

Analyzing Void Dokkaebi's Cython-Compiled InvisibleFerret Malware

By By: Kazuki Fujisawa May 22, 2026 Read time: 13 min (3405 words)

Published: 2026-05-22 · Archived: 2026-05-28 02:01:49 UTC

Malware

Void Dokkaebi, a North Korea-aligned intrusion set, has updated its information-stealing malware, InvisibleFerret, shifting its delivery format to evade script-based detections.

Save to Folio

Key takeaways

- Void Dokkaebi (aka Famous Chollima) has migrated InvisibleFerret from readable Python scripts to Cython-compiled binaries, distributing the malware as .pyd files on Windows and .so files on macOS.
- The update gives the intrusion set an additional layer of evasion while preserving InvisibleFerret's core capabilities, including backdoor access, browser credential theft, clipboard monitoring, keylogging, and cryptocurrency wallet targeting. BeaverTail has also expanded beyond its original downloader and stealer role into a broader malware with overlapping functions, including credential harvesting and wallet trojanization.
- The campaign remains especially relevant to software developers, cryptocurrency users, and organizations whose developers have access to wallet credentials, signing keys, CI/CD pipelines, or production systems.
- Defenders should move from script-only detection to binary-aware detection to account for extension modules, embedded artifacts, runtime execution scripts, and browser extension tampering.
- Hunting rules and indicators of compromise (IoCs) are provided below to help identify and mitigate threats associated with Void Dokkaebi.

Introduction

Void Dokkaebi, also tracked as Famous Chollima, is a North Korea-aligned intrusion set that systematically targets software developers who hold cryptocurrency wallet credentials, signing keys, and access to continuous integration/continuous delivery (CI/CD) pipelines and production infrastructure. As [previously documented by TrendAI™ Research](#), the group poses as recruiters from cryptocurrency and AI firms, luring developers into cloning and executing code repositories as part of fabricated job interviews.

TrendAI™ Research observed that InvisibleFerret, a Python-based malware family composed of multiple modules and delivered through the infection chain, has been obfuscated using Cython.

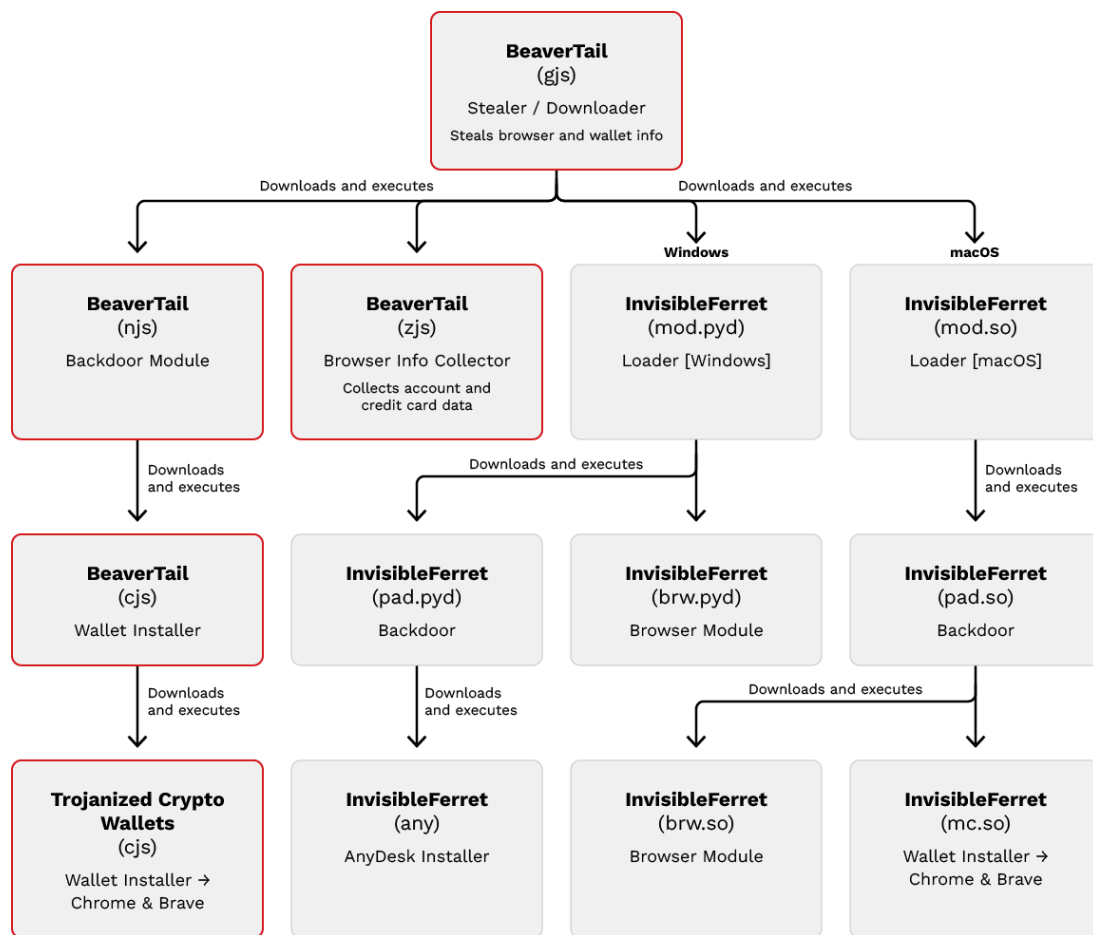
Cython is a tool that translates Python code into C/C++ source code and then compiles into native binaries, which improves execution speed. As a result, InvisibleFerret is now distributed as .pyd files (Python extension modules

in DLL format) on Windows and .so files (shared libraries) on macOS, rather than as plain Python scripts.

Because Cython-generated binaries are not standalone executables but Python extension modules, they cannot run on their own and require a Python script or interpreter to load them. Consequently, the infection chain generates a Python execution script to run the Cython-obfuscated InvisibleFerret.

From a detection evasion perspective, these changes mean that existing detection rules targeting Python scripts might fail to identify the malware. Although IP addresses and port numbers can be extracted from the Cython binaries through binary analysis, the runtime Python execution scripts could override these values with different C&C destinations passed as command-line arguments. Consequently, for some modules, the actual C&C destination cannot be determined from the binary alone without the accompanying execution script.

Infection chain: Cross-platform targeting



[open on a new tab](#)

Figure 1. Infection chain of the Void Dokkaebi intrusion set that abuses Cython to obfuscate its InvisibleFerret malware

While the initial attack vector has not been confirmed in this campaign, Void Dokkaebi has historically targeted software developers through fake job interviews. Our analysis of the infection chain reveals notable changes in both BeaverTail and InvisibleFerret.

BeaverTail now appears to function as a multistage component with capabilities similar to those of InvisibleFerret. This allows the threat not only to establish a JavaScript-only infection chain based on its development language, but also to download platform-specific versions of InvisibleFerret (e.g., mod.pyd, mod.so). This establishes the InvisibleFerret infection chain as well.

An analysis of BeaverTail

Initially, BeaverTail handled both data theft (e.g., browser credentials, cryptocurrency wallets) and InvisibleFerret payload delivery. The new variant of BeaverTail now carries functionality similar to InvisibleFerret. In addition, BeaverTail's obfuscation techniques have become more complex compared with earlier versions, using several layers of string protection and decoding logic:

- **Array shuffling and index lookup:** At startup, a large array containing approximately 300 Base64 fragments is shuffled using an immediately invoked function expression (IIFE). A lookup function is then used to retrieve each fragment by its hexadecimal index.
- **Base64 encoding with character stripping:** The first character of each encoded string is a randomly inserted junk byte, which is used to evade simple Base64 detection.
- **XOR encryption:** The most sensitive strings, such as file paths and execution commands, are XOR-encrypted using a 4-byte key.
- **Split-and-swap IP address encoding:** C&C IP addresses are split into two halves, which are swapped before Base64 decoding.

BeaverTail variants

BeaverTail has evolved into a set of multiple variants that use the same obfuscation techniques. In addition to the original information-stealing and downloader module, it now includes a backdoor, browser-stealing, and trojanized cryptocurrency wallet installation modules.

These variants appeared by the end of October 2025. They are identified based on the names used when they are downloaded. Table 1 lists the main features of each BeaverTail variant.

Name	Main Features
Information-stealing and downloader module: BeaverTail (gjs)	• Steals information stored in web browsers and cryptocurrency wallet data • Downloads and executes BeaverTail (njs), BeaverTail (zjs), and InvisibleFerret
Backdoor module: BeaverTail (njs)	• Possesses backdoor capabilities with backdoor execution functions starting with "ssh_ ", and downloads and executes BeaverTail (cjs) as the next stage payload based on commands

Name	Main Features
	- Collects system and network information, and accesses <code>hxxp://ip-api[.]com/json</code> to gather geolocation data based on the IP address - Backdoor execution functions starting with <code>"ssh_"</code> and the collection of OS and network information identical to those implemented in InvisibleFerret (pay)
Browser-stealing module: BeaverTail (zjs)	- Steals cryptocurrency wallet data, seed phrases, private keys, and passwords - Steals other information related to developer settings
Trojanized cryptocurrency wallet installation module: BeaverTail (cjs)	- Downloads and installs trojanized browser extensions in Chrome and Brave Browser, including wallet-related extensions, such as MetaMask, Coinbase Wallet, and Phantom - Downgrades the Chrome version on macOS; this functionality is equivalent to InvisibleFerret (mc)

Table 1. Key features of the BeaverTail modules

The Cython-obfuscated InvisibleFerret is downloaded by BeaverTail (gjs). In the code, the `_rum()` function is used to initiate the InvisibleFerret infection chain for both non-Windows and Windows environments, as shown in Figure 8.

```
const _rum = async () => await new Promise((a, b) => {
  if (pl[0] != 'w')
    dnp_m();
  else {
    const c = '' + hd + '\.py2\py.exe';
    existSinc('' + c) ? dnp() : inp0();
  }
});
```

[open on a new tab](#)

Figure 8. The function call to start InvisibleFerret's infection chain

The `dnp_m()` and `dnp()` functions are responsible for downloading and executing InvisibleFerret. The Windows version of the `dnp()` function is shown in Figure 9.

In previously observed BeaverTail (gjs) samples, InvisibleFerret (main) was downloaded from a URL path in the format `/client/{sType}` where `sType` is an identifier used by the threat actor. It is saved with the filename, `main_{sType}.py`. In this campaign, InvisibleFerret is downloaded from `/clw/{sType}`.

A .mod file is then created and executed via Python. On non-Windows systems, the payload is instead downloaded from /clw1/{sType}, after which a similar .mod file is created and executed. However, the binary is in Mach-O format and operates only in macOS environments.

Because the Cython-generated binaries are not self-contained executables, they depend on the CPython runtime as extension modules. As a result, they must be launched through a Python script or interpreter. The .mod file serves this purpose, and BeaverTail creates and executes it accordingly.

The execution flow is shown in Figure 4. BeaverTail (gjs) is a JavaScript-based module, so it can't load Cython extension modules directly. Instead, it does the following:

1. Downloads the Cython binary (mod.pyd or mod.so) from the C&C server
2. Writes a Python script (.mod) to disk
3. Invokes the Python interpreter to execute it

The .mod script then imports mod.pyd or mod.so as a Python extension module, passing sType, encoded IP addresses, and a port number as command-line arguments. The Cython module deobfuscates the embedded Python payload and executes it via exec, as shown in Figure 10.

An analysis of the Cython-obfuscated InvisibleFerret

Cython compilation obfuscates the original Python source by converting it into native binaries. With this technique, defenders should not rely on simple text-based searches for Python scripts.

However, numerous forensic artifacts remain in the resulting binaries. The InvisibleFerret binaries obfuscated with Cython retain programming-related artifacts. In Windows environments, the initialization function name in the export table, such as PyInit_, can reveal the user-defined module name.

Additionally, searching the binary could uncover embedded string references to source files, such as .pyx and .c, which can provide clues about the original development environment and build-time configuration. For example, in mod.pyd, the export table still contains PyInit_mod. Using mod as a keyword to search the binary strings reveals references such as mod.pyc, mod.c, and mod.lambda as shown in Figure 11.

In macOS environments, the following file paths were retained as strings within the binaries:

- /Users/administrator/Pictures/Work/py_module_work/build/temp.macosx-10.13-universal2-cpython-312/mod.o
- /Users/administrator/Pictures/Work/py_module_work/build/temp.macosx-10.13-universal2-cpython-312/pad.o
- /Users/administrator/Pictures/Work/py_module_work/build/temp.macosx-10.13-universal2-cpython-312/brw.o/Users/administrator/Pictures/Work/py_module_work/build/temp.macosx-10.13-universal2-cpython-312/mc.o

Files with the .o extension are known as an object file, a machine code compiled for the CPU. The file paths indicate the environment used for compilation. In the macOS build environment, the user name administrator was

used, with Pictures serving as the working directory. The project name is presumed to have been `py_module_work`.

In the Cython-obfuscated InvisibleFerret, Python string constants within modules were found to be Zlib-compressed and stored in binary sections, as shown in Figure 12 right panel). This suggests that `CYTHON_COMPRESS_STRINGS` was enabled during Cython compilation.

As shown in Figure 12, by examining the arguments of the `PyMemoryView_FromMemory` function, one can obtain the offset and size of the Zlib-compressed string constants. Decompressing with Zlib yields a string table, as shown in Figure 13.

The format is as follows:

[XOR Encoded IP address] ? [Module Name].pyx [identifiers] [Base64 blob]

In practice, applying the XOR encoding described below to `91d840f599206f13` yields the IP address `45[.]59[.]160[.]199`. The port number is defined within the binary file, as shown in Figure 14.

Subsequently, the embedded Python payload is invoked through `PyRun_StringFlags()`, as shown in Figure 15

From the string table, the following identifiers can be confirmed:

- `zlib`
- `base64`
- `decompress`
- `b64decode`
- `__import__`,

The `PySlice_New` function reveals that `::1` is used. Based on this information, the obfuscation technique is identical to the one used by previous versions of InvisibleFerret, as shown in Figure 16. A lambda function is used to reverse a Base64-encoded string, then Base64-decode the result, and decompress it with Zlib. This process is repeated iteratively. The code that performs this deobfuscation and expansion was converted to C/C++ source code by Cython and then compiled.

This finding is significant, especially for defenders. Despite the Cython-based obfuscation, the core deobfuscation logic remains unchanged from prior versions. Analysts familiar with previous InvisibleFerret techniques can apply the same methods to recover the embedded Python payload from the compiled binaries.

Therefore, the code after deobfuscation remains a Python script. Although IP addresses and port numbers are defined within `mod.pyd` and `mod.so`, executing a Cython-generated binary requires a Python script. For this purpose, the execution Python scripts, such as `.mod` shown in Figure 11, are used. These scripts are designed to pass specific command-line arguments to `mod.pyd` or `mod.so`, including `sType`, two encoded IP addresses, and a port number.

The deobfuscated InvisibleFerret introduces a new `start` function and a `dcp` function. First, it executes the `start` function shown in Figure 17.

The start function passes argument values to the dcp function shown in Figure 18 to obtain the connection destination. It also creates Python scripts pad0 and brw0 for executing the next-stage payloads.

The dcp function reconstructs the IP address from a 16-character, XOR-encoded string received as an argument using an XOR-based routine (Figure 19). In this manner, the connection destination could be obtained from arguments passed at runtime via the Python execution script, or it could use values hardcoded within the binary file. Furthermore, some variants delete the execution Python script after use. Table 2 summarizes which modules receive arguments at runtime and which possess the script deletion capability

InvisibleFerret Name	Connection Destination from Execution Python Script	Deletion of Execution Python Script
mod.pyd, mod.so	Obtains both IP address and port number	None
pad.pyd, pad.so	Obtains both IP address and port number	None
brw.pyd, brw.so	Obtains both IP address and port number	None
mc.so	None (hardcoded in binary)	Yes

Table 2. Summary of the capabilities of InvisibleFerret's modules (e.g., execution Python script, delete script)

Defenders can use the code shown in Figure 19 to determine the connection destination from the InvisibleFerret execution Python scripts, with the exception of mc.so. At present, only mc.so deletes its execution Python script. Since deleting the execution Python scripts of other modules would also conceal connection destinations, we expect that the remaining modules will be updated to delete their execution Python scripts as well.

The key capabilities of the Cython-obfuscated InvisibleFerret variants discovered in this investigation are shown in Table 3.

Name	Key Features
mod.pyd, mod.so	- Downloads pad.pyd and brw.pyd for Windows, and pad.so for macOS - Features consistent with the main InvisibleFerret
pad.pyd, pad.so	- Possesses backdoor capabilities with backdoor execution functions starting with "ssh_" - Downloads and executes InvisibleFerret (any) as any_{sType}.py and brw.pyd or brw.so as the next-stage payload based on commands - Collects system and network information, and accesses hxxp://ip-api[.]com/json to gather geolocation data based on the IP address, cryptocurrency wallets, service keys and passwords, clipboard monitoring, and keylogging (Windows only) - Features consistent with InvisibleFerret (pay variant)

Name	Key Features
brw.pyd, brw.so	- Steals authentication information and credit card details stored in browsers - Features consistent with InvisibleFerret (bro variant)
mc.so	- Downloads and installs trojanized browser extensions in Chrome and Brave Browser, including wallet-related extensions such as MetaMask, Coinbase Wallet, and Phantom - Has the ability to downgrade the version of Chrome in the system - Features are consistent with InvisibleFerret (mc variant)

Table 3. The key features of the Cython-obfuscated InvisibleFerret

The original InvisibleFerret (mc) targeted only MetaMask, but the current mc.so now also targets Coinbase and Phantom, as shown in Figure 20. We expect that the number of targeted cryptocurrency wallets will increase further in the future.

To make the trojanized cryptocurrency wallet component work as a Chrome extension, the attackers downgrade the version of Chrome on macOS. This is done to bypass Google's enforced transition to Manifest V3 for Chrome extensions. The new extension specification does not provide the functionality required for relaying or tampering with cryptocurrency wallets. As a result, they downgrade Chrome to a version that still supports Manifest V2. Brave Browser is also targeted, as it still offers only limited support for Manifest V2.

In this infection chain, any variants of InvisibleFerret used to download and configure the AnyDesk execution environment appear to be in the midst of a transition to Cython. The downloaded code remains a Python script, and as shown in Figure 21, it contains sType and a 116-character, XOR-encoded string preceding the obfuscation code.

However, the deobfuscated InvisibleFerret (any) code still contains the legacy code used in previous versions, which decodes a 20-character, Base64-encoded string in 10-character segments to recover the IP address. It does not include a function to decode the new 16-character XOR-encoded string (Figure 22). Furthermore, because the 20-character Base64-encoded string is commented out, execution fails.

```

1  # hst='cuMjA1===NjcuMjAzLj'; hst2='YzLjU1===NDUuNTkuMT'
2  # pt0=1244; pt1=1244
3
4  import socket, os, platform, time, subprocess, requests, sys, getpass
5  from base64 import b64decode
6  os_type = platform.system()
7
8  appdata = os.getenv('LOCALAPPDATA')
9  hn = socket.gethostname()
10 if os_type=="Darwin":
11     try:gu = getpass.getuser();hn = f'{hn}+{gu}'
12     except:pass
13
14 host1 = b64decode(hst[10:] + hst[:10]).decode();host2 = f'http://{host1}:{pt0}'
15 host4 = b64decode(hst2[10:] + hst2[:10]).decode();host5 = f'http://{host4}:{pt1}'
16

```

[open on a new tab](#)

Figure 22. The conventional decoding routine revealed after deobfuscation

The threat actor adopted Cython-based obfuscation, thereby converting the malware from Python scripts to binary files. Based on the capabilities of the Cython-based obfuscated InvisibleFerret, we assess that it is still under development, as its intended purpose remains unchanged and does not appear to be fully functional. These changes mean that existing detection rules targeting Python scripts may fail to identify the malware, and even if the binary is discovered, the C&C destination may remain unidentifiable without the accompanying execution script.

MITRE ATT&CK Mapping

Tactics ID	Technique Name
T1059	Command and Scripting Interpreter
T1027	Obfuscated Files or Information
T1562.001	Disable or Modify Tools
T1070.004	Indicator Removal: File Deletion
T1547.001	Boot or Logon AutoStart Execution: Registry Run Key / Startup Folder
T1053.005	Scheduled Task/Job: Scheduled Task
T1082	System Information Discovery
T1083	File and Directory Discovery
T1518	Software Discovery
T1057	Process Discovery
T1555.003	Credentials from Password Stores: Credentials from Web Browser

Tactics ID	Technique Name
T1056.001	Input Capture: Keylogging
T1115	Clipboard Data
T1071.001	Application Layer Protocol: Web Protocols
T1071.002	Application Layer Protocol: File Transfer Protocols
T1219.002	Remote Access Software: Remote Desktop Software
T1102.001	Web Service: Dead Drop Resolver
T1571	Non-Standard Port
T1048	Exfiltration Over Alternative Protocol
T1041	Exfiltration Over C&C Channel

Table 4. A summary of the tactics, techniques, and procedures (TTPs) used in the campaign

Based on technical artifacts and TTPs as well as code and infrastructure overlaps with BeaverTail and InvisibleFerret, TrendAI™ Research attributes this campaign to Void Dokkaebi with high confidence.

Conclusion

Void Dokkaebi's adoption of Cython-compiled malware represents an evolution in the group's capabilities. The Cython-based obfuscation converts readable Python scripts into native binaries, and thus bypasses previous Python script-based detections. InvisibleFerret is now distributed as .pyd files on Windows and .so files on macOS.

While the original source code is no longer directly readable, our analysis shows that the underlying obfuscation techniques remain unchanged from previous versions. Programming artifacts, build environment paths, and string tables are still recoverable from the binaries, which can enable defenders to identify variants and extract C&C infrastructure through binary analysis. The mc module's wallet trojanization capabilities (particularly the Chrome downgrade attack on macOS) also show the adversary's attempts to bypass modern browser security controls.

Despite these advancements, the campaign exhibits telltale signs of ongoing development. Incomplete variable definitions and missing functionality in the any.py component suggest that threat actors face challenges in fully finishing their Cython migration.

A BeaverTail variant with a functionality equivalent to InvisibleFerret also exists within the infection chain. Even so, it continues to download and execute InvisibleFerret. Although the two malware families are developed in different programming languages, their functionality overlaps significantly. This raises questions about why attackers maintain both BeaverTail and InvisibleFerret within the infection chain. While this remains speculative, the use of a shared C&C server suggests the presence of a malware developer cluster organized around specific programming languages.

Given the incomplete Cython migration and active development patterns observed, Void Dokkaebi will likely continue refining both BeaverTail and InvisibleFerret. Defenders should also anticipate an expanded set of trojanized cryptocurrency wallet extensions targeting additional platforms.

TrendAI™ Research continues to monitor Void Dokkaebi and related campaigns, delivering actionable intelligence that keeps your organization ahead of evolving threats. Our comprehensive threat intelligence, combined with advanced detection capabilities, ensures organizations remain protected against sophisticated attacks targeting cryptocurrency assets and sensitive enterprise data.

TrendAI Vision One™ Threat Intelligence Hub

[TrendAI Vision One™ Threat Intelligence Hubproducts](#) provides the latest insights on emerging threats and threat actors, exclusive strategic reports from TrendAI™ Research, and TrendAI Vision One™ Threat Intelligence Feed in the TrendAI Vision One™ platform.

Emerging Threats: [Void Dokkaebi Adopts Cython-Compiled InvisibleFerret](#)

Threat Actor: [https://portal.xdr.trendmicro.com/index.htmlVoid Dokkaebi](https://portal.xdr.trendmicro.com/index.htmlVoid%20Dokkaebi)

[Void Dokkaebi Adopts Cython-Compiled InvisibleFerret](#)

Hunting Queries

TrendAI Vision One™ customers can use the XDR Data Explorer App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

```
eventSubId:(101 or 109) AND objectFilePath:( "\.vscode\mod.pyd" OR "/.vscode/mod.so" OR "\.vscode\pad.pyd" OR "\.vscode\brw.pyd" OR "/.vscode/pad.so" OR "/.vscode/brw.so" OR "/.vscode/mc.so" OR "\.vscode\.mod" OR "\.vscode\pad0" OR "\.vscode\brw0" OR "/.vscode/.mod" OR "/.vscode/pad0" OR "/.vscode/brw0" OR "/.vscode/mc0")
```

```
eventSubId:2 AND processCmd:( "\.vscode\mod.pyd" OR "/.vscode/mod.so" OR "\.vscode\pad.pyd" OR "\.vscode\brw.pyd" OR "/.vscode/pad.so" OR "/.vscode/brw.so" OR "/.vscode/mc.so" OR "\.vscode\.mod" OR "\.vscode\pad0" OR "\.vscode\brw0" OR "/.vscode/.mod" OR "/.vscode/pad0" OR "/.vscode/brw0" OR "/.vscode/mc0")
```

More hunting queries are available for TrendAI Vision One™ with [Threat Intelligence Hubproducts](#) entitlement enabled.

Indicators of Compromise

The indicators of compromise for this entry can be found [here](#).

Tags