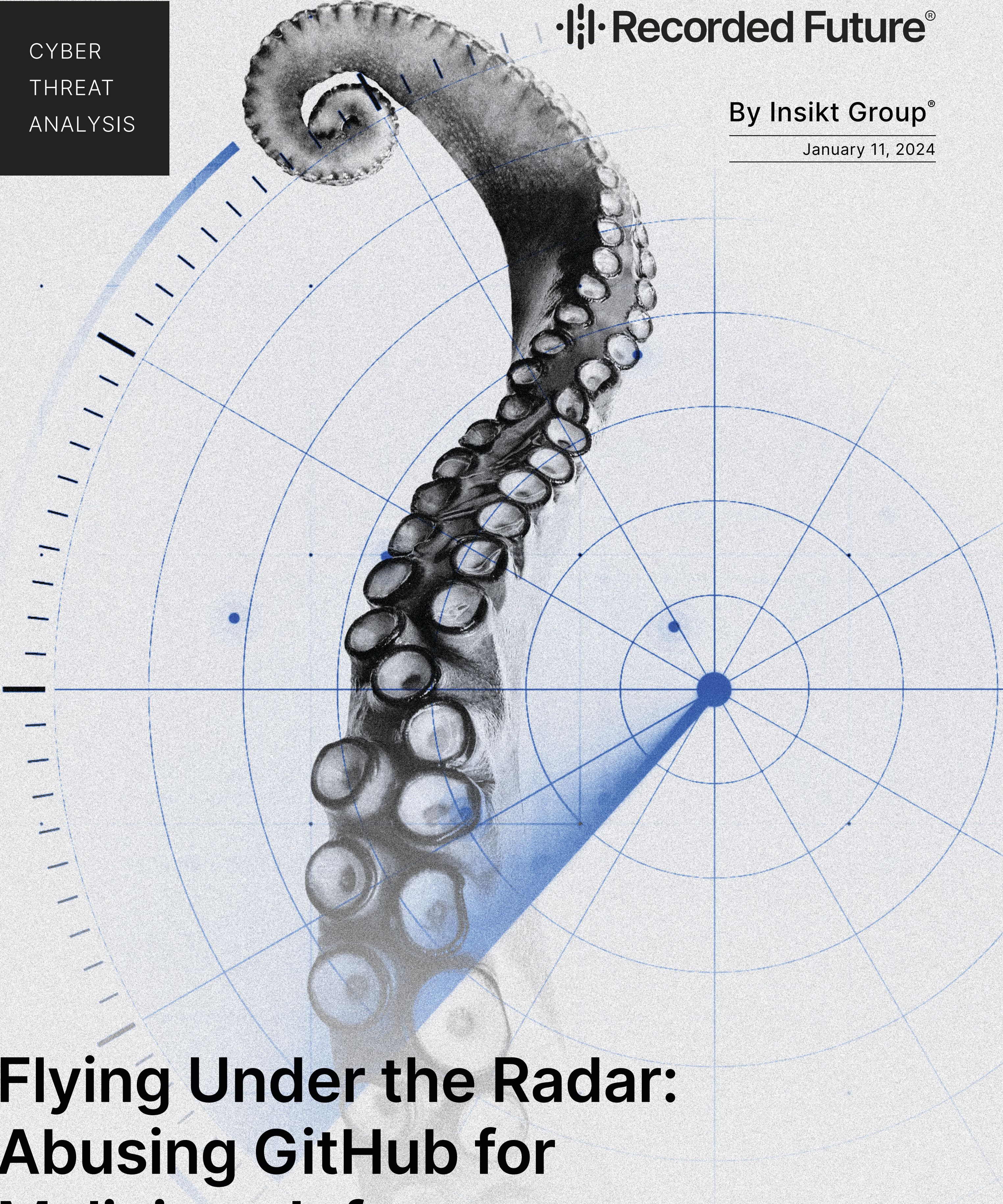
CYBER
THREAT
ANALYSIS

Recorded Future®

By Insikt Group®

January 11, 2024



Flying Under the Radar: Abusing GitHub for Malicious Infrastructure

Executive Summary

GitHub's services are frequently abused both by cybercriminals and advanced persistent threats (APTs) for a wide range of malicious infrastructure schemes. While payload delivery stands out as the most common infrastructure scheme in instances of GitHub abuse, others, such as dead drop resolving (DDR), exfiltration, and full command-and-control (C2), are also observed. Using GitHub services for malicious infrastructure allows adversaries to blend in with legitimate network traffic, often bypassing traditional security defenses and making upstream infrastructure tracking and actor attribution more difficult. The "living-off-trusted-sites" (LOTS) approach is expected to grow with APTs' increasing usage and less-sophisticated groups swiftly following suit.

In the near term, defenders should pursue a service-based strategy by flagging or even blocking specific GitHub services that are not normally used in their environment and are known to be used maliciously. This should be paired with a context-based strategy based on the principle that only specific parts of a corporate environment necessitate interaction with particular GitHub services. In the longer term, organizations should allocate resources to better understand how GitHub and other code repositories are abused. This understanding will facilitate the development of more sophisticated detection mechanisms based on log data and threat intelligence, ultimately enhancing the overall protection of organizations. Simultaneously, tailored threat hunting can help defenders identify novel threat actor tactics, techniques, and procedures (TTPs) to serve as a base for detections.

As these attacks are likely to increase, legitimate internet services (LIS) will increasingly constitute a new third-party risk vector for customers. Further, as threat actors refine their abuse techniques, the success of current defense systems will diminish, paralleling paradigm shifts seen with the rise of "living-off-the-land binaries" (LOLBins) in recent years. Effective mitigation strategies will require more advanced detection methods, more comprehensive visibility, and more diverse detection angles. More importantly, shifts in security ownership are anticipated, as LIS may be expected to assume more responsibility for combating abuse through structural changes and product innovations; their unique visibility into user and usage data may be required to develop tailored, robust defenses. Accordingly, LIS's competitive advantage will likely be shaped by its ability to manage this risk and support customer protection.

Key Findings

- GitHub is frequently abused across all malware categories and by all types of threat actors, offering benefits like detection evasion, reduced operational overhead, and lower costs. The 4 primary infrastructure schemes for GitHub abuse are payload delivery, DDR, full C2, and exfiltration.
- Payload delivery stands out as the most prevalent infrastructure scheme, mainly due to its ease of implementation and alignment with GitHub's "legitimate" use case. Yet, it poses a risk to threat actors of unintended exposure, potentially revealing operational insights into development

capabilities and tempo, targets, and attack vectors, as seen in some notable operational security failures.

- The use of GitHub for DDR is common and comes in various forms. This is driven by the minimal risk of data removal, which is exacerbated by the challenges faced by platform operators like GitHub in discerning malicious intent behind posted addresses or strings without further context.
- Full C2 implementations in GitHub are relatively uncommon and are predominantly linked to APT activity thus far. This may be due to functional constraints imposed by GitHub's services, concerns about potential exposure of threat operations, and implementation challenges.
- While GitHub can be used for exfiltration, it's used less commonly for this than for other infrastructure schemes. There is no robust evidence explaining why this is the case, but the most likely reasons include file size and storage limitations, the existence of more efficient alternatives, cost considerations, and concerns about detectability.
- Beyond the 4 primary infrastructure schemes, GitHub services are abused for various other malicious infrastructure-related purposes. These include hosting phishing operations, acting as fallback channels, and serving as an infection vector through repository poisoning techniques.
- There is no universal solution for GitHub abuse detection. A mix of detection strategies is needed, influenced by specific environments and factors such as the availability of logs, organizational structure, service usage patterns, and risk tolerance, among others.
- With the increasing abuse of GitHub and other LIS, not only will defenders need to allocate more resources to combat this abuse, but LIS themselves are also expected to assume a more prominent role in addressing it through policy changes and technical innovations, leveraging their unique visibility and clout.

Background

GitHub, utilized by approximately 94 million users, is a widely adopted platform initially created for version control and collaborative coding, enabling developers to store, manage, and track changes to their code repositories. It facilitates collaborative software development by providing tools for code hosting, version history tracking, issue tracking, and code review. Similar to most legitimate internet services (LIS), GitHub's services are abused both by cybercriminals and advanced persistent threats (APTs) for a wide range of malicious infrastructure schemes, including payload delivery, DDR, exfiltration, command-and-control (C2), and other purposes (such as phishing). This follows the principle that every freely accessible API can theoretically serve as a cost-free resource for malicious infrastructure.

Abusing GitHub for malicious infrastructure not only enables threat actors to evade detection within victim networks by blending with benign network traffic but also provides other advantages, as noted in the first part of this series. These advantages include:

- **No blocking** of GitHub domains in most corporate networks, given its popularity among businesses and the fact that many of them rely on it
- **Reduced operational overhead** by simplifying the overall C2 server installation process by utilizing publicly endorsed TLS encryption
- **Widespread practical experience** with GitHub among malware developers, given its legitimate use cases beyond malicious activities
- **Lower infrastructure costs** by saving on typical hosting or registration fees
- **High uptime** as GitHub is designed to be highly available, with redundant servers and failover mechanisms
- **Minimal vetting** to register new accounts on GitHub (for example, the absence of a requirement for a credit card during registration represents significant cost savings for sophisticated APTs, as creating untraceable financing and payment methods is time-consuming and introduces unnecessary complexity)
- **Limited detection possibilities** for service providers (especially with respect to human-controlled accounts)
- **Tracing a threat actor upstream** or **identifying victims becomes more challenging** when the threat actor uses an LIS. More specifically, if tracking efforts hinge on network traffic analysis, encountering an LIS becomes a major obstacle, making it difficult to distinguish malicious traffic from legitimate traffic, resulting in a virtual dead end in the investigation.
- **Limited availability of tooling for threat modeling** and little actionable threat intelligence specific to such infrastructure setups

Although these advantages might make GitHub appear to be an ideal choice, using it for malicious infrastructure also entails disadvantages. These disadvantages include:

- **Limitations imposed by GitHub's functionality** prevent full flexibility — one such limitation is the inability to host PHP-based tools such as phishing kits due to the absence of PHP backend services
- **Ease with which GitHub can be blocked** within the victim network
- **GitHub likely has greater visibility** into hosted infrastructure compared to when threat actors manage it independently. This increased visibility could result in improved tracking of infrastructure discovery and identification of victims.
- **The existence of specialized teams** within GitHub assigned to detect and counter system abuses; these teams are known for their high level of responsiveness
- **Limitations in terms of file sizes** for free accounts, including warnings being issued when a user attempts to add or update files exceeding 50 MB, the blocking of files larger than 100 MB (unless Git Large File Storage is used), and a size limitation of 25 MB for files added via the browser
- **Limitations in terms of usage and bandwidth** (for example, Git Large File Storage and GitHub Pages sites have bandwidth limits of 1 GB and 100 GB per month, respectively)
- **The absence of privacy in public repositories** permits anyone to access the hosted code and its Git history, potentially enabling researchers to gain operational insights (for example, an operational security failure by APT37 in 2023 surfaced previously unknown targets and attack vectors); this exposure also facilitates the reconstruction of timelines related to distributed malware families and provides insights into the threat actor's development capability and tempo

While our first report in this series provided a systematic overview of how LIS is generally leveraged for malicious purposes (both in qualitative and quantitative terms), this report analyzes how threat actors abuse GitHub for various malicious infrastructure schemes and discusses detection strategies. While GitHub vulnerabilities can be exploited for malicious infrastructure purposes (for example, recent namespace retirement conditions within GitHub's repository creation and username renaming allowed repojacking[1]), this report concentrates on the abuse of legitimate GitHub services without any exploitation.

[1] Repojacking, a shortened term for repository hijacking, is a method by which a threat actor can circumvent a security mechanism known as "popular repository namespace retirement" to gain control over a repository.

Overview of GitHub's Services

GitHub is often used as an umbrella term for a variety of services, each with different use cases, abuse possibilities, and, ultimately, detection strategies. In what follows, we introduce commonly abused GitHub services and concepts to facilitate detailed discussions in subsequent sections. **Figure 1** shows the breakdown of abuse across GitHub services among the analyzed samples.

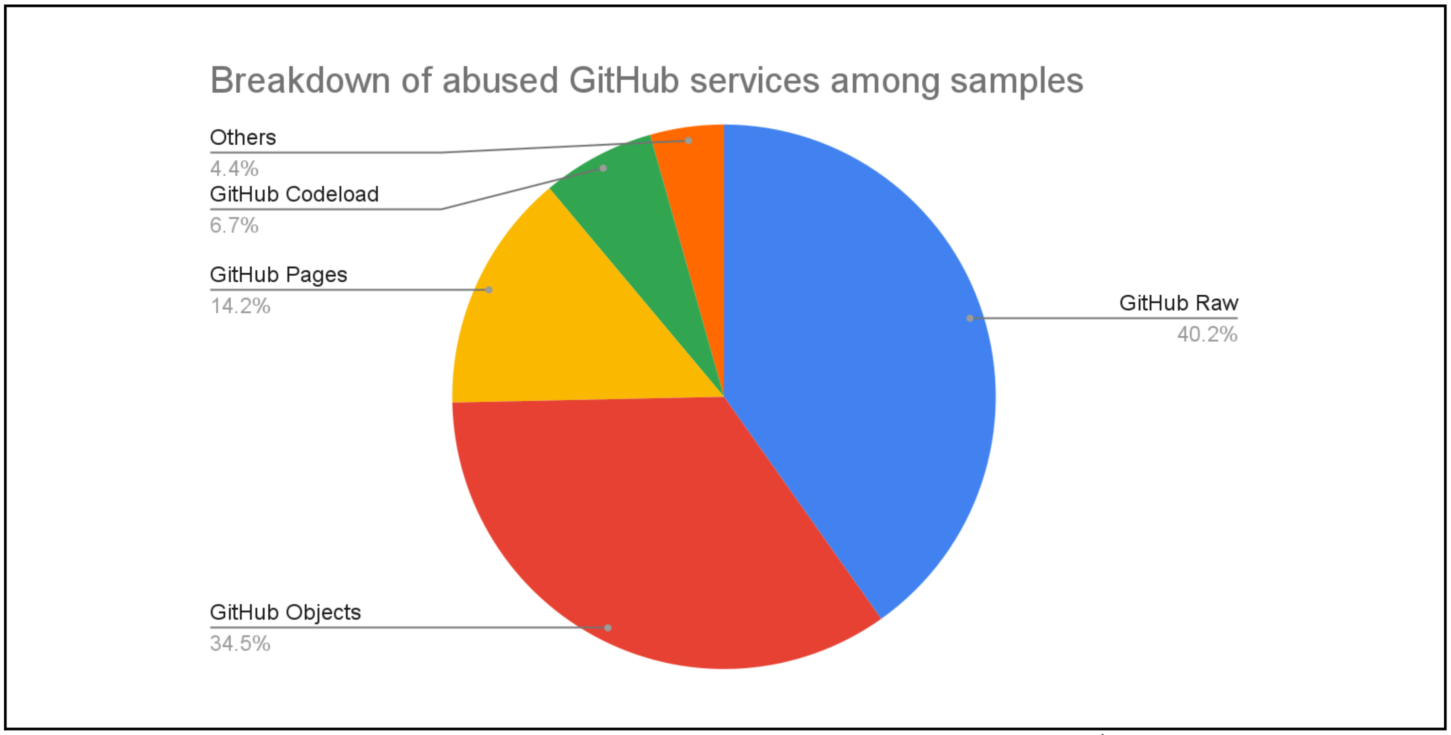


Figure 1: *Breakdown of abused GitHub services among samples from March to November 2023 (Source: Recorded Future)*

GitHub Codespaces

GitHub Codespaces offers an instant, highly customizable, cloud-based integrated development environment (IDE) that utilizes Docker containers to equip developers with essential languages, tools, and utilities. It was made freely accessible (with a range of limits) to the public in November 2022 after initially only being available to select users. Codespaces instances are isolated virtual machines (VMs) hosted on Azure that are accessible through GitHub CLI, web browsers, and various integrated IDEs, including Visual Studio Code and JetBrains.

While Codespaces provides a wide range of features, one that has been demonstrated to be susceptible to abuse for malware delivery is its capacity to publicly share forwarded ports. These publicly forwarded ports can be accessed via a specific URL without authentication. What sets this apart from direct GitHub hosting is that the abused environments are less likely to be flagged as malicious or suspicious due to the unique subdomains based on unique identifiers (see **Figure 2**).

```
<username>-<codespace>-<random_identifier>-<exposed_port>.preview.app.github.d
ev
```

Figure 2: *Format of publicly exposed Codespaces URL (Source: Trend Micro)*

GitHub Actions

GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform offered by GitHub. With GitHub Actions, developers can automate a range of workflows and tasks related to their software development directly within their GitHub repository, including the testing of pull requests or the deployment of merged pull requests into production.

A GitHub Actions workflow is triggered in response to specific events in a repository (such as the opening of a pull request by a developer or an automation tool like Dependabot). A workflow consists of 1 or more jobs that can be executed sequentially or concurrently. Each job operates within its dedicated virtual machine runner or container and comprises 1 or more steps. These steps can either run a script or execute an action, which is a custom application designed to perform complex and often repetitive tasks (see **Figure 3**).

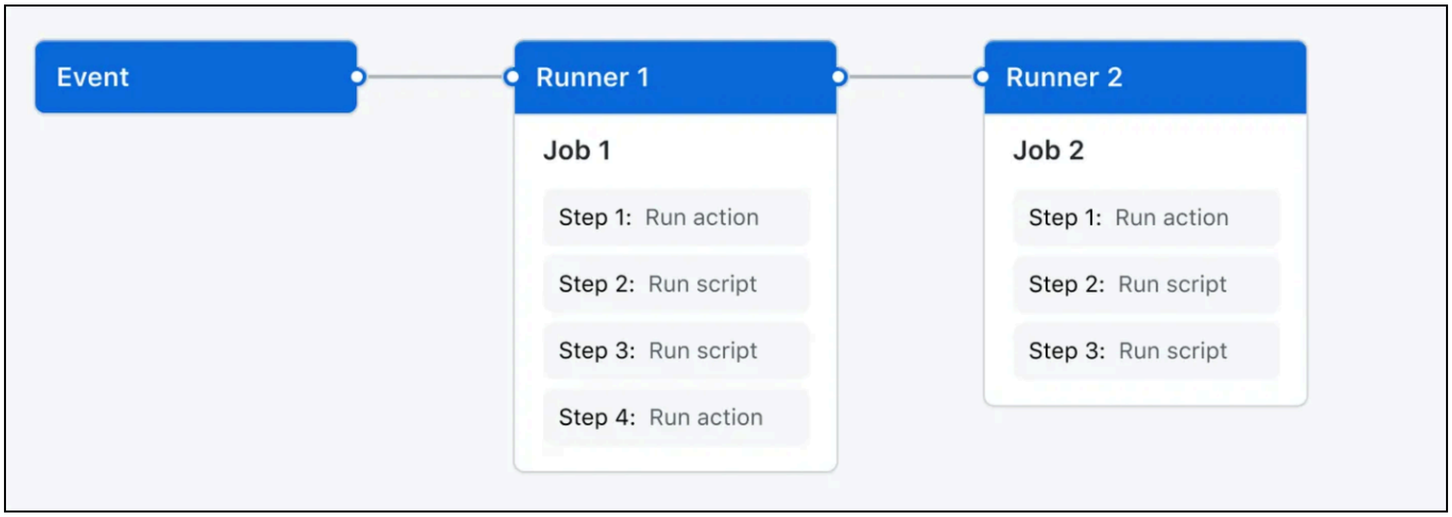


Figure 3: *Components of GitHub Actions (Source: GitHub)*

Legit Security, a software supply-chain security firm, recently issued an advisory warning about a vulnerability known as "artifact poisoning". This vulnerability potentially affects software projects using GitHub Actions, as it initiates the build process upon detecting alterations in software dependencies, potentially resulting in infection and payload delivery. In another recent scenario, threat actors added a new GitHub Action file to the repository's workflows. This file is triggered upon each code-push event and subsequently transmits GitHub secrets and variables to a designated URL.

GitHub Pages

GitHub Pages is a hosting service for static websites that takes HTML, CSS, and JavaScript files directly from a GitHub repository. Optionally, it can process these files through a build pipeline before publishing the website. There are 3 types of GitHub Pages sites: project, user, and organization. Project sites are linked to a particular project hosted on GitHub (such as a JavaScript library). User and organization sites are associated with specific GitHub accounts. If no custom domain is configured, user and organization sites are accessed at the domains in **Figure 4**.

```
<username>.github.io
<organization>.github.io
```

Figure 4: *Format of domains for GitHub Pages user and organization sites (Source: GitHub)*

The source files for a project site are located within the same repository as the project itself. If no custom domain is configured, project sites are accessed at the domains in **Figure 5**.

```
<username>.github.io/<repository>
<organization>.github.io/<repository>
```

Figure 5: *Format of URLs for GitHub Pages project sites (Source: GitHub)*

GitHub Raw

The domain *raw.githubusercontent.com* is used to retrieve raw versions of files stored within GitHub repositories. For instance, if the page represents a Ruby installation script, the domain will provide an actual Ruby installation script that can be interpreted.

```
raw.githubusercontent.com/<username>/<repository>/<branch_name>/<path>
```

Figure 6: *Format of URL for accessing raw files (Source: GitHub)*

GitHub Objects

The runner, which is responsible for executing a job within a GitHub Actions workflow, can take 2 forms: GitHub-hosted or self-hosted. GitHub-hosted runners provide a convenient and swift method for running workflows, while self-hosted runners offer a highly customizable approach within specific environments. When employing self-hosted runners, GitHub requires a reliable mechanism for distributing files and data to these runners (such as for runner version updates). This is where *objects.githubusercontent.com* steps in, serving as a content delivery network (CDN) dedicated to efficiently and reliably delivering the essential resources required for the successful execution of workflows by self-hosted runners.

```
objects.githubusercontent.com/github-production-release-asset-*
```

Figure 7: *Format of URL for accessing GitHub Objects (Source: GitHub)*

GitHub Gists

GitHub Gists offers the ability to store and distribute code snippets without the need to set up a complete repository. Gists can accommodate various content types, such as code strings, bash scripts, markdown documents, text files, and other small data pieces. Each gist functions as a Git repository, allowing forking and cloning. If you are logged into GitHub while creating a gist, it will be linked to the logged-in account.

Gists can be either public or secret. Public gists are displayed in the Discover section, allowing users to explore and search for newly created gists. In contrast, secret gists do not appear in Discover and are not searchable unless the viewer is logged in and happens to be the author of the secret gist. However, it's important to note that secret gists are not entirely private, as the URL of a secret gist can be accessed by anyone who knows the URL.

```
gist.github.com/<username>/<gist_id>
gist.github.com/<username>/<gist_id>/raw
```

Figure 8: *Format of URL for GitHub Gists of a specific user (Source: Recorded Future)*

Owing to their ease of setup, gists have often been abused for malicious infrastructure ever since their introduction. For example, as early as 2019, gists were used as an easy way to send commands to the SLUB backdoor, as reported by Trend Micro. More recently, Hydra abused gists for payload delivery.

GitHub Codeload

The *codeload.github.com* subdomain serves raw files (such as Actions files) and complete GitHub repositories as ZIP archives without going through the regular web interface. This subdomain is commonly used by developers and users for caching for both manual and automated retrieval of code from GitHub repositories. Although it is possible to utilize the Codeload URL directly, it is generally advised to use the API URL, which provides more robust authentication handling.

```
codeload.github.com/<username>/<repository>/<path>
```

Figure 9: *Format of URL for GitHub Codeload (Source: URLScan)*

GitHub API

The *api.github.com* domain is used for accessing GitHub's REST API, which provides programmatic access to a wide range of GitHub features and data. Developers and applications use this API to interact with GitHub programmatically and to integrate GitHub functionality into their applications. It can be used to create, read, update, or delete repositories, issues, pull requests, and more. It also provides access to user and organization information, and even fields like the repository description can be abused for payload delivery. Access to the GitHub API typically requires authentication using tokens or OAuth.

```
api.github.com/repos/<username>/<repository>/
```
Figure 10: *Format of URL for GitHub API (Source: GitHub)*

Threat Analysis

Threat actors can abuse LIS like GitHub for their infrastructure in 4 primary conceptual categories: payload delivery, DDR, full C2, and exfiltration. These infrastructure schemes are not mutually exclusive, can be combined, and do not rely on the exploitation vulnerabilities in GitHub services but instead leverage existing features.

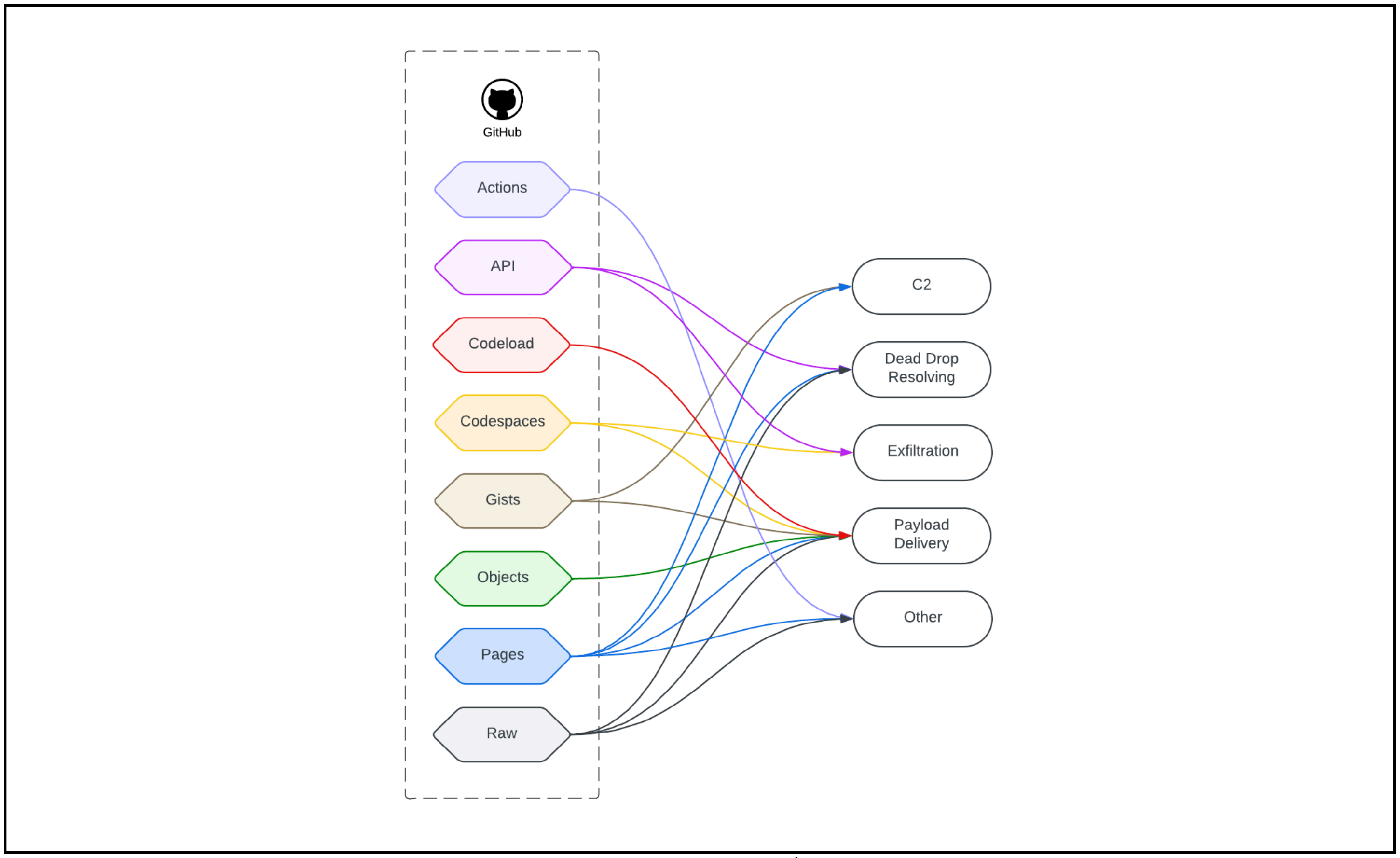


Figure 11: *GitHub services abused by malicious infrastructure schemes (Source: Recorded Future)*

GitHub for Payload Delivery

Unsurprisingly, payload delivery is the most prevalent infrastructure scheme among the GitHub abuse instances and has been consistently observed for several years by both cybercriminals (such as BUHTRAP or TeamTNT) and state-sponsored groups (such as Gaza Cybergang or APT37). Netskope, which categorized sources into cloud and web categories, found that GitHub accounted for 7.6% of all malware downloads originating from cloud-based applications in 2022. Abuse instances can be broadly categorized into staging and infection-focused scenarios.

Staging

Staging via GitHub services refers to a procedure for delivering malicious code to a target system, regardless of the initial infection, which may occur via an alternative attack vector, such as (spear)phishing or vulnerability exploitation. In this context, malicious code can include legitimate open-source projects that are used as part of the attack chain (such as when Hydra loads a Tor client) or code and data owned by threat actors (such as when Mustang Panda stored an encrypted Google Drive token on GitHub). Recent examples include:

- In **January 2023**, the Qualys Threat Research Unit reported on a campaign that used Excel spreadsheets as bait to distribute the off-the-shelf malware known as BitRAT. The Excel spreadsheet includes an obfuscated Macro that drops an INF payload in the `%temp%` directory and executes it through `advpack.dll`. The INF file contains a hex-encoded second-stage DLL payload, which is decoded using `certutil` and then run via `rundll32`. This DLL retrieves the final BitRAT payload from GitHub and executes it. Once loaded, BitRAT communicates with its C2 via the TLS 1.2 protocol. The corresponding GitHub account, which was not specified in the report, seemed to be a "throwaway" account established exclusively for hosting payloads.
- In **March 2023**, Uptycs discovered a new infostealer dubbed HookSpoofer. As part of its infection chain, HookSpoofer loads 2 auxiliary DLLs, `DotNetZip.dll` and `AnonFileApi.dll`, as raw files from the StormKitty repository owned by the user LimerBoy, which has since been archived. LimerBoy's repositories have also been seen being used in other instances, such as to load code for audio recording in an attack chain involving QwixxRAT.
- In **March 2023**, Deep Instinct reported on a DuckTail campaign targeting individuals and businesses operating on Facebook's Business Ads platform. The infection begins with a malicious LNK contained within an archive, which triggers PowerShell to download a payload hosted on GitHub alongside other publicly accessible file-sharing platforms like Discord or Google. The GitHub accounts involved follow a similar naming convention and include sparta137, rebel137, and putzer1001. The final payload exfiltrates data via the Telegram API.
- In **March 2023**, @iamdeadlyz reported on a fake project dubbed PureLand, likely impersonating the video game Rune Teller. The fake project's domain automatically redirects to a designated landing page where, after entering an access code, the victim is provided with 3 Dropbox links to download an executable, an archive, and a MacOS installer package. Once the executable is executed, it retrieves a GitHub URL from a Pastebin link and proceeds to download `pureland.7z`. Utilizing `7zr.exe` and the password "pureland", it extracts an additional executable named `pureland.exe`, which turns out to be RedLine Stealer.
- In **May 2023**, AhnLab reported that operators of RecordBreaker (the successor to Raccoon Stealer) can include one or more executable payload URLs in its configuration (plaintext and config line starts with `ldr_`), with GitHub URLs being abused to host these additional payloads, among others. Following its theft and exfiltration activities, RecordBreaker proceeds to download and execute these additional payloads. One of the downloaded files observed by AhnLab was `GUI_Modernista.exe`, a program designed to download various crack files. This deceptive tactic leads users to mistakenly perceive the downloaded content as a typical crack program,

thereby making it more difficult to notice the malware infection. Similar observations have been made in relation to the Vidar stealer.

- In **June 2023**, Morphisec Labs reported on a phishing campaign targeting US law firms that used PDF attachments and lured victims into clicking on an embedded icon to initiate the process. Upon clicking on the icon, the victim is redirected to the final URL, which prompts the victim to enter a PIN and downloads a VBScript. Following deobfuscation, a PowerShell script retrieves the GuLoader shellcode from the GitHub Pages site, base64-decodes it, and divides it into the decrypting and encrypted shellcode. The shellcode is invoked by providing it as a callback function to `CallWindowProcA`, along with the encrypted shellcode and `NtProtectVirtualMemory` as arguments, subsequently enabling the shellcode to download REMCOS RAT from *quickcheckx[.]github[.]io*.
- In **August 2023**, a phishing campaign targeting Mexico and distributing Metamorfo (also known as Casbaneiro) was discovered by @0xToxin. The phishing email lured victims into clicking on a URL that resulted in the automatic download of a RAR archive. Subsequently, a CMD script executes a PowerShell command and downloads another PowerShell script from *raw[.]githubusercontent[.]com*. The PowerShell script downloads an archive from another URL, extracts its contents, and establishes persistence by creating an LNK file in the startup folder. Upon reboot, the LNK file executes the Metamorfo DLL and establishes a C2 connection.

Infection-Focused

In contrast, in infection-focused scenarios, GitHub assumes a pivotal role in the actual initial infection process, achieved through methods like repository poisoning or the creation of fake repositories. Examples include the following:

- In **April 2022**, Netskope Threat Labs identified a RedLine Stealer campaign spread via YouTube, employing a fake bot to purchase Mystery Box NFTs from Binance. The video description leads victims to download a ZIP file containing the fake bot from a GitHub repository named NFTBOT owned by NFTSupp via *raw[.]githubusercontent[.]com*. The `README.md` file provides instructions for running the fake NFT bot, including the installation of Microsoft Visual C++. This step is likely required because RedLine is built in .NET. Upon closer examination, it was discovered that the NFTBOT repository hosted a variety of other RedLine Stealer loaders.
- In **October 2022**, researchers at the Leiden Institute of Advanced Computer Science discovered thousands of repositories on GitHub offering fake proof-of-concept (PoC) exploits for diverse vulnerabilities, with some of them even incorporating malware such as Houdini RAT or Cobalt Strike. According to their research, the risk of malware infection instead of acquiring a genuine PoC could be as high as 10.3%, excluding confirmed fake PoCs and prankware.
- In **August 2022**, a hacker using the alias "Pl0xP" targeted approximately 35,000 of the most popular GitHub repositories, cloning them and inserting malicious code. However, no actual malicious code appears to have been merged back into the original repositories, and GitHub promptly removed all instances of this code, which was crafted to pilfer sensitive data, including cloud credentials, passwords, environment variables, documents, and more. This attack, because it was a form of dependency confusion, posed a potential threat to developers who

might unknowingly use the malicious GitHub repositories without proper verification of the software's source. The threat intensifies as threat actors can bolster their perceived credibility by manipulating metadata (for example, falsifying metrics like commit counts) or posing as Dependabot contributors.

- In **September 2022**, CrowdStrike reported on an instance of waterholing activity involving GitHub where a threat actor exploited a misconfiguration in GitHub repositories to gain code execution and initial access to thousands of hosts worldwide. The victims clicked a link in a legitimate repository's wiki, which directed them to a redirection URL service using Linkify. This service then directed the download toward an unknown GitHub account hosting the malicious file. Upon further inspection, the wiki of the legitimate repository showed multiple instances where new GitHub accounts had successfully edited it to redirect users to malware by altering the main download link.

Researchers at NTT Communications have also described a hypothetical threat where a threat actor might post and share custom Actions on the GitHub Marketplace involving malicious code such as a hidden backdoor. The threat arises from users unwittingly incorporating these malicious Actions into their workflows and thereby becoming infected with malicious code embedded in the custom Actions. While this threat has not been observed in the wild, defenders should keep it on their radar as a possible vector for staging and infection.

GitHub for DDR

Similar to other platforms that facilitate convenient access to structured data, GitHub can be abused for DDR (T1102.001). Users can post URLs, domains, or IP addresses in various formats, including encrypted files. Even when shared openly without encryption or obfuscation, there is minimal immediate risk of data removal, as it poses a challenge for the platform operator to discern the malicious intent of these addresses without additional context or information. Services with a high chance of data remaining accessible when accessed by malware are ideal for DDR, and GitHub is a prime example of such a service. Examples include the following:

- In **September 2023**, Insikt Group identified a campaign by BlueDelta that employed a multi-step attack chain designed to target specific locations and evade sandbox analysis. It leveraged GitHub Pages hosting an HTTP redirect script directing to the free mock API services *mockbin[.]org*. This involved mimicking Windows Update identifiers and Microsoft domains, a tactic previously observed in BlueDelta-associated campaigns.
- In **March 2023**, SentinelOne reported on the SmoothOperator supply-chain attack attributed to the Lazarus Group and known for abusing GitHub (for example, with VSingle). The attack involved retrieving icon files from a dedicated GitHub repository, which were appended with base64-encoded chunk data following the "$" character. The malware searches for the "$" character within the ICO file and extracts the subsequent data bytes. By decoding and decrypting the bytes, the malware obtains the actual C2 URL.
- In **February 2023**, Fox-IT reported on Hydra, also known as BianLian, which was one of the most active mobile banking malware families in 2022. One of the variants Fox-IT came across was

designed to fetch a file from GitHub that contained a JSON object encoded in base64 with a list of C2 servers.

- In **December 2022**, Secureworks reported on Drokbk, a malware associated with COBALT MIRAGE. The malware uses the GitHub API to search for a specific repository, allowing it to find both the associated GitHub account and the request used to locate the actual C2 server. Storing the response within the `README.md` file hosted on the GitHub account grants the threat actor a level of resilience against the potential takedown of their GitHub account, as they can establish a new account with a repository name matching the previous one.
- In **February 2023**, Trend Micro reported on ShellBox, a stager written in C++ abusing Dropbox for payload delivery. To obscure its operations and enhance stealth, ShellBox does not have an Access Token embedded in its code but first accesses a hardcoded GitHub repository by “lettermaker” to acquire the URL hosting the Dropbox Access Token. Subsequently, it retrieves and decrypts the contents from the URL to obtain the Dropbox Access Token.

While these examples illustrate the various ways GitHub can be abused for DDR, numerous other threat actors and malware families have been observed abusing the platform, often with slight variations in their implementations.

GitHub for Full C2

Full C2 using LIS refers to a scenario where threat actors and malware do not engage in direct communication but rely on an “abstraction layer” such as GitHub. Apart from a few open-source projects that primarily demonstrate the feasibility of abusing GitHub for full C2, such as GithubC2 or Dystopia, there are some noteworthy real-world instances, including the following:

- In **July 2015**, Mandiant reported on APT29’s backdoor HAMMERTOSS, which contains an algorithm that generates social media handles on a daily basis and instructs the malware to visit a designated handle on a specific date. If registered for that specific date, the handle posts a GitHub URL combined with a hashtag. After HAMMERTOSS retrieves the daily GitHub Pages URL, it downloads its content, including image files. While the images look innocuous, they contain appended and encrypted data that can be decrypted using a hard-coded key provided through the hashtag from the post, which consists of the byte offset and the decryption key. The decrypted data can contain instructions for actions like data uploads, command execution (including PowerShell), and reconnaissance tasks on the victim network, among others.
- In **January 2022**, Malwarebytes analyzed a campaign by Lazarus Group targeting job seekers, marking the first instance of the threat actor leveraging GitHub for full C2 purposes. The attack begins with the execution of malicious macros embedded in a lure Word document. The malware injects `core_module.dll` into `explorer.exe`, enabling C2 communication and module loading. The absence of string encoding simplifies the analysis as it includes the hard-coded username, repo name, directory, and token to make HTTP requests to GitHub by the username "DanielManwarningRep". The malware fetches files from the `images` repository, which contain the malicious modules. Once executed, the result is uploaded to the `metafiles` directory via an HTTP PUT request, and it is named after the timestamp when the module was run. This file,

committed to the repository, holds the results of the commands executed by the module on the target system.

- In **March 2019**, Trend Micro reported on the custom backdoor SLUB, which derives its name from the combination of SLack and githUB. Utilizing the statically linked boost library, the backdoor retrieves a specific gists snippet from GitHub and parses it in search of commands to execute. It exclusively processes lines that start with "^" and end with "$", disregarding all other lines. The output of the executed commands is posted in a specific workspace of a private Slack channel. Note that the setup inadvertently prevents the threat actor from issuing commands to specific targets, as each infected computer executes the enabled commands in the gists snippet upon inspection. Analyzing executed commands in the Slack channel provides insights into the threat actor's activities and preferences.

Overall, despite its higher flexibility than other LISs such as Telegram or Discord, full C2 implementations involving GitHub are less common compared to other infrastructure schemes. Functional constraints imposed by GitHub’s services when compared to traditional servers may be one reason. In addition, due to GitHub's nature as a version control system, malware operators might also be concerned about the potential exposure of their activities and interests, even in cases where encryption or encoding is used.

GitHub for Exfiltration

Although GitHub can serve as an exfiltration proxy (T1567.001), this usage is less common in comparison to other infrastructure schemes. Reasons for this include:

- Utilizing other LIS is potentially perceived as more efficient by threat actors.
- Accessing data from open GitHub services appears less suspicious than actively posting information to GitHub.
- GitHub imposes file-size limits in repositories, issuing warnings for files exceeding 50 MB, while files added via the browser are restricted to 25 MB.
- Storing large files in GitHub's Large File Storage (LFS) incurs additional charges.

Regardless of these limitations, there are instances where GitHub has been employed for exfiltration purposes. Examples include:

- PowerShell Empire, a post-exfiltration framework released in 2015, has a GitHub exfiltration feature. Although the original project is no longer supported, its code has been forked and is currently maintained by BC-Security.
- In **March 2023**, the North Korean group Kimsuky appears to have uploaded gathered victim data to a GitHub repository through the GitHub API in the form of log files.
- In **January 2023**, Trend Micro reported on Rust-based infostealers leveraging exposed ports on a Codespaces instance to exfiltrate data from infected machines. More specifically, the malware initially uploads a ZIP file of stolen data to *gofile.io*, obtains the *gofile.io* URL, and then compiles

victim information, including the *gofile.io* URL, into a JSON file. This JSON file is subsequently sent via a POST request to a GitHub Codespaces URL on port 8080 without authentication.

Though it was not employed explicitly for exfiltration, Mustang Panda leveraged a GitHub repository during the exfiltration process to obtain an encrypted token, subsequently utilizing it to exfiltrate data to Google Drive. A related topic is the risk of source code exfiltration from internal GitHub repositories to personal or unsanctioned repositories. This is pertinent not only in the context of insider risks but also in scenarios where threat actors aim to pilfer intellectual property by targeting source code.

GitHub for Other Schemes

In addition to the 4 main infrastructure schemes described above, GitHub services are also abused in other ways for infrastructure-related purposes. For example, GitHub Pages have frequently been abused as phishing hosts or as traffic redirectors, allowing the actual phishing pages to remain operational for a longer duration before they are removed. Phishing hosts abusing GitHub Pages can be detected using heuristic patterns on sources such as URLScan.

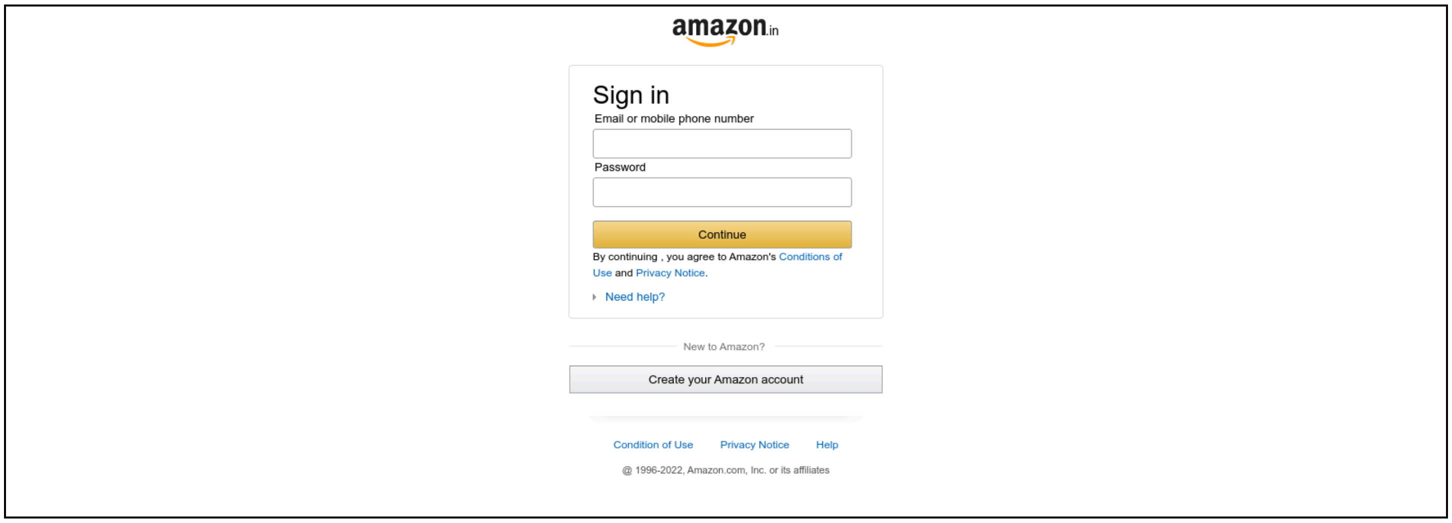

amazon.in
Sign in
Email or mobile phone number
Password
Continue
By continuing , you agree to Amazon's Conditions of Use and Privacy Notice.
Need help?
New to Amazon?
Create your Amazon account
Condition of Use
Privacy Notice
Help
@ 1996-2022, Amazon.com, Inc. or its affiliates

Figure 12: *Suspected phishing page hosted on github.io (Source: URLScan)*

ESET also found some variants of Crutch v3, a backdoor associated with the Turla APT group, employing a hardcoded GitHub repository as a fallback channel in case the Dropbox account used for C2 was taken down. The threat actors may have opted for GitHub because they were expecting a lower account takedown risk compared to traditional server setups.

GitHub Alternatives

GitHub is just one of several code repositories and version control platforms. With 77% of developers surveyed by the Czech software company JetBrains regularly using GitHub, it is likely the most widely used platform, compared to 40% for GitLab and 25% for BitBucket. These other platforms, offering similar advantages to threat actors such as blending in with benign network traffic, are frequently abused for malicious purposes as well, primarily for payload delivery. Threat actors conduct thorough reconnaissance, adapting to the service usage patterns of their victims.

GitLab

For example, Check Point Research recently reported on a campaign where websites posing as legitimate software utilities redirected users to a GitLab page hosting numerous dotRunpeX-infected programs, eventually leading to RedLine Stealer infections. Similarly, Cyble identified a GitLab repository masquerading as "citrixproject" being used to distribute AresLoader, resulting in LummaC2 and IcedID infections. IcedID has been observed being loaded through GitLab in other instances as well.

BitBucket

BitBucket, a Git-based source code repository hosting service owned by Atlassian, is also frequently abused for payload delivery. For example, in a campaign observed by Cybereason, the downloading of cracked software resulted in AZORult and Predator The Thief infections, followed by infections by various other malware families. Kaspersky has also recently discovered a modular framework, called StripedFly, leveraging BitBucket, among other code repositories, to fetch updates and additional modules disguised as firmware binaries for “m100” devices. Other recent examples include NodeStealer leveraging BitBucket in addition to Dropbox and GitLab.

Codeberg

Most recently, Cado Security reported on the first instance of Codeberg (*codeberg[.]org*), an alternative Git-hosting platform that offers comparable functionality to GitHub, being utilized to disseminate Qubitstrike. However, as of October 20, 2023, just 363 files submitted to VirusTotal communicated with *codeberg[.]org* when opened or executed, and it is important to note that this communication may not necessarily be indicative of malicious activity.

Gitee

Gitee, the Chinese equivalent of GitHub launched by the Shenzhen-based OpenSource Consensus Technologies (OSChina) community in 2013, has also been abused for payload delivery in the past, as noted by Cisco Talos. Although the precise degree of abuse is uncertain, the Chinese government's suspected censorship measures appear to have forced Gitee to restrict access to its code in 2022, possibly making it less attractive to threat actors.

Detection Strategies

Abusing platforms like GitHub for malicious purposes serves as one of several tactics for evading detection, inherently complicating the detection process. Detecting such abuse within a specific environment depends on various factors, including the availability of logs and other resources, the organizational structure, and risk tolerance, among others. There is no one-size-fits-all solution; instead, a combination of detection strategies should be considered in addition to file-based detections (such as YARA rules). While certain detection methods can be applied in a production environment, others are better suited for threat hunting.

Context-Based Detections

A context-based detection strategy is based on the principle that if only specific parts within a corporate environment require interaction with particular GitHub services, any network traffic to these services from parts beyond those designated environments is regarded as suspicious. For example, it might be that only developers should access the GitHub API, making any network traffic to it from other departments suspicious. A context-based detection strategy, though highly effective, demands a comprehensive understanding of the organizational environment and requirements. With respect to the aforementioned example, it is essential to maintain a list of authorized developers, VLANs, and internal IP addresses for GitHub API access, possibly leading to significant overhead in practice.

Service-Based Detections

A service-based detection strategy is based on the idea that only specific (sub-)services of GitHub are used within a corporate environment. Examples of situations suitable for service-based detections or even blocking in some cases may include:

- An organization may have an internal Git Enterprise server, which makes various external GitHub services theoretically unnecessary.
- An organization may use self-hosted runners that connect to various GitHub hosts for job assignments and updates. While some hosts are needed, others can be blocked when not in use.
- An organization with minimal or no legitimate use for GitHub Codespaces should consider flagging POST and GET requests to *app.github.dev*.

Similar to the context-based detection strategy, knowing the environment is key to implementing the service-based detection strategy. It is important to keep in mind that in an effort to blend in, threat actors perform reconnaissance and likely consider the specific service usage of their victims.

Log-Based Detections

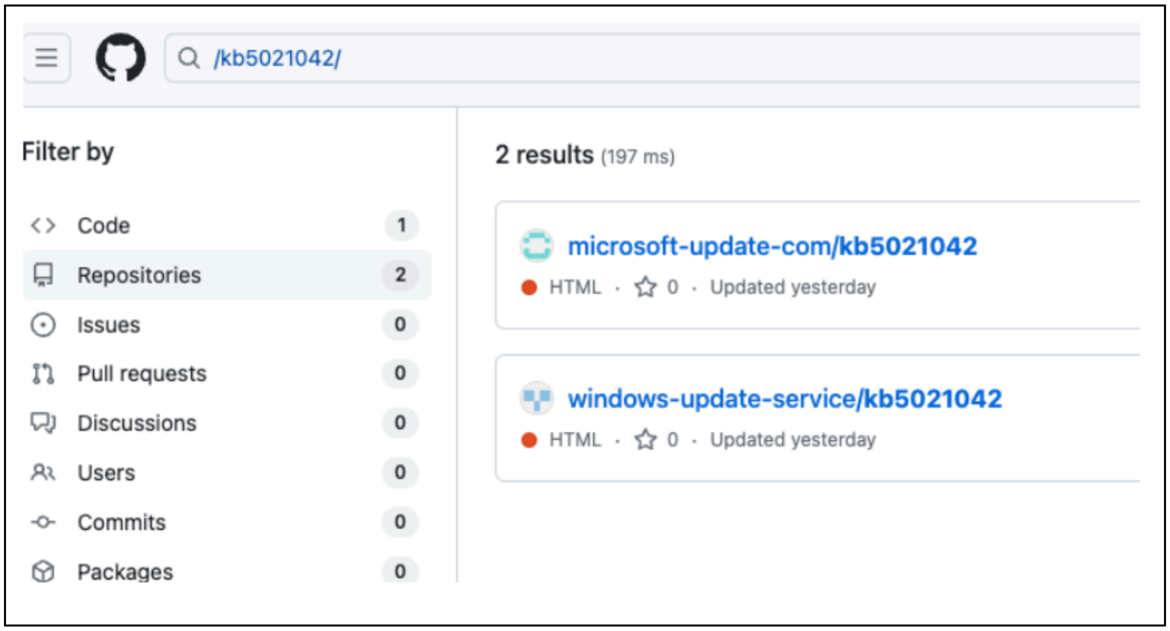
While the connection to GitHub may lack inherent differences between malware and legitimate connections, the contextual interactions of a system with GitHub services can offer valuable insights into the potentially suspicious nature of the connection. Suspicious connections can be identified through log-based detections, utilizing both proxy and audit logs. Examples of log-based detections include:

- Malware may utilize specific living-off-the-land binaries (LOLbins), such as `certutil` with flags like `verifyctl` or `urlcache` (in the case of BazaLoader or Agent Tesla, for example), to retrieve payloads from GitHub. The execution of `certutil` with these flags, particularly in connection with GitHub domains (and other LIS domains), can be flagged using Sigma rules or similar frameworks. Similar rules can be applied to other LOLbins used for downloading, such as `wget` and `bitsadmin` (1, 2).
- In a similar but more generic vein, detecting all non-browser executables making DNS requests to GitHub domains and other LIS domains can help prevent DDR, among others. Prior to implementing these detections, it is crucial to validate the legitimacy of GitHub usage for other services within the environment and subsequently exempt them from the detections.
- Threat actors may use the Git commands (which underlie GitHub Desktop as well) for data exfiltration to their repositories. Detection rules can be created for specific commands such as remote, add, or push, among others involving GitHub domains other than the enterprise's own GitHub instance. To mitigate the risk of false positives, it is important to filter out commands and their sub-executions related to Git installation, among others.
- Proxy logs can detect specific URL patterns with specific file extensions such as `.exe`. A more comprehensive list of such URL patterns can be found here. Note that even seemingly innocuous file types, like raw Markdown files (such as `README.md`), can be weaponized. For example, a simple string replacement may transform Markdown content into executables.

Furthermore, it is worth exploring time-based detections, such as identifying regular check-ins, which can be especially valuable for uncovering full C2 schemes. Ultimately, determining what should be labeled as suspicious relies on the unique characteristics of the environment, and there is an ongoing balancing act between blocking, flagging, and disregarding potential threat signals.

Detections Based on LIS Combinations

As found in the first part of this series, approximately 70% of malware families that supported the abuse of LIS as part of their infrastructure abused more than 1 LIS. In other words, the abuse of LIS frequently involves the concurrent abuse of multiple LISs, creating opportunities for network detection through the identification of these LIS combinations. For example, BlueDelta used GitHub Pages containing an HTTP redirect script pointing to the free mock API services *mockbin[.]org* (see **Figure 13**).



microsoft-update-com/kb5021042
HTML · 0 · Updated yesterday

windows-update-service/kb5021042
HTML · 0 · Updated yesterday

Figure 13: *BlueDelta repositories mimicking Windows Update identifiers (Source: Recorded Future)*

The challenges include initially identifying LIS abuse combinations, legitimate use cases (such as if a developer visits GitHub and then shares internal documents via Google Drive) that can result in false positives, ongoing changes in the threat landscape, and the existence of less common combinations where investing in detection development may not be cost-effective. Given this context, it is a reasonable starting point to prioritize the most common LIS abuse combinations.

Network-Based Detection

As shown above, payload delivery and DDR represent the prevailing infrastructure schemes observed in instances of GitHub abuse. In both of these infrastructure schemes, the malware may establish communication with other malicious infrastructure at some stage during its operation. While this strategy helps defenders in detecting infections through traditional C2 feeds, a downside to this approach is that detection may occur too late (after data has already been exfiltrated).

Proactive Threat Hunting

Taking a proactive approach to hunting for instances of GitHub abuse not only provides insights into what to search for within a corporate network but also provides a more nuanced understanding of threat actor behaviors. While detections stemming from these hunts may be integrated into production, a substantial portion of threat-hunting activities still involve manual processes. There are various ways to identify instances of GitHub abuse beyond malware sandboxes, including those described below.

Hunting Via GitHub Usernames, Repositories, and Organization Names

To find additional GitHub abuse instances associated with a specific campaign or threat actor, one can pivot via usernames, repositories, organization names, or code snippets, among others. For example, as previously noted, BlueDelta employed various GitHub organizations that hosted repositories with the same name (`kb5021042`), containing an HTTP redirect script pointing to the free mock API services *mockbin[.]org*, which helped to identify additional GitHub organizations (see **Figure 13**). Searching for specific code snippets is also possible but limited to the advanced search interface.

Another example is StormKitty, a now inactive repository owned by the user LimerBoy, which was used by infostealer malware to load various payloads. A more thorough examination of the user's repositories reveals a multitude of suspicious payloads. Overall, identifying additional related repositories, organizations, and users can contribute to a deeper understanding of whole attack chains, including capabilities, techniques, and motivations.

Searching for code snippets may include generic elements (for example, the function `eval(base64_decode` used for PHP code execution attacks), leading to the potential discovery of exploit code, or distinct pieces of code linked to specific malware strains (such as `a6f452ec3293d7fb72c5b677257b20ec`, a password associated with AlfaShell), threat actors, or campaigns.

Hunting Via Website Scanning Tools

Another way of identifying GitHub abuse instances is through website scanning tools, including the Recorded Future® Intelligence Cloud, which allows for identifying malware-hosting sites associated with GitHub domains (see **Figure 14**). Similarly, it is possible to filter these searches by specific keywords likely used to impersonate particular services.

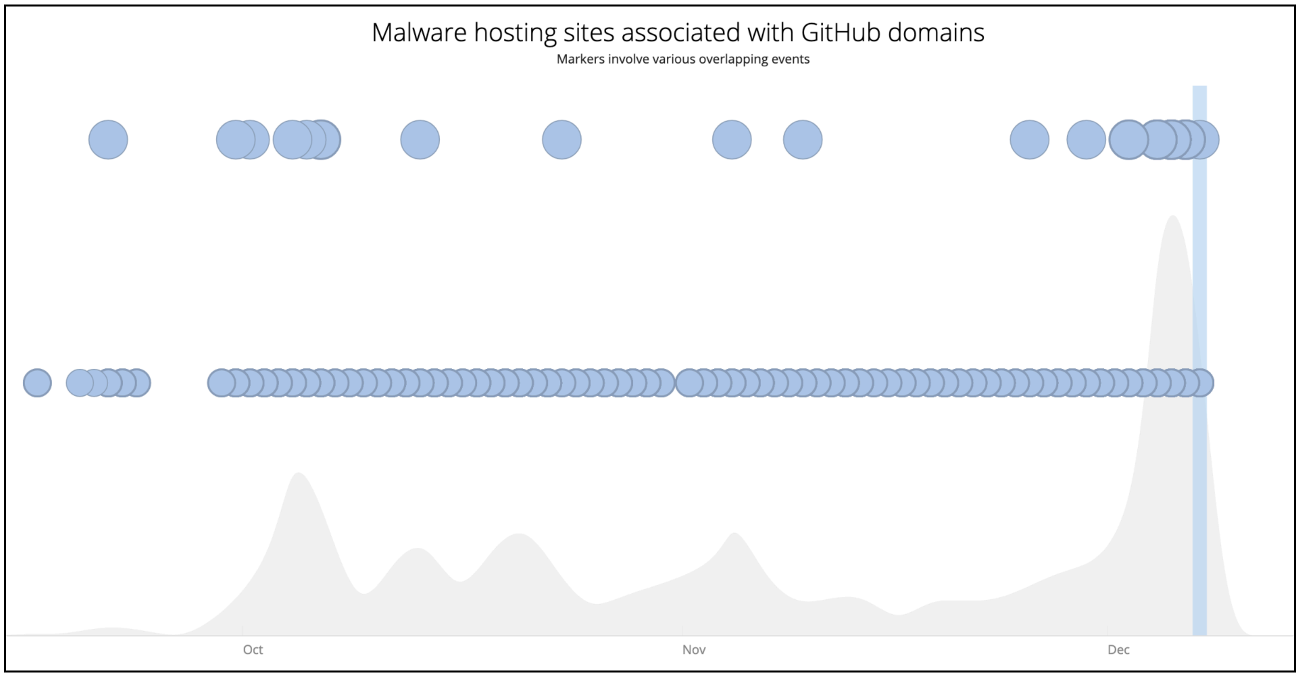


Figure 14: *Malware hosting sites associated with GitHub domains (Source: Recorded Future)*

One of the GitHub Pages used by GuLoader to load additional payloads returned a specific message ("get out!") when being scanned using URLScan. The message itself or the hash of the DOM can be used for hunting, potentially identifying new GitHub Pages used for payload delivery.

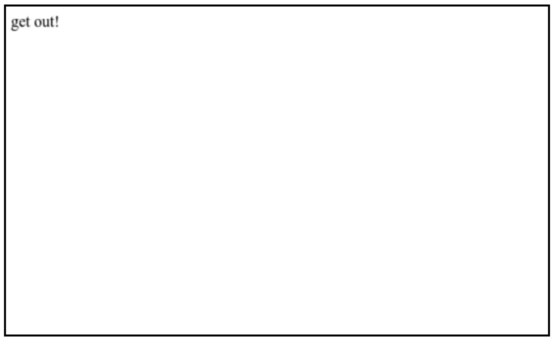

get out!

Figure 15: *Response of queried GuLoader-associated GitHub page (Source: Recorded Future)*

Using URLScan, one could also search for GitHub Pages that include specific keywords (such as those that appear on legitimate login pages of interest), allowing for the detection of potential GitHub-based phishing hosts (see **Figure 16**).

```
(text.content:"sign in" OR text.content:"log in" OR text.content:"login")
AND task.url:/.*github\.io.*/ AND verdicts.score:>30
```

Figure 16: *URLScan query for potential GitHub-based phishing hosts (Source: Recorded Future)*

Similarly, it is possible to conduct searches for GitHub domains commonly used for delivering payloads. To refine the scope of research results or focus on specific threat actors or campaigns, these searches can be combined with other artifacts like URL shorteners. For instance, DuckTail has been observed using *rebrand[.]ly* in conjunction with Dropbox. Investigating the combination of this shortener with GitHub domains may uncover new campaigns (see **Figure 17**). Other approaches may involve specifying file names or naming conventions. For example, Icarus Stealer loaded its payload as `rt.jpg` (1, 2).

```
task.url:/.*.rebrand\.ly.*/ AND page.url:/.*github.*/ AND files.filesize:>0
```

Figure 17: *URLScan query-specific LIS combination associated with DuckTail operations (Source: Recorded Future)*

One limitation associated with website scanning-based methods is the reliance on others to perform the initial website scan.

Analyzing GitHub Commit History

Analyzing unintentionally exposed GitHub repositories can provide valuable insights for threat hunting and detection purposes, as it unveils previously undocumented aspects of threat actors' attack vectors, motivations, targets, themes, and IOCs.

An illustrative case of this is the exposure of a GitHub repository linked to a member of APT37 in 2023, which had been actively used for over 2 years to stage malicious payloads. This case is particularly noteworthy due to the atypical behavior of a sophisticated group like APT37, which opted to repeatedly use the same repository instead of creating disposable repositories for individual victims or campaigns. Although the exact reasons for the repository's continued existence remain uncertain, one possible hypothesis is that it managed to effectively blend in within the environments it targeted and/or that the organizations affected may have had little motivation to report its presence to GitHub for removal.

Despite the threat actor's consistent deletion of files from the repository, the threat researchers successfully recovered these files. This recovery enabled them to construct a detailed timeline of the threat actor's activities, reaching as far back as October 2020, which was the time of the repository's first commit. In the case of APT37, this analysis not only unveiled complete attack chains but also revealed subtler details, such as the reuse of the same virtual machine to generate multiple payloads. Such information holds valuable implications for future threat-hunting and threat-attribution endeavors. This case also highlights the risks associated with the malicious use of GitHub infrastructure from a threat actor's perspective.

Examining GitHub repositories, especially when they serve for more than DDR or a single payload delivery, can be challenging. Tools like PyDriller and Go-based Herkules can assist analysts in gaining a deeper comprehension of the repository's contents and history and identifying potential connections to other related activities. Other tools, such as GitFive, can assist in gaining insights into GitHub profiles, for example, by identifying potential secondary GitHub accounts. When analyzing GitHub repositories, it is crucial to bear in mind that seemingly “empty” repositories may still contain payloads, such as in the description section.

Mitigations and Recommendations

- **Enhance visibility** by implementing granular monitoring of all web and cloud traffic, coupled with context-aware policies enforced at the instance level.
- **Maintain an up-to-date and comprehensive asset inventory**, clearly delineating authorized users — employees, VLANs, or internal IP addresses — that should access GitHub for their tasks.
- **Tailor the implementation of the discussed detection strategies** to align with your environment's specific requirements, capabilities, and circumstances. Given the complexity of the threat, there is no one-size-fits-all solution.
- **Establish adaptive security policies** that evolve with changing needs and threat landscapes, and, where feasible, implement application allowlisting to enhance control and protection.
- **Protect your GitHub accounts**, as threat actors may not only attack them to steal code but also potentially abuse them as malicious infrastructure for subsequent attacks, taking advantage of their ability to blend in.
- **Integrate scenarios of LIS abuse into routine attack simulations** to continually assess the effectiveness of your infrastructure's detection capabilities.
- **Engage with GitHub to counter known malicious activities** on its platform since it alone has the capability to prevent abuse. This not only enhances your security but also offers a more accurate assessment of the extent of abuse related to LIS in the medium term.
- **Perform proactive threat hunting** to detect novel instances of LIS abuse or to identify LIS with potential for abuse. Recorded Future clients can search and alert on specific patterns associated with malware activity observed on a selection of LIS.

Outlook

This report offers a comprehensive and systematic overview of the abuse of GitHub, the predominant code repository solution, for malicious infrastructure purposes across various malware categories and threat actors. Although abusing GitHub, along with other code repository solutions, is not a novel phenomenon, and quantifying trends proves challenging due to the absence of comparable industry reporting, certain features continue to be highly attractive from a threat actor's perspective. These features include the multitude of services that accommodate various infrastructure schemes, the ability to blend in within most corporate environments, and lower operational overhead and infrastructure costs. When examining future developments related to both GitHub and the broader abuse of LIS, this topic can be analyzed across 3 dimensions: attacker, defender, and the LIS itself as the “medium”.

Attacker: Taking into account the advantages that threat actors currently enjoy and the challenges faced by defenders, we anticipate a continued rise in the abuse of GitHub in the coming years. To elaborate quantitatively and in line with our prior assessments, we expect an increase not only in the proportion of malware families and threat actors abusing GitHub but also in the diversity of GitHub services identified for abuse. Qualitatively, there is a potential for an upswing in sophistication, and we anticipate that APT groups will persist in spearheading advancements in this domain, leading to cascading effects influencing less-sophisticated groups over time.

Defender: Despite defenders facing challenges in addressing the abuse of GitHub and other LIS and the absence of a one-size-fits-all solution, effective mitigation strategies are available. These encompass implementing measures to block or flag malicious usage of specific GitHub services across entire or partial corporate environments, creating log-based detections, developing more advanced detections (such as for suspicious LIS combinations), directing attention to other components within the "triangle of detections", and conducting proactive threat-hunting or leveraging threat-hunting-based intelligence. Ultimately, it is important for defenders to gain an understanding of how specific GitHub services are legitimately utilized by an organization's users and how these services can be abused.

LIS: While defenders can play a part in fighting the abuse by adhering to security best practices, their influence is limited. In the end, the LIS themselves have the greatest impact on the potential for abuse. This involves the formation of dedicated teams committed to detecting and countering system abuses. Additionally, we anticipate more structural shifts in the approach towards LIS abuse. These shifts involve policy adjustments (as when Dropbox imposed limits on unlimited business storage plans), technical modifications (as when Discord adopted temporary file links to prevent malware delivery), or even service shutdowns (as with AnonFiles). Lastly, artificial intelligence, already extensively employed in safeguarding application security, may also become more prevalent in the fight against system abuse in addition to other advanced detections.

Recorded Future®