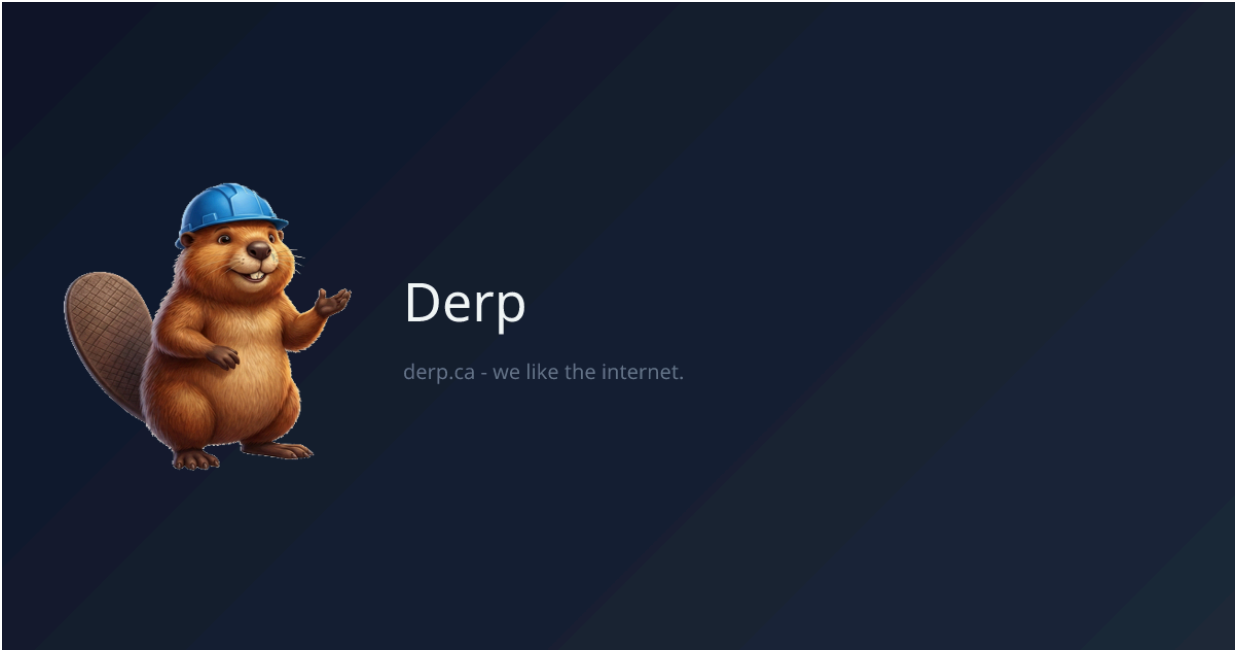


VioletWorm v4.7 (Violet RAT): The Most Dangerous Payload in a 9-RAT Toolkit



malware rat .net incident-response violet-rat violetworm xworm

We recovered a .NET assembly from the same multi-stage intrusion that produced [PureLogs](#) and [PureCrypter](#). Across six parallel attack chains, the actor deployed nine RAT families. This one is the largest payload, has the widest capability set, and sits on separate infrastructure. The actor's internal label for it was Viooooooooo.

Tria.ge classifies this family as **violetworm**. We use **VioletWorm** as the canonical name throughout this post and use **Violet RAT** only where campaign artifacts use that naming.

Sample overview

Field	Value
Family	violetworm (aka Violet RAT)
Version	v4.7
Original filename	VioletClient.exe

Field	Value
Internal label	Viooooooooo / vio
SHA256 (Nov10)	d656bcfe9a98007fcb3e2be4430a9b24d258c046b0c768ac43699436aceb98e6
Size (Nov10)	1,432,360 bytes
SHA256 (Nov19)	e2c5921e5c354000c38653d05ac3255865d20bc986da34e68246da6f72ec1fed
Size (Nov19)	1,425,920 bytes
SHA256 (Nov24)	cbd4cd4c42b3ee0fb99a254a97ff2c2aacc47d83e08601ae8cdf3b333f07806
Size (Nov24)	1,029,120 bytes
Type	.NET assembly (PE32, Mono)
Framework	.NET Framework 4.0
Campaign overlap	SERPENTINE#CLOUD (Securonix) -- shared tooling and TTPs, but Securonix reporting does not mention VioletWorm
Assessment	Most dangerous payload in the toolkit

The assembly is the inner payload of a multi-stage loader chain. Python droppers handle delivery. AES-256-CBC or RC4 protects the outer layer. Donut v0.9.2 shellcode handles CLR bootstrap. The final stage runs in memory through reflective assembly loading.

Three builds were recovered. The Nov10 build is the primary analysis target. The Nov19 build is a recompile with the same configuration and command set. The Nov24 build is distinct: different C2 endpoint (vigroup2125.duckdns.org:2125), different XOR keys, and a different mutex. It was recovered from an unobfuscated developer artifact.

Classification and Tria.ge corroboration (2026-02-25)

Tria.ge gives independent classification and runtime support for the three payload assemblies and three loader scripts recovered in this intrusion.

Sample	Type	Tria.ge ID	Score	Family	Status
VioletWorm Nov10 payload	payload	260225-st7zrsbz6d	10	violetworm	reported
VioletWorm Nov19 payload	payload	260225-st8ajabz6e	10	violetworm	reported
VioletWorm Nov24 payload	payload	260225-st8w3abz6f	10	violetworm	reported
VioletWorm Nov10 loader	loader	260225-syqahsb15d	3	-	reported
VioletWorm Nov19 loader	loader	260225-syq7tab15f	3	-	reported
VioletWorm Nov24 loader	loader	260225-syrtcab15g	3	-	reported

All three payload submissions were classified as **violetworm** with score 10 across static and behavioral tasks. Loader submissions still showed behavior but did not receive family attribution. A Tria.ge search for **family:violetworm** currently shows samples dating back to 2025-11-28T02:07:01Z (251128-cjy14sylvcr).

Tria.ge task telemetry corroborated both C2 endpoint sets already recovered from reversing:

- 45.58.143.254:7575 for Nov10/Nov19 payloads (vijsklet.duckdns.org)
- Tria.ge runtime endpoint 213.227.152.82:2125 for the Nov24 payload (vigroup2125.duckdns.org); breach-report infrastructure attribution for this domain remains TBD

Tria.ge sample links:

- Nov10 payload: <https://tria.ge/260225-st7zrsbz6d>
- Nov19 payload: <https://tria.ge/260225-st8ajabz6e>
- Nov24 payload: <https://tria.ge/260225-st8w3abz6f>
- Nov10 loader: <https://tria.ge/260225-syqahsb15d>
- Nov19 loader: <https://tria.ge/260225-syq7tab15f>
- Nov24 loader: <https://tria.ge/260225-syrtcab15g>

Architecture: massive command dispatcher with C2-delivered plugins

VioletWorm (Violet RAT) is a command dispatcher. The main binary contains an 8,000-line IL handler with 120 decoded command branches. Capabilities such as ransomware, HVNC, keylogging, file management, and credential theft are implemented in plugin DLLs delivered by C2 at runtime. The binary loads those plugins with `Assembly.Load()`, instantiates `Class1`, and calls methods through VB.NET late binding (`NewLateBinding.LateCall`).

This makes it structurally closer to [PureLogs](#) than it first appears. Both receive capabilities from C2. The difference is scale. PureLogs has a handful of commands and a small dispatcher. VioletWorm covers remote shell, file manager, keylogger, screenshot streaming, active window monitoring, screen control, HVNC, remote desktop, webcam/mic/audio capture, DDoS, clipboard hijacking, crypto clipping, password stealing, network recon, TCP connection monitoring, Discord grabbing, ransomware, process injection, USB spreading, ngrok tunneling, live chat with the victim, AV enumeration, UAC elevation, and Windows Defender tampering.

At 1.4 MB, it is the largest payload in the toolkit -- 10x larger than AsyncRAT (52 KB) and 3x larger than VenomRAT (70 KB). The bulk is the dispatcher itself and 11 large base64-encoded data blobs stored in the .NET User Strings heap. These blobs are initialized in the config class constructor but never referenced by the main binary's code -- they are likely accessed via reflection by C2-delivered plugins at runtime.

String encoding scheme

All strings in the .NET User Strings (#US) heap are base64 encoded. VioletWorm uses two encoding tiers:

2-layer encoding (plaintext strings): `plaintext -> base64 -> base64`. These cover configuration fields, command names, registry paths, and file paths. Decoding two layers of base64 yields the plaintext directly.

3-layer encoding (XOR-obfuscated strings): `plaintext -> XOR(key) -> base64 -> base64 -> base64`. These cover C2 infrastructure, sensitive command identifiers, and User-Agent strings. Three layers of base64 yield XOR-encrypted bytes that require the key to decode.

XOR key derivation

The key is self-contained in the binary. String index [24] in the #US heap stores the key through the same triple-base64 encoding:

```
Stored:      ZVc1T1JVNUpUQT09
1x base64:   eW50RU5JTA==
2x base64:   ynNENIL
```

Key: `ynNENIL` (7 bytes: `0x79 0x6e 0x4e 0x45 0x4e 0x49 0x4c`)

XOR decryption is byte-wise with key recycling: `plaintext[i] = ciphertext[i] ^ key[i % 7]`.

Verification: the known plaintext `uninstall` XOR'd with the key matches the bytes stored (after triple-base64) at string index [64]. Once the key is recovered, every XOR-encoded string in the binary decodes immediately.

C2 protocol

Transport

HTTP POST over TCP. No TLS. The C2 address and port are XOR-encoded in the binary.

Parameter	Value
C2 domain	<code>vijdklet.duckdns.org</code>
C2 port	<code>7575</code>
Protocol	HTTP POST
Content-Type	<code>application/x-www-form-urlencoded</code>
Field separator	<code>XSXSXSX</code>
Keepalive	<code>PING?</code> / <code>PONG</code>

Field separator

All C2 messages use `XSXSXSX` as the initial field separator in the POST body. Command arguments, file paths, and data chunks are concatenated with this delimiter. The separator is initialized from a 2-layer base64 string in the config, but it is mutable -- the C2 server rotates it mid-session. Since the AES encryption key is derived from `MD5(UTF8.GetBytes(separator))`, rotating the separator also rotates the encryption key.

Keepalive

The RAT sends `PING?` at regular intervals. The server responds `PONG`. If the keepalive fails, the client reconnects.

User-Agent rotation

Six hardcoded User-Agent strings rotate across requests:

#	Platform
1	iPhone / Safari / iOS
2	Windows / Chrome / Edge
3	Mac / Safari / macOS
4	Linux / Firefox
5	iPad / Safari / iPadOS

#	Platform
6	Windows / alternate

All six are XOR-encoded (3-layer base64). The rotation makes traffic pattern matching harder -- each request appears to come from a different browser and operating system.

Microsoft URL whitelist

VioletWorm checks outbound URLs against a whitelist of Microsoft domains. If the destination matches any of the following, the request is excluded from interception:

- windowsupdate
- winatp-gw-cus
- watson
- msedge
- go.microsoft.com
- activation.sls

These are telemetry and update endpoints. Whitelisting them prevents the RAT from intercepting legitimate Windows traffic.

Configuration

Extracted fields from the decoded binary:

Field	Value	Source
C2 domain	vjdkklet.duckdns.org	XOR-encoded, index [18]
C2 port	7575	XOR-encoded, index [19]
Version	Violet v4.7	Base64, index [37]
Tag	<Violet>	Base64, index [21]
XOR key	ynNENIL	Base64, index [24]
Mutex (single-instance)	aXTyo1HpFXkKUYoL	Base64, index [25]
Mutex GUID #1	3d847c5c-4f5a-4918-9e07-a96cea49048d	Base64, index [43]

Field	Value	Source
Mutex GUID #2	89c43fcf-5e52-4be7-a719-a26139ce636a	Base64, index [50]
USB spreader	USB.exe	Base64, index [22]
Separator (initial)	XSXSXSX	Base64, index [20]; rotated by C2 at runtime
PasteUrl	%PasteUr1% (placeholder, unused)	Base64, index [0]
Install filename	3d847c5c-4f5a-4918-9e07-a96cea49048d.exe	Derived from mutex GUID
Cleanup script	WinTempClean32.bat	Base64, index [41]
Tunneling	\ngrok.exe	Path reference

Persistence targets:

- HKEY_CURRENT_USER\SOFTWARE\ -- Run key persistence
- HKEY_LOCAL_MACHINE\software\classes\ -- COM/class registration abuse

Three mutex values serve different purposes. aXTyo1HpFXkKUYoL is the primary single-instance lock -- despite looking like an AES-128 key, IL tracing confirms it's only used in `new Mutex(false, "aXTyo1HpFXkKUYoL", out bool)`. The first GUID (3d847c5c...) doubles as the install filename. The second GUID (89c43fcf...) is a secondary instance check.

Capabilities

The dispatcher has 120 command branches, each decoded from double-base64 + XOR-encoded constants in the IL. Below are the ones with enough context to describe. The full command map is in the reference documentation.

Remote shell

Command	Function
shellfuc	Open interactive shell (CMD.EXE)
runnnnnn	Execute command in shell
closeshell	Terminate shell session

File manager

Command	Function
<code>showfolderfile</code>	List directory contents
<code>creatnewfolder</code>	Create directory
<code>creatfile</code>	Create file
<code>downloadfile</code>	Exfiltrate file to C2
<code>Execute</code>	Run file
<code>Rename</code>	Rename file
<code>viewimage</code>	View image file
<code>Delete / DelP</code>	Delete file or path
<code>7zIT / 7zzip</code>	Compress files with 7zip

The file manager includes a bundled 7zip integration (`7zip\7z.exe`) for compressing files before exfiltration.

Keylogger

Command	Function
<code>KL</code>	Start keylogger
<code>KLget / KLGET</code>	Retrieve captured keystrokes
<code>closeKL</code>	Stop keylogger

Keylog data is stored under `HKEY_CURRENT_USER\SOFTWARE\` -- the same registry path used for persistence.

Screen control and HVNC

Command	Function
HVNC / HvNcX	Hidden VNC session (invisible remote desktop)
hvincxdis	Disconnect HVNC
shwup	Display fake "Windows Update" screen overlay
hidup	Remove the fake overlay
BSOD	Trigger Blue Screen of Death

While the victim sees "Installing updates, please wait..." the operator has full control of the hidden desktop.

Webcam, microphone, and audio

Command	Function
WBCM	Webcam capture
MICL	Microphone listener
wsound	System audio capture

These three are the only cached plugins. On first use, the C2 sends the plugin DLL and the RAT stores the raw bytes in a static field. Subsequent calls reload from cache instead of re-requesting from the C2.

Screenshot capture

Command	Function
RSS	Start remote screenshot stream
RSSDis	Stop screenshot stream

Screenshots are captured as JPEG ([image/jpeg](#)) with configurable region bounds (X, Y, Width, Height). The stream commands suggest live screen monitoring, not just one-off captures.

Active window monitoring

Command	Function
<code>ACT</code>	Start active window tracking
<code>ACTG</code>	Get active window title
<code>killAct</code>	Stop active window tracking

Passive surveillance -- tracks which application the victim is using without capturing keystrokes. The operator can watch for banking sites, email clients, or crypto wallets opening and then trigger other modules (keylogger, screenshot, HVNC) at the right moment.

DDoS

Command	Function
<code>DDoSS</code>	Start DDoS attack
<code>DDoST</code>	Stop DDoS attack

An unusual capability for a RAT deployed in a targeted intrusion. The infected machines can be enrolled as DDoS nodes on demand.

Clipboard and crypto clipper

Command	Function
<code>Cilpper</code>	Enable clipboard monitor / crypto clipper
<code>clss</code>	Clear clipboard

The crypto clipper monitors the clipboard for cryptocurrency wallet addresses and silently replaces them with attacker-controlled addresses. The dispatch command is misspelled -- `Cilpper` instead of `Clipper`. A correctly-spelled copy exists at string index [104], but the dispatcher uses the XOR-encoded misspelling at index [103].

Network reconnaissance and sniffer

Command	Function
NetDisCV	Network discovery scan
SnifStrt	Start network sniffer
SniffKll	Stop sniffer
TCPV	View active TCP connections
TCPG	Get TCP connection details

Ransomware

Command	Function	Plugin method	Args
ENC	Encrypt target file	ENC(hwid, path)	2
DEC	Decrypt target file	DEC(hwid, path)	2
RENC	Recursive directory encryption	ENC(hwid, ...params)	6 (5 ByRef)
RDEC	Recursive directory decryption	DEC(hwid)	1

All four commands are confirmed in the IL command dispatcher. Like every other capability, the actual encryption logic lives in a C2-delivered plugin DLL.

The first argument to every call is a machine fingerprint: MD5(ProcessorCount + UserName + MachineName + OSVersion + DriveSize), output as a 32-character lowercase hex string. This HWID is the primary cryptographic input to the plugin -- RDEC receives it as its **only** argument, meaning the plugin needs nothing else from the RAT to reverse the encryption. The actual encryption algorithm and key derivation are inside the C2-delivered plugin DLL, which we do not have.

ENC and DEC are single-file operations: the C2 sends a plugin DLL and a target path. RENC and RDEC operate on directory trees and are governed by a state machine. A guard flag prevents concurrent execution:

1. RENC arrives -- flag set to 1 (encrypting), plugin loaded, ENC() called with 6 arguments (5 ByRef, allowing the plugin to return status data)
2. Plugin returns -- flag set to 2 (encryption complete)
3. RDEC arrives -- only executes if flag is exactly 2, then calls DEC(hwid), resets flag to 0

This means the operator must complete encryption before decryption is possible. The state machine enforces the ransom workflow: encrypt first, decrypt only after.

Persistence and AV evasion

Command	Function
<code>install</code>	Install persistence (via plugin)
<code>uninstall</code>	Remove persistence (via plugin)
<code>update</code>	Update RAT binary
<code>WDKillerNew</code>	Disable Windows Defender
<code>WDPL</code>	Windows Defender plugin (separate from WDKillerNew)
<code>AntiiReset</code>	Survive system reset
<code>PSleep</code>	Prevent system sleep
<code>askuac</code>	Prompt for UAC elevation
<code>bot</code>	Bot control
<code>admin</code>	Admin privilege check
<code>kill</code>	Kill process
<code>JMar / JMarKLL / SysKLL</code>	Process killing variants

Before persisting, the RAT queries `Select * from AntivirusProduct` via WMI (namespace `\root\SecurityCenter2`) to enumerate installed security products.

It also sets `ShowSuperHidden` to 0 under `Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced`. This hides protected OS files in Explorer -- the RAT's dropped files become invisible even with "Show hidden files" enabled.

`WDKillerNew` kills Windows Defender. `AntiiReset` survives system reset attempts.

USB spreading

Command	Function
USB.exe	USB worm payload name
WinBIT32-*	USB worm variant name pattern

The USB spreader creates `.lnk` shortcut files on removable drives using `WScript.Shell` COM automation. Each shortcut has its `TargetPath` set to `cmd.exe` with arguments like `/c start [payload] & start explorer [folder] & exit --` the payload runs, then the original folder opens so the victim sees what they expected. Icon paths are read from `HKEY_LOCAL_MACHINE\software\classes\folder\defaulticon\` so the shortcuts look like normal folders.

XWorm v3.1 from the same campaign uses an identical `USB.exe` spreading mechanism.

Ngrok tunneling

Command	Function
ngrok	Ngrok tunnel connection
InstallN	Install ngrok binary

Ngrok provides reverse proxy tunneling for C2 redundancy. If the primary DuckDNS domain is blocked, the operator can route C2 traffic through ngrok.

Live chat

Command	Function
LLCHAT	Start live chat session
Xchat	Extended chat
SNote	Send note to victim

Real-time text chat between the operator and the victim. The operator can impersonate IT support, demand ransom, or tell the victim to take specific actions mid-intrusion.

Other commands

Command	Function
<code>hrdp / hrdp+ / RD- / RD+</code>	Remote desktop (connect, extended, disconnect, reconnect)
<code>PassR / PC#</code>	Password recovery
<code>Email</code>	Email credential theft
<code>GrabberDC</code>	Discord token theft
<code>CookieST / coki</code>	Browser cookie stealing
<code>injRun</code>	Inject and run payload in target process
<code>PE</code>	PE file execution
<code>Pvbnet / vbb / cbb</code>	VB.NET payload execution variants
<code>script</code>	Script execution
<code>regfuc</code>	Registry operations
<code>openhide</code>	Hidden browser instance via <code>internetexplorer.application</code> COM
<code>NETINS</code>	Network install
<code>FURL</code>	URL fetch
<code>UPtoS</code>	Upload to server
<code>MapsPLU</code>	Maps plugin
<code>JustFun</code>	Unknown (likely nuisance function)
<code>### / \$\$\$ / ^^& / </code>	Signal/control codes

Plugin loading architecture

Every plugin DLL arrives from the C2 server. The loading path is:

1. AES decrypt the incoming TCP stream (AES-128-ECB, key = MD5 of the separator)
2. UTF-8 decode the plaintext bytes
3. Split by the separator (XSXSXSX) -- A[0] = command name, A[1] = plugin DLL bytes (base64)
4. Base64 decode A[1] to raw bytes
5. Assembly.Load() the decoded bytes, get modules, get types, CreateInstance("Class1")
6. NewLateBinding.LateCall() the target method on Class1

IL disassembly confirms 46 plugin loader call sites in the command handler. 43 load the plugin directly from C2 command data (A[1]). Three -- webcam, microphone, and audio -- cache plugin bytes in static fields and reload from cache on subsequent calls. All three cached plugins call method CON on Class1.

Separator rotation

The separator is not static. Six stsfld write instructions in the command handler update it to values received from the C2 server, which means the AES key also changes mid-session.

Infrastructure

The Nov10/Nov19 builds use vjkdlet.duckdns.org:7575, which resolves to 45.58.143.254 (Sharktech, Amsterdam NL). This is the same IP hosting two PureLogs variants:

RAT	Domain	Port
VioletWorm v4.7 (Violet RAT, Nov10/Nov19)	vjkdlet.duckdns.org	7575
VioletWorm v4.7 (Violet RAT, Nov24)	vigroup2125.duckdns.org	2125
PureLogs (Mvfxsog.dll)	ydspwie.duckdns.org	9045
PureLogs (Qdjlj.dll)	nhvncpure*.duckdns.org	6757-6759, 8075-8076

The Nov24 build uses a second domain (vigroup2125) on a different port (2125). The naming convention is consistent: vi- prefix for VioletWorm domains, like vjkdlet.

Three RAT groupings ran on separate infrastructure. The Sharktech Amsterdam server was split from the ServerAstra Budapest cluster (91.219.238.167) that hosted four commodity RATs (DcRat, AsyncRAT, VenomRAT, Remcos). XWorm V3.1 ran on a third server (12.202.180.133, AT&T, Baltimore MD). The layout separates operations: VioletWorm and PureCoder tools shared one server, commodity RATs shared another, and XWorm remained isolated.

As of February 2026, port 7575 (VioletWorm) is still accepting connections on 45.58.143.254.

Build comparison

Property	Nov10 build	Nov19 build	Nov24 build
SHA256	d656bcfe...aceb98e6	e2c5921e...72ec1fed	cbd4cd4c...f07806
Assembly size	1,432,360 bytes	1,425,920 bytes	1,029,120 bytes
Shellcode size	1,464,594 bytes	1,722,111 bytes	1,067,778 bytes
Loader encryption	AES-256-CBC + 2x XOR	RC4 (per-file key)	AES-256-CBC + 2x XOR
Injection method	explorer.exe APC	In-process CFUNCTYPE	explorer.exe APC
C2 domain	vijdklet.duckdns.org	vijdklet.duckdns.org	vigroup2125.duckdns.org
C2 port	7575	7575	2125
XOR dispatch key	ynNENIL	ynNENIL	XSRXSX
Mutex	aXTyo1HpFXkKUYoL	aXTyo1HpFXkKUYoL	H3n0q1XPeIv1umQI

The Nov10 and Nov19 builds share the same C2 endpoint and XOR key -- the attacker recompiled between waves (assembly size changed by 6,440 bytes) but didn't change the configuration. The Nov24 build is a distinct compilation: different C2 endpoint ([vigroup2125](#) instead of [vijdklet](#)), different XOR keys, different mutex, and 400 KB smaller. It was recovered from an unobfuscated developer artifact ([1uunov24.py](#)), one of only two unobfuscated loaders in the entire campaign.

Deployment history

Wave	Loader filename	Encryption	Injection	C2
Wave 2 (Nov 10)	BKSN0LazyNov10_Viooooooooo.py	AES-256-CBC + 2x XOR	explorer.exe APC	vijdklet:7575

Wave	Loader filename	Encryption	Injection	C2
Wave 3 (Nov 10)	BKN0LazyNov10_Vioooooo.py	AES-256-CBC + 2x XOR	explorer.exe APC	vijdklet:7575
Wave 4 (Nov 19)	1vio-obf.py	RC4 key N0pzga4k	In-process CFUNCTYPE	vijdklet:7575
Nov 24	1uunov24.py (dev artifact)	AES-256-CBC + 2x XOR	explorer.exe APC	vigroup2125:2125
Wave 5 (Nov 19)	0BKSLazyNov20_vio.py	RC4 key NQzn2pMo	In-process CFUNCTYPE	vijdklet:7575
Wave 5 (Nov 19)	WBKSLazyNov20_vio.py	RC4 key Hs3TFDx1	In-process CFUNCTYPE	vijdklet:7575
Dec 2	1Dec2vio.py	Kramer .pyc RC4	In-process CFUNCTYPE	vijdklet:7575
Wave 7 (Dec 16)	via Shoopify.bat	Same payload	Same	vijdklet:7575

VioletWorm was deployed in every wave from Wave 2 onward. Eight separate loaders with different encryption keys deliver the RAT across two C2 endpoints. The Nov24 **1uunov24.py** is an unobfuscated developer artifact -- one of only two in the campaign -- and is the only loader that delivers the **vigroup2125** build.

Attribution

VioletWorm (also tracked as Violet RAT) is a separate MaaS product, not part of the PureCoder suite (PureLogs, PureCrypter, PureHVNC, PureMiner, PureRAT). The threat actor purchased tools from at least two MaaS vendors and deployed them together.

The C2 IP **45.58.143.254** is shared exclusively with PureLogs variants from the same campaign. No other RATs in the toolkit use this server. The attacker put VioletWorm and the PureCoder tools on one server (Sharktech Amsterdam) and the commodity RATs on another (ServerAstra Budapest). That's an operational choice by the buyer, not a developer-level connection between the tools.

VioletWorm is delivered through the same Python multi-stage loader infrastructure used by the other RATs in this campaign: same Donut shellcode version, same AES/XOR scheme, same injection methods, and the same Kramer obfuscator in later waves.

The attacker's internal label `Viooooooooo` follows the same pattern as `Ploggggggg` (PureLogs), `Asynccccc` (AsyncRAT), `Venommm` (VenomRAT), and `Annoriii` (DcRat) -- repeated last characters as padding. VioletWorm was always deployed alongside PureLogs. Both appear in the same loader sets, staging directories, and batch launchers.

VirusTotal classification

Neither sample had been submitted to VirusTotal before this analysis. On first submission, both scored 40-41/71. No vendor identifies it by name -- the dominant label is `Gen:Variant.MSILHeracles`, a generic Bitdefender heuristic for suspicious .NET binaries. ESET classifies both as `MSIL/XWorm.L`. IL comparison against an XWorm client confirms this is correct -- VioletWorm is a fork of XWorm with an expanded command set and different string encoding.

XWorm lineage

We compared VioletWorm's IL disassembly (27,623 lines) against an XWorm client sample (11,893 lines, SHA256 `ced525930c76834184b4e194077c8c4e7342b3323544365b714943519a0f92af`) across 10 structural dimensions. Five produced byte-identical IL sequences. Four were structurally similar. One differed.

Byte-identical IL:

- **AES implementation:** Both use `RijndaelManaged + MD5CryptoServiceProvider + ldc.i4.2` (CipherMode.ECB) with no IV. The encrypt, decrypt, and config-decryption methods match opcode-for-opcode.
- **HWID generation:** Both build a 5-element object array (`ProcessorCount, Username, MachineName, OSVersion, DriveInfo.TotalSize`), concatenate it, and pass it to an MD5 hash function. The IL from offset `0000` through `004e` is identical in both binaries. Both use `"Err HWID"` as the fallback string.
- **Mutex creation:** `new Mutex(false, name, out bool)`. The close/dispose sequence (null-check, `Close()`, set null) also matches byte-for-byte.
- **C2 socket setup:** `new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp)` with `ReceiveBufferSize` and `SendBufferSize` both set to `0xc800` (51,200 bytes), followed by `BeginReceive` for async I/O. Identical from the constructor through the buffer configuration.
- **Language:** Both are VB.NET, not C#. Both reference `Microsoft.VisualBasic` as their first external assembly, use `Operators.CompareString` for string comparisons, and include identical `My.MyApplication / My.MyComputer / My.MyProject` scaffolding from the VB compiler.

Structurally similar:

- **Plugin loading:** Both use `Assembly.Load(byte[])` for in-memory plugin execution. XWorm searches for a type named `"Plugin"` with a method named `"Run"`. VioletWorm searches by `FullName` suffix and uses `CreateInstance()`.
- **Command dispatch:** Both use a long if/else chain comparing `A[0]` against command constants via `CompareString`. XWorm stores commands as plaintext strings (~20 commands). VioletWorm XOR-encodes `A[0]` with key `ynNENIL` and compares against double-base64 constants (120 commands).
- **Class structure:** 1:1 mapping between core classes -- settings/config, entry point, connection handler, command dispatcher, utilities (crypto + HWID + mutex), persistence. Same functional decomposition, different names.

- **Config class:** Both use a single static class with a `.cctor` that initializes all fields. XWorm stores AES-encrypted base64 values (14 fields). VioletWorm stores 2-layer base64 values (30+ fields, with `Google` suffix appended to all names).

Different:

String encoding: XWorm encrypts config strings with AES-ECB using a key stored in the binary. VioletWorm replaced this with 2-layer base64 encoding (simpler, less secure). VioletWorm also added anti-decompiler marker classes (`ObfuscatedByGoliath`, `NineRays.Obfuscator`, `NetGuard`, `dotNetProtector`, `YanoAttribute`, `Xenocode`) that XWorm lacks.

The fork relationship is clear. The VioletWorm author took XWorm's codebase, replaced string encryption with base64 encoding, added XOR command encoding, expanded the command set from about 20 to 120, and added HTTP-based C2 address updates before socket connection. Core infrastructure remained the same: AES, HWID, mutex, sockets, and VB.NET late binding.

Vendor reporting

None of the existing campaign reports cover VioletWorm specifically. The closest are:

Vendor	Report	Coverage
Securonix	Analyzing SERPENTINE#CLOUD	Campaign with overlapping tooling and TTPs (does not mention VioletWorm)
Check Point Research	Under the Pure Curtain	PureCoder developer analysis (does not mention VioletWorm)
Fortinet	PureHVNC Deployed via Python Multi-stage Loader	Loader chain, PureHVNC component
eSentire	Quartet of Trouble	PureLogs delivery via TryCloudflare tunnels

Indicators of compromise

Network

Indicator	Type	Context
<code>vjdklet.duckdns.org</code>	Domain	C2 server (Nov10/Nov19)
<code>vigroup2125.duckdns.org</code>	Domain	C2 server (Nov24)

Indicator	Type	Context
45.58.143.254	IPv4	C2 IP (Sharktech, Amsterdam NL)
45.58.143.254:7575	Endpoint	Tria.ge behavioral endpoint (Nov10/Nov19 payloads)
213.227.152.82:2125	Endpoint	Tria.ge behavioral endpoint (Nov24 payload)
7575/tcp	Port	C2 port (Nov10/Nov19)
2125/tcp	Port	C2 port (Nov24)
XSXSXSX	String	HTTP POST field separator
PING? / PONG	String	Keepalive pattern

Host

Indicator	Type	Context
d656bcfef98007fcb3e2be4430a9b24d258c046b0c768ac43699436aceb98e6	SHA256	Nov10 assembly
e2c5921e5c354000c38653d05ac3255865d20bc986da34e68246da6f72ec1fed	SHA256	Nov19 assembly
cbd4cd4c42b3ee0bfb99a254a97ff2c2aacc47d83e08601ae8cdf3b333f07806	SHA256	Nov24 assembly
<Violet>	Mutex	RAT mutex tag
3d847c5c-4f5a-4918-9e07-a96cea49048d	GUID	Mutex / install filename
89c43fcf-5e52-4be7-a719-a26139ce636a	GUID	Secondary mutex
VioletClient.exe	Filename	Original assembly name (from PE headers)

Indicator	Type	Context
ynNENIL	String	XOR dispatch key (Nov10/Nov19)
XSRXSX	String	XOR dispatch key (Nov24)
aXTyo1HpFXkKUYoL	String	Mutex (Nov10/Nov19)
H3n0qlXPeIv1umQI	String	Mutex (Nov24)
USB.exe	Filename	USB spreader
WinTempClean32.bat	Filename	Cleanup script
\ngrok.exe	Path	Tunneling binary

Behavioral

- HTTP POST traffic to port 7575 with `XSXSXSX` field separator
- Tria.ge payload behavioral signature: `Suspicious use of AdjustPrivilegeToken`
- Tria.ge loader behavioral signatures: `Suspicious use of SetWindowsHookEx`, `Modifies registry class`, `Enumerates physical storage devices`
- `PING?` / `PONG` keepalive cycle
- Rotating User-Agent strings across 6 platform profiles per session
- Registry writes under `HKEY_CURRENT_USER\SOFTWARE\` and `HKEY_LOCAL_MACHINE\software\classes\`
- Hidden VNC session creation
- `.lnk` file creation on removable drives (USB spreading)
- ngrok process execution
- Windows Defender tampering (`WDKillerNew`)
- File encryption operations matching ransomware behavior
- WMI query `Select * from AntivirusProduct` (namespace `\root\SecurityCenter2`)
- `ShowSuperHidden` registry value set to 0 under `Explorer\Advanced`
- `taskkill.exe /pid` for targeted process termination
- JPEG screenshot capture with region bounds

Conclusion

Nothing about this binary is hard to reverse. The XOR key is in the binary, the encoding is stacked base64, and the C2 protocol is plaintext HTTP POST with no TLS. The difficulty isn't obfuscation -- it's that the interesting code isn't here. The 8,000-line command handler is a switchboard. Every capability worth analyzing lives in a plugin DLL that

only exists on the C2 server, delivered at attack time and loaded once into memory. We can map every command the RAT accepts, but we can't see what those commands actually do once the plugin takes over.

The 11 encrypted blobs in the config class make this worse. They're initialized in the constructor, never referenced by the dispatcher, and don't decrypt with any key in the binary. They're there for the plugins to read -- and without the plugins, they stay opaque.

This post is the third in the series. [PureLogs](#) covered the plugin stager and crypto-stealing fat client. [PureCrypter](#) covered the loader infrastructure. [Remcos Banking Fraud & Autolt Persistence](#) covers the Autolt-based persistence chains that kept everything alive -- including the discovery that PureHVNC shares exact C2 infrastructure with PureLogs. VioletWorm is the payload the attacker put on the Sharktech Amsterdam server alongside the information stealer, and the one they deployed in every wave from November onward.

Kirk

I like the internet. Want to get in touch? ·