Machine Learning for Security Defense*, 2021, pp. 132–156.

[8] R. Duan, O. Alrawi, R. P. Kasturi, R. Elder, B. Saltaformaggio, and W. Lee, "Towards measuring supply chain attacks on package managers for interpreted languages," in *Network and Distributed Systems Symposium (NDSS)*, 2021.

[9] M. Henriksen. (2021) Finding evil Go packages. [Online]. Available: https://michenriksen.com/blog/finding-evil-go-packages/

[10] M. O. F. Rokon, R. Islam, A. Darki, E. E. Papalexakis, and M. Faloutsos, "Sourcefinder: Finding malware source-code from publicly available repositories in github," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 149–163.

[11] D. Gonzalez, T. Zimmermann, P. Godefroid, and M. Schäfer, "Anomalicious: Automated detection of anomalous and potentially malicious commits on GitHub," in *IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 2021, pp. 258–267.

[12] W. La Cholter, M. Elder, and A. Stalick, "Windows malware binaries in C/C++ GitHub repositories: Prevalence and lessons learned." in *ICISSP*, 2021, pp. 475–484.

[13] L. Ren, S. Zhou, and C. Kästner, "Poster: Forks insight: Providing an overview of GitHub forks," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018, pp. 179–180.

[14] S. Brisson, E. Noei, and K. Lyons, "We are family: Analyzing communication in GitHub software repositories and their forks," in *IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, pp. 59–69.
[15] M. Biazzini and B. Baudry, "'May the fork be with you': novel metrics to analyze collaboration on GitHub," in *Proceedings of the 5th international workshop on emerging trends in software metrics*, 2014, pp. 37–43.
[16] M. Taylor, R. Vaidya, D. Davidson, L. De Carli, and V. Rastogi, "Defending against package typosquatting," in *Network and System Security*, 2020.
[17] D.-L. Vu, I. Pashchenko, F. Massacci, H. Plate, and A. Sabetta, "Typosquatting and combosquatting attacks on the Python ecosystem," in *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 509–514.
[18] ClamAV. Clamav. [Online]. Available: https://www.clamav.net
[19] W. Ballenthin and M. Raabe. capa: Automatically identify malware capabilities. [Online]. Available: https://www.mandiant.com/resources/capa-automatically-identify-malware-capabilities
[20] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital Investigation*, vol. 3, pp. 91–97, 2006, the Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).
[21] F. Pagani, M. Dell'Amico, and D. Balzarotti, "Beyond precision and recall: understanding uses (and misuses) of similarity hashes in binary analysis," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 354–365.
[22] B. Wallace, "Optimizing ssdeep for use at scale," Virus Bulletin, Tech. Rep., 2015.
[23] E. Raff and C. Nicholas, "Lempel-Ziv Jaccard Distance, an effective alternative to ssdeep and sdhash," *Digital Investigation*, vol. 24, pp. 34–49, 2018.
[24] J. Oliver, C. Cheng, and Y. Chen, "TLSH - a locality sensitive hash," in *4th Cybercrime and Trustworthy Computing Workshop*, 2013.
[25] Block explorer for Ethereum Mainnet. [Online]. Available: https://etherscan.io/address/0xaebFf746F6a49ab158CD7C00C1208D789024f7A3
[26] A. Newdigate. GitLab issue #25044: Integrate malware detection directly into GitLab. [Online]. Available: https://gitlab.com/gitlab-org/gitlab/-/issues/25044
[27] Y. Ji, L. Cui, and H. H. Huang, "Vestige: Identifying binary code provenance for vulnerability detection," in *International Conference on Applied Cryptography and Network Security*, 2021, pp. 287–310.
[28] N. E. Rosenblum, B. P. Miller, and X. Zhu, "Extracting compiler provenance from program binaries," in *Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, 2010, pp. 21–28.
[29] A. Munoz. (2020) The Octopus Scanner malware: Attacking the open source supply chain. [Online]. Available: https://securitylab.github.com/research/octopus-scanner-malware-open-source-supply-chain/
[30] Y. Li, S. C. Sundaramurthy, A. G. Bardas, X. Ou, D. Caragea, X. Hu, and J. Jang, "Experimental study of fuzzy hashing in malware clustering analysis," in *8th Workshop on Cyber Security Experimentation and Test (CSET 15)*, 2015.
[31] F. Hoffa. GitHub on BigQuery: Analyze all the open source code. [Online]. Available: https://cloud.google.com/blog/topics/public-datasets/github-on-bigquery-analyze-all-the-open-source-code