

RONINGLOADER: DragonBreath's New Path to PPL Abuse

 elastic.co/security-labs/roningloader

Jia Yu Chan, Salim Bitam

November 15, 2025

• [Jia Yu Chan](#) • [Salim Bitam](#)

Elastic Security Labs uncovers RONGLOADER, a multi-stage loader deploying DragonBreath's updated gh0st RAT variant. The campaign weaponizes signed drivers, thread-pool injection, and PPL abuse to disable Defender and evade Chinese EDR tools.

 28 min read  [Malware Analysis](#)

 RONGLOADER: DragonBreath's New Path to PPL Abuse

Introduction

Elastic Security Labs identified a recent campaign distributing a modified variant of the gh0st RAT, attributed to the Dragon Breath APT (APT-Q-27), through trojanized NSIS installers masquerading as legitimate software such as Google Chrome and Microsoft Teams. The infection chain employs a multi-stage delivery mechanism that leverages various evasion techniques, with many redundancies aimed at neutralising endpoint security products popular in the Chinese market. These include bringing a legitimately signed driver, deploying custom [WDAC](#) policies, and tampering with the Microsoft Defender binary through PPL abuse.

This campaign primarily targets Chinese-speaking users and demonstrates a clear evolution in adaptability compared to earlier DragonBreath-related campaigns documented in 2022-2023. Through this report, we hope to raise awareness of new techniques this malware is starting to implement and to shine a light on a unique loader we are naming RoningLoader.

Key takeaways

- The malware employs an abuse of Protected Process Light (PPL) to disable Windows Defender
- Threat actors leverage a valid, signed kernel driver to kill processes
- Custom unsigned WDAC policy applied to block 360 Total Security and Huorong executables
- Phantom DLLs and payload injection via thread pools for further antivirus process termination
- Final payload has minor updates and is associated with DragonBreath

Discovery

In August 2025, [research](#) was published detailing a method to abuse Protected Process Light (PPL) to disable endpoint security tooling. Following this disclosure, we produced a behavioral rule, [Potential Evasion via ClipUp Execution](#), and, after some threat hunting of telemetry data, we identified a live campaign employing the technique.

RONINGLOADER code analysis

The initial infection vector is a trojanized installer created using the [Nullsoft Scriptable Install System \(NSIS\)](#). NSIS is a legitimate, open-source tool for creating Windows installers, but it is frequently abused by threat actors to package and deliver malware, as seen in [GULoader](#). In this campaign, we have observed the malicious installers being distributed under various

themes, masquerading as legitimate software such as Google Chrome, Microsoft Teams, or other trusted applications to lure users into executing them.

Upon execution, the [primary NSIS installer](#) functions as a dropper, containing two additional embedded NSIS installers. One of the nested installers is benign and installs the legitimate software, while the second is malicious and responsible for deploying the attack chain.



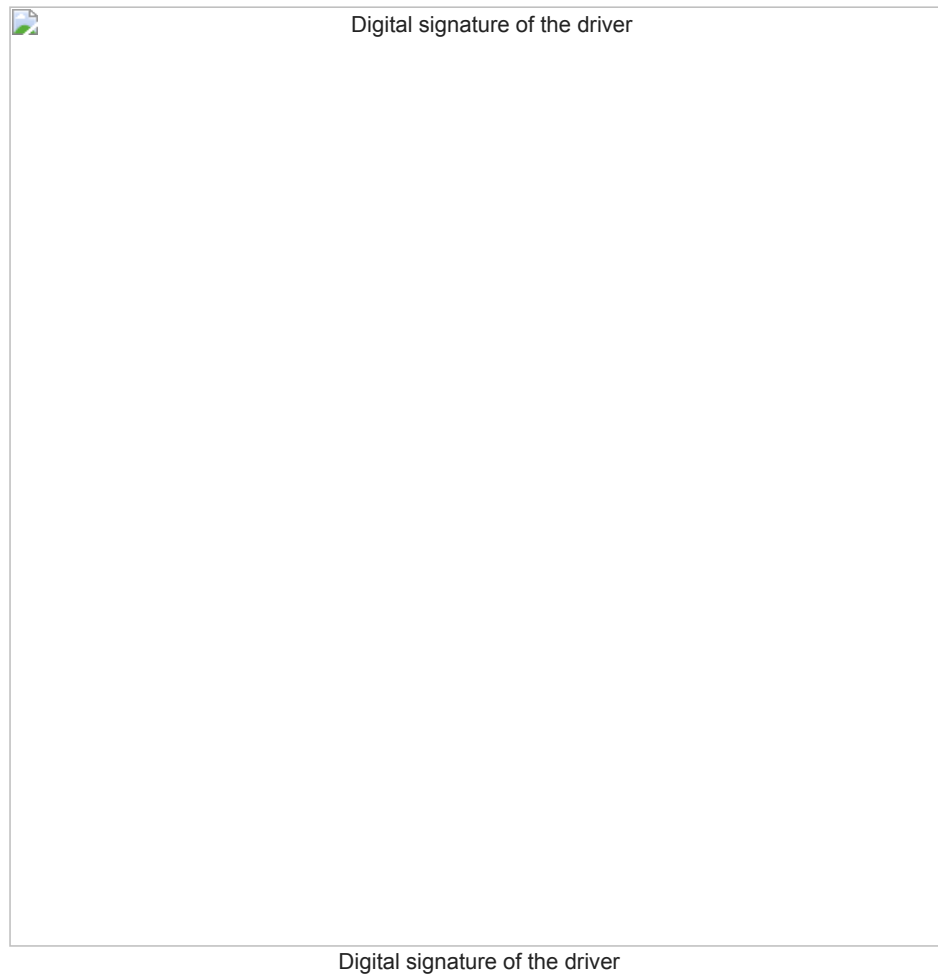
RONINGLOADER Execution flow

The attack chain leverages a signed driver named `ollama.sys` for antivirus process termination. The driver has a signer name of `Kunming Wuqi E-commerce Co., Ltd.`, with a certificate valid from February 3, 2025, to February 3, 2026. Pivoting on VirusTotal revealed 71 additional signed binaries. Among these, we identified AgentTesla droppers masquerading as `慕讯公益加速器 (MuXunAccelerator)`, a gaming-focused VPN software popular among Chinese users, with samples dating back to April 2025. Notably, the signing techniques vary across samples. Some earlier samples, like `inject.sys`, contain `HookSignTool` artifacts including the string `JemmyLoveJenny`, while the October 2025 `ollama.sys` sample shows no such artifacts and uses standard signing procedures, yet both share the same certificate validity period.

Comparing `ollama.sys`'s PDB string artifact `D:\VS_Project\加解密\MyDriver1\x64\Release\MyDriver1.pdb` with other samples, we discovered different artifacts from other submitted samples -

- `D:\cpp\origin\ConsoleApplication2\x64\Release\ConsoleApplication2.pdb`
- `D:\a_work\1\s\artifacts\obj\coreclr\windows.x86.Release\Corehost.Static\singlefilehost.pdb`
- `C:\Users\0\Desktop\EAMap\x64\Release\ttt.pdb`
- `h:\projects\netfilter3\bin\Release\Win32\nfregdrv.pdb`

Due to the diversity of binaries and the large volume of submissions, we suspect the certificate may have been leaked, but this is speculation at this time.



Stage 1

Our analysis began with the initial binary, identified by its SHA256 hash:

`da2c58308e860e57df4c46465fd1cfc68d41e8699b4871e9a9be3c434283d50b`. Extracting it reveals two embedded executables: a benign installer, `letsvpnlatest.exe`, and the malicious installer `Snieoatwtregoable.exe`.

The malicious installer, `Snieoatwtregoable.exe`, creates a new directory at `C:\Program Files\Snieoatwtregoable\`. Within this folder, it drops two files: a DLL named `Snieoatwtregoable.dll` and an encrypted file, `tp.png`.



Files dropped on disk

Files dropped on disk

The core of the malicious activity resides within `Snieoatwtregoable.dll`, which exports a single function: `DllRegisterServer`. When invoked, this function reads the contents of the `tp.png` file from disk, then decrypts this data using a simple algorithm involving both a Right Rotate (ROR) and an XOR operation.



The decrypted content is shellcode that reflectively loads and executes a PE file in memory. The malware first allocates a new memory region within its own process using the `NtAllocateVirtualMemory` API, then creates a new thread to execute the shellcode by calling `NtCreateThreadEx`.

The malware attempts to remove any userland hooks by loading a fresh new `ntdll.dll`, then using `GetProcAddress` with the API name to resolve the addresses.



Loads a fresh NTDLL

Loads a fresh NTDLL

The malware attempts to connect to localhost on port 5555 without serving any real purpose, as the result will not matter; speculatively, this is likely dead code or pre-production leftover code



Dead code

Dead code

Stage 2 - tp.png

RONINGLOADER first checks whether it has administrative privileges using the `GetTokenInformation` API. If not, it attempts to elevate its privileges by using the `runas` command to launch a new, elevated instance of itself before terminating the original process.



Elevates privileges with RunAs command

Interestingly, the malware tries to communicate with a hardcoded URL <http://www.baidu.com/> with the user-agent "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko", but this appears to be dead code, likely due to either a removed feature or placeholder code for future versions. It is designed to extract and log the HTTP response header date from the URL.

The malware then scans a list of running processes for specific antivirus solutions. It checks against a hardcoded list of process names and sets a corresponding boolean flag to "True" if any are found.



Scans for specific processes

Scans for specific processes

The following is a table of processes and the associated security products hardcoded in the binary:

Process name	Security Product
MsMpEng.exe	Microsoft Defender Antivirus
kxemain.exe	Kingsoft Internet Security
kxetray.exe	Kingsoft Internet Security
kxecenter.exe	Kingsoft Internet Security
QQPCTray.exe	Tencent PC Manager
QQPC RTP.exe	Tencent PC Manager
QMToolWidget.exe	Tencent PC Manager
HipsTray.exe	Qihoo 360 Total Security
HipsDaemon.exe	Qihoo 360 Total Security
HipsMain.exe	Qihoo 360 Total Security
360tray.exe	Qihoo 360 Total Security

AV process termination via injected remote process

Next, the malware kills those processes. Interestingly, the Qihoo 360 Total Security product takes a different approach than the others.

First, it blocks all network communication by changing the firewall. It then calls a function to inject shellcode into the process (`vssvc.exe`) associated with the Volume Shadow Copy (VSS) service.

It first grants itself the high integrity `SeDebugPrivilege` token.



Grants SeDebugPrivilege to itself

Grants SeDebugPrivilege to itself

It then starts the [VSS \(Volume Shadow Copy Service\)](#) if it is not already running and fetches the PID of its associated process (vssvc.exe).



Starts VSS service

Starts VSS service

Next, the malware uses `NtCreateSection` to create two separate memory sections. It then maps views of these sections into the memory space of the `vssvc.exe` process. The first section contains a full Portable Executable (PE) file, which is a driver with the device name `\\.\Ollama`. The second section contains shellcode intended for execution.



Mapping section views to the remote process

RONINGLOADER takes a different approach to this process injection compared to other injection methods used elsewhere in the malware. This technique leverages the thread pool to remotely execute code via a file write trigger in the remote process. This technique was documented [by SafeBreach](#) in 2023 with different variants.



Injection through ThreadPool tasks

Once executed, the shellcode begins by dynamically resolving the addresses of the Windows APIs it needs to function. This is the only part of RONGLOADER that employs any obfuscation, using the [Fowler–Noll–Vo hash](#) (FNV) algorithm to look up functions by hash instead of by name.



It first fetches the addresses of `CreateFileW`, `WriteFile`, and `CloseHandle` to write the driver to disk to a hardcoded path, `C:\windows\system32\drivers\1912763.temp`.

Then it performs the following operations:

- Create a service named `xererre1` to load the driver dropped to disk

- For each of the following processes (`360Safe.exe`, `360Tray.exe`, and `ZhuDongFangYu.exe`), which are all associated with Qihoo 360 software, it calls 2 functions: one to find the PID of the process by name, followed by a function to kill the process by PID
- It then stops and deletes the service `xererre1`



Function calls to kill Qihoo 360 software processes

To kill a process, the malware uses the driver. An analysis of the driver reveals that it registers only 1 functionality: it handles one IOCTL ID (**0x222000**) that takes a PID as a parameter and kills the process by first opening it with **ZwOpenProcess**, then terminating it with **ZwTerminateProcess** kernel APIs.



Kernel driver kills a process by PID

Kernel driver kills a process by PID

AV process termination

Returning to the main execution flow, the malware enters a loop to confirm the termination of `360tray.exe`, as handled by the shellcode injected into the VSS service. It proceeds only after verifying that the process is no longer running. Immediately after this confirmation, the system restores its firewall settings. This action is likely a defensive measure intended to sever the software's communication channel, preventing it from uploading final activity logs or security alerts to its backend services.

It then terminates the other security processes directly from its main process. Notably, it makes no attempt to hide these actions, abandoning the earlier API hashing technique and calling the necessary functions directly.



RONINGLOADER follows a consistent, repeatable procedure to terminate its target processes:

- First, it writes the malicious driver to disk, this time to the temporary path `C:\Users\analysis\AppData\Local\Temp\ollama.sys`.
- A temporary service (`ollama`) is created to load `ollama.sys` into the kernel
- The malware then fetches the target process's PID by name and sends a request containing the PID to its driver to perform the termination.
- Immediately after the kill command is sent, the service is deleted.



Write driver, create service, start service

Write driver, create service, start service



Kill by PID and delete the service afterwards

Kill by PID and delete the service afterwards

Regarding Microsoft Defender, the malware attempts to kill the `MsMpEng.exe` process using the same approach described above. We noticed a code bug from the author: for Microsoft Defender, the code does not check whether Defender is already running, but proceeds directly to searching for the `MsMpEng.exe` process. This means that if the process is not running, the malware will send 0 as the PID to the driver.



Microsoft Defender process killing

The malware has more redundant code to kill security solution processes. It also injects another shellcode into svchost.exe, similar to what was injected into `vssvc.exe`, but the list of processes is different, as seen in the screenshot below.



The injection technique also uses threadpools, but the injected code is triggered by an event.



After the process termination, the malware creates 4 folders

- C:\ProgramData\lnk
- C:\ProgramData\<current_date>
- C:\Users\Public\Downloads\<current_date>
- C:\ProgramData\Roning



Folder creation to drop files

Folder creation to drop files

Embedded archives

The malware then writes three `.txt` files to `C:\Users\Public\Downloads\<current_date>`. Despite their extension, these are not text files but rather containers built with a specific format, likely adapted from another code base.

This custom file structure is organized as follows:

- **Magic Bytes:** The file begins with the signature `4B 44 01 00` for identification.
- **File Count:** This is immediately followed by a value indicating the number of files encapsulated within the container.
- **File Metadata:** A header section then describes the information for each stored file.
- **Compressed Data:** Finally, each embedded file is stored in a ZLIB-compressed data block.

Here's an example file format for the `hjk.txt` archive, which contains 2 files: `1.bat` and `fhq.bat`.

This archive format applies to 2 other embedded files in the current stage:

- `agg.txt`, which contains 3 files - `Enpug.bin`, `goldendays.dll`, and `trustinstaller.bin`
- `kill.txt`, which contains 1 file - `1.dll`



Batch scripts to bypass UAC and AV networking

1. `bat` is a simple batch script that disables User Account Control (UAC) by setting the `EnableLUA` registry value to 0.



1.bat content

`fhq.bat` is another batch script that targets the program defined in `C:\ProgramData\lnk\123.txt` and the Qihoo 360 security software (360Safe.exe) by creating firewall rules that block inbound and outbound connections to them. It also disables firewall notifications across all profiles.



fhq.bat content

AV process termination via Phantom DLL

The deployed DLL, `1.dll`, is copied to `C:\Windows\System32\Wow64\Wow64Log.dll` to be side-loaded by any WOW64 processes, as `Wow64Log.dll` is a [phantom DLL](#) that is not present on Windows machines by default. Its task is redundant, essentially attempting to kill a list of processes using standard Windows APIs (`TerminateProcess`).



Wow64Log.dll Dllmain code

Wow64Log.dll Dllmain code

ClipUp MS Defender killer

The malware then attempts to use a PPL abuse technique documented by [Zero Salarium](#) in August 2025. The article's PoC targets Microsoft Defender only. Note that all of the system commands executed are through `cmd.exe` with the `ShellExecuteW` API

- It searches for Microsoft Defender's installation folder under `C:\ProgramData\Microsoft\Windows Defender\Platform\*`, targeting only the directory with the most recent modification date, which indicates the currently used version

- Create a folder `C:\ProgramData\roming` and a directory link with `mklink` to point to the directory found with the following command: `cmd.exe /c mklink /D "C:\ProgramData\roming" "C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.25050.5-0"`
- It then runs `C:\Windows\System32\ClipUp.exe` with the following parameter: `-pp1 C:\ProgramData\roming\MsMpEng.exe`, which overwrites `MsMpEng.exe` with junk data, effectively disabling the EDR even after a restart

The author appears to have copied code from [EDR-Freeze](#) to start `ClipUp.exe`.

CiPolicies

The malware directly targets Windows Defender Application Control (WDAC) by writing a policy file to the path

`C:\Windows\System32\CodeIntegrity\CiPolicies\Active\{31351756-3F24-4963-8380-4E7602335AAE}.cip`.



Write policy to disk

The malicious policy operates in a “deny-list” mode, allowing most applications to run while explicitly blocking two popular Chinese antivirus vendors:

- Qihoo 360 Total Security by blocking 360rp.exe and 360sd.exe
- Huorong Security by blocking ARPProte.exe
- All executables signed by Huorong Security (北京火绒网络科技有限公司) via certificate TBS hash
A229D2722BC6091D73B1D979B81088C977CB028A6F7CBF264BB81D5CC8F099F87D7C296E48BF09D7EBE275F5498661A4

A critical component is the Enabled:Unsigned System Integrity Policy rule, which allows the policy to be loaded without a valid digital signature.

Truncated...

```
<Rule>
  <Option>Enabled:Inherit Default Policy</Option>
</Rule>
<Rule>
  <Option>Enabled:Unsigned System Integrity Policy</Option>
</Rule>
<Rule>
  <Option>Enabled:Advanced Boot Options Menu</Option>
</Rule>
<Rule>
  <Option>Enabled:Update Policy No Reboot</Option>
</Rule>
</Rules>
<EKUs />
<FileRules>
  <Allow ID="ID_ALLOW_A_019A298478CE7BF4902DE08CA2D17630" FileName="" />
  <Allow ID="ID_ALLOW_A_019A298478CE7AB089C369772F34B39B" FileName="" />
  <Deny ID="ID_DENY_A_019A298478CE7DBA9913BFC227DACD14" FileName="360rp.exe" InternalName="360rp.exe"
FileDescription="360杀毒 实时监控" ProductName="360杀毒" />
  <Deny ID="ID_DENY_A_019A298478CE763C85C9F42EC8669750" FileName="360sd.exe" InternalName="360sd.exe"
FileDescription="360杀毒 主程序" ProductName="360杀毒" />
  <FileAttrib ID="ID_FILEATTRIB_A_019A298478CE766B9C39FB9CE6805A11" FileName="ARPProte.exe"
MinimumFileVersion="6.0.0.0" />
</FileRules>
<Signers>
  <Signer ID="ID_SIGNER_A_019A298478CE7608908CAE58FD9C3D8E" Name="">
    <CertRoot Type="TBS"
Value="A229D2722BC6091D73B1D979B81088C977CB028A6F7CBF264BB81D5CC8F099F87D7C296E48BF09D7EBE275F5498661A4" />
    <CertPublisher Value="北京火绒网络科技有限公司" />
    <FileAttribRef RuleID="ID_FILEATTRIB_A_019A298478CE766B9C39FB9CE6805A11" />
  </Signer>
  <Signer ID="ID_SIGNER_A_019A298478CE77F7B523D1581F518639" Name="">
    <CertRoot Type="TBS"
Value="A229D2722BC6091D73B1D979B81088C977CB028A6F7CBF264BB81D5CC8F099F87D7C296E48BF09D7EBE275F5498661A4" />
    <CertPublisher Value="北京火绒网络科技有限公司" />
  </Signer>
</Signers>
...Truncated
```

Stage 3 - goldendays.dll

In the previous stage, RONINGLOADER creates a new service named `MicrosoftSoftware2ShadowCop4yProvider` to run the next stage of execution with the following command: `regsvr32.exe /S "C:\ProgramData\Roning\goldendays.dll`.



The primary goal of this component is to inject the next payload into a legitimate, high-privilege system process to camouflage its activities.

To achieve this, RONINGLOADER first identifies a suitable target process. It has a hardcoded list of two service names that it attempts to start sequentially:

1. TrustedInstaller (`TrustedInstaller.exe`)

2. MicrosoftEdgeElevationService (`elevation_service.exe`)

The malware iterates through this list, attempting to start each service. Once a service is successfully started, or if one is found already running, the malware saves its Process ID (PID) for the injection phase.



Start both TrustedInstaller and MicrosoftEdgeElevationService services

Start both TrustedInstaller and MicrosoftEdgeElevationService services

Next, the malware establishes persistence by creating a batch file with a random name within the `C:\Windows\` directory (e.g., `C:\Windows\KPeYvogsPm.bat`). The script inside this file runs a continuous loop with the following logic:

- It checks if the captured PID of the trusted service (e.g., PID `4016` for `TrustedInstaller.exe`) is still running
- If the service is not running, the script restarts the previously created malicious service (`MicrosoftSoftware2ShadowCop4yProvider`) to ensure the malware's components remain active
- If the service process is running, the script sleeps for 10 seconds before checking again



Batch file content

Finally, the malware reads the contents of `C:\ProgramData\Roning\trustinstaller.bin`. Using the PID of the trusted service it acquired earlier, it injects this payload into the target process (`TrustedInstaller.exe` or `elevation_service.exe`). The injection method is straightforward: it performs a remote virtual allocation with `VirtualAllocEx`, writes to it with `WriteProcessMemory`, and then creates a remote thread to execute it with `CreateRemoteThread`.





Remote process injection

Stage 3 - trustinstaller.bin

The third stage, contained within `trustinstaller.bin`, is responsible for injecting the final payload into a legitimate process. It starts by enumerating running processes and searching for a target by matching process names against a hardcoded list of potential processes.



When found, it will inject the shellcode into `C:\ProgramData\Roning\Enpug.bin`, which is the final payload. It will create a section with `NtCreateSection`, map a view of it in the remote process with `NtMapViewOfSection`, and write the payload to it. Then it will create a remote thread with `CreateRemoteThread`.



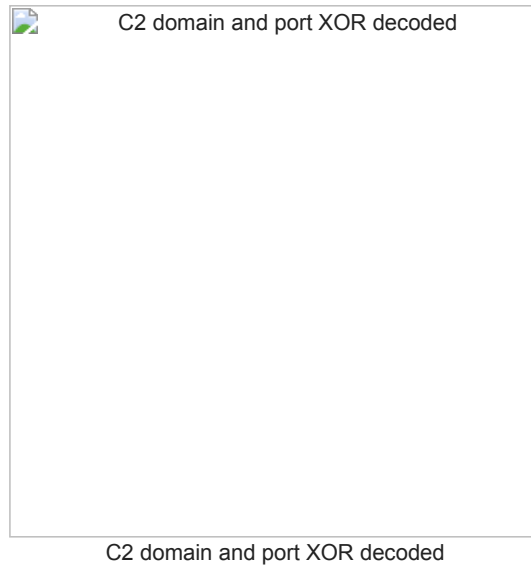
Stage 4 - Final Payload

The [final payload](#) has not undergone major changes since [Sophos's](#) discovery of a DragonBreath campaign in 2023 and [QianXin's report](#) in mid-2022. It is still a modified version of the open-source [gh0st](#) RAT.

In the more recent campaigns, a mutex of value `Global\DHGGlobalMutex` is created at the very beginning of execution. Outside the main C2 communication loop, dead code is observed creating a mutex named `MyUniqueMutexName` and immediately destroying it afterward.



The C2 domain and port remain hardcoded but are now XOR-encrypted. The C2 channel operates over raw TCP sockets with messages encrypted in both directions.



Victim Beacon Data

The implant checks in with the C2 server and repeatedly beacons to the C2 at random intervals, implemented through `Sleep(<random_amount> * 1000)`. Below is the structure for the data that the implant returns to the C2 server during the beaconing interval:

```

struct BeaconData {

    // +0x000

    uint32_t message_type;           // Example Beacon ID - 0xC8 (200)

    // +0x004

    uint32_t local_ip;              // inet_addr() of victim's IP

    // +0x008

    char hostname[50];              // Computer name or registry "Remark"

    // +0x03A

    char windows_version[?];        // OS version info

    // +0x0D8

    char cpu_name[64];              // Processor name

    // +0x118

    uint32_t entry_rdx;

    // +0x11C

    char time_value[64];            // Implant installed time or registry "Time" value

    // +0x15C

```

```
char victim_tag[39];           // Command 6 buffer (Custom victim tag)


// +0x183


uint8_t is_wow64;             // 1 if 32-bit on 64-bit Windows


// +0x184


char av_processes_found[128]; // Antivirus processes found


// +0x204


char uptime[12];             // System uptime


char padding[52];


// +0x244


char crypto_wallet_track[64]; // "狐狸系列" (MetaMask) or registry "ZU" (crypto related tracking)


// +0x284


uint8_t is_admin;            // 1 if running with admin rights


// +0x285


char data[?];


// +0x305
```

```

uint8_t telegram_installed;    // 1 if Telegram installed

// +0x306

uint8_t telegram_running;      // 1 if Telegram.exe running

// +0x307

// (padding to 0x308 bytes)

};

```

C2 commands

Request messages sent from the C2 server to the implant follow the structure:

```

struct C2_to_implant_msg {

    uint32_t total_message_len;

    uint32_t RC4_key;

    char encrypted_command_id;

    uint8_t encrypted_command_args;

};

```

The implant decrypts C2 messages through the following formula:

```
RC4_decrypt(ASCII(decimal(RC4_key)), encrypted_command_id || command)
```

Below is a list of available commands that, for the most part, remain the same as 2 years ago:

Command ID	Description
0	ExitWindowsEx via a supplied EXIT_WINDOWS_FLAGS
1	Terminate implant gracefully
2	Set registry key Enable to False to terminate & disable implant persistently
3	Set registry key Remark for custom victim renaming (default value: hostname)
4	Set registry key ZU for MetaMask / crypto-related tagging

Command ID	Description
5	Clear Windows Event logs (Application, Security, System)
6	Set additional custom tags when client beacons
7	Download and execute file via supplied URL
9	ShellExecute (visible window)
10	ShellExecute (hidden window)
112	Get clipboard data
113	Set clipboard data
125	ShellExecute cmd.exe with command parameters (hidden window)
126	Execute payload by dropping to disk or reflectively load and execute PluginMe export
128	First option - open a new session with a supplied C2 domain, port, and beacon interval. Second option - set registry key CopyC to update C2 domain and port permanently. Stored encrypted via Base64Encode(XOR(C2_domain_and_port, 0x5)).
241	Check if Telegram is installed and/or running
243	Configure Clipboard Hijacker
101, 127, 236, [...]	Custom shellcode injection into svchost.exe using WTS session token impersonation, falling back to CREATE_SUSPENDED process injection via CreateRemoteThread

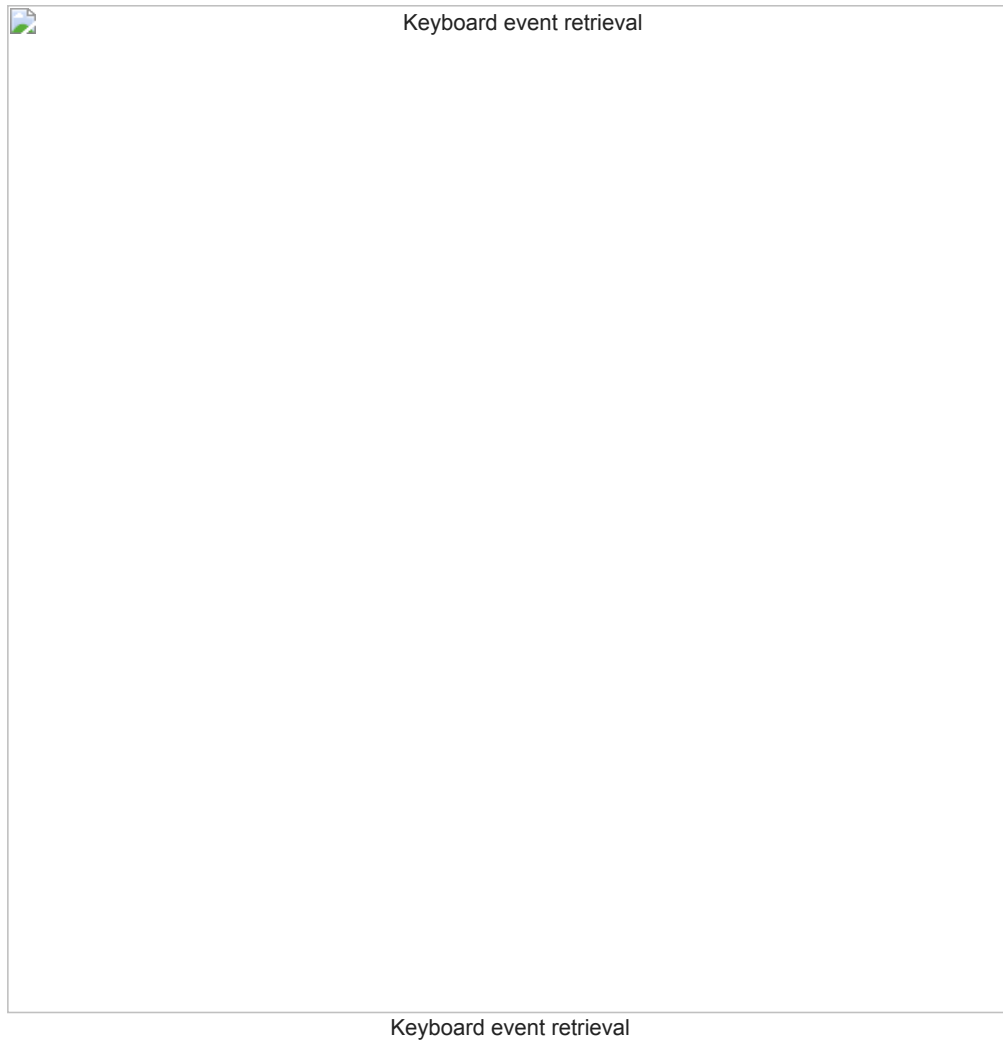
Analyst note: There are multiple command IDs that point to the same command. We used an ellipsis to identify when this was observed.

System Logger

In addition to the C2 commands, the implant implements a keystroke, clipboard, and active-window logger. Captured data is written to %ProgramData%\microsoft.dotnet.common.log and can be enabled or disabled via a registry key at HKEY_CURRENT_USER\offlinekey\open (1 to enable, 0 to disable). The log file implements automatic rotation, deleting itself when it exceeds 50 MB to avoid detection through excessive disk usage.

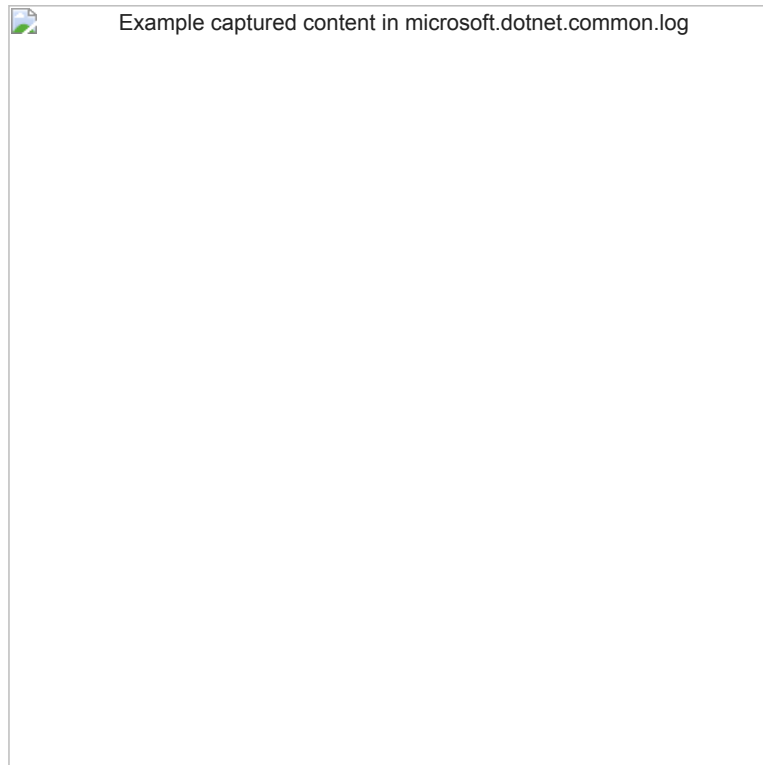
The code snippet below demonstrates the initialization routine that implements log rotation and configures a DirectInput8 interface to acquire the keyboard device for event capture, followed by the keyboard event retrieval logic.





The malware then enters a monitoring loop to capture three categories of information.

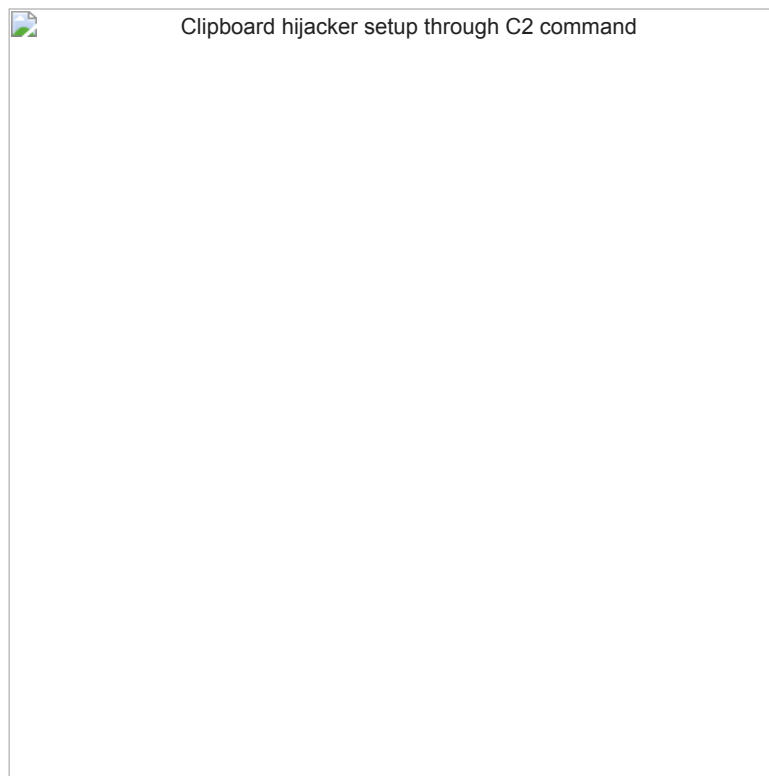
- First, it monitors the clipboard using `OpenClipboard` and `GetClipboardData`, logging any changes to text content with the prefix [剪切板:].
- Second, it tracks window focus changes via `GetForegroundWindow`, logging the active window title and timestamp with the prefixes [标题:] and [时间:], respectively, whenever the user switches applications.
- Third, it retrieves buffered keyboard events from the `DirectInput8` device (up to 60 events per poll) and translates them into readable text through a character mapping table, prepending the results with a prefix [内容:].



Example captured content in microsoft.dotnet.common.log

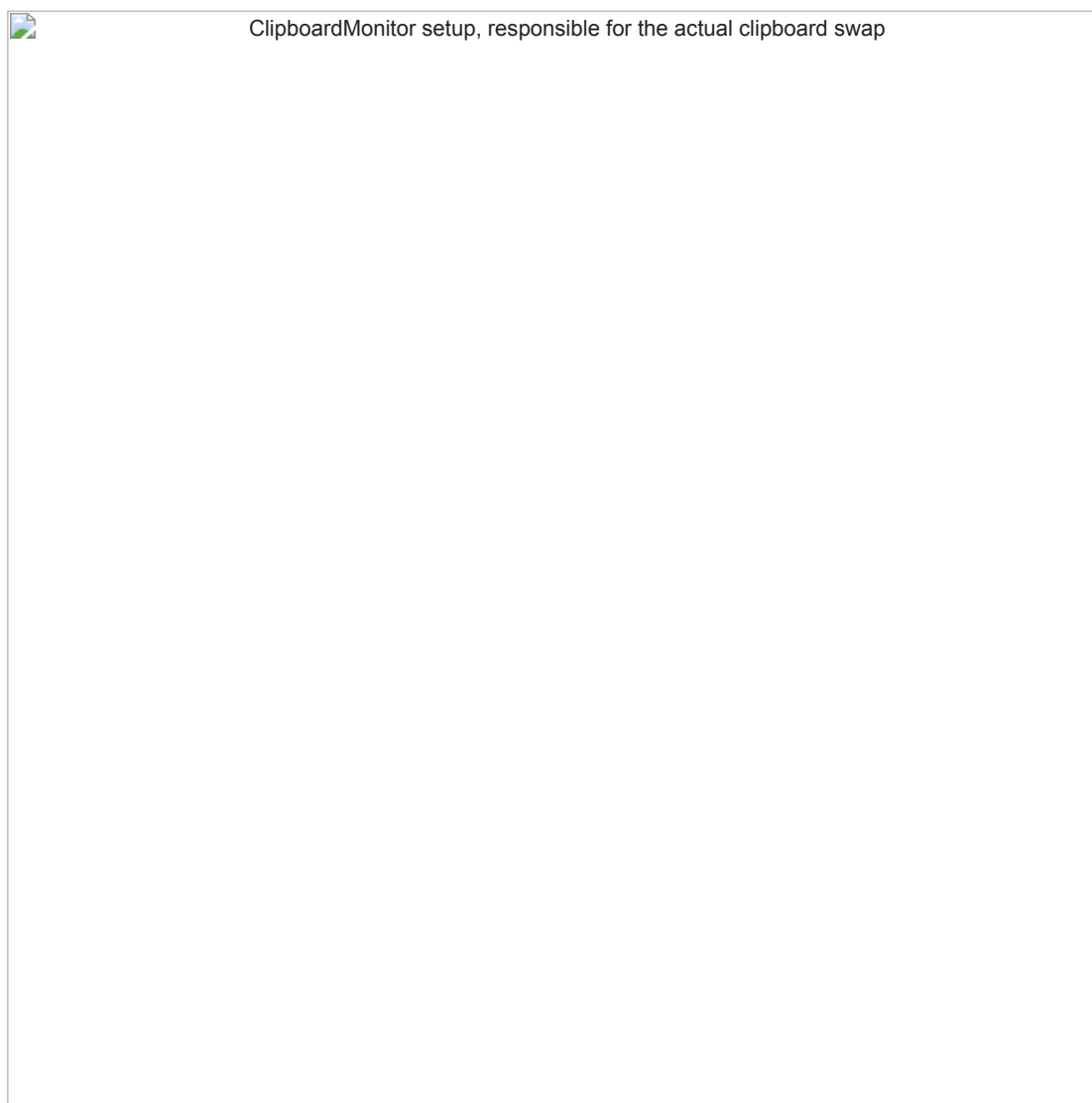
Clipboard Hijacker

The malware also implements a clipboard hijacker that is remotely configured through C2 command ID 243. It monitors clipboard changes and performs search-and-replace operations on captured text, substituting attacker-defined strings with replacement values. Configuration parameters are stored in the registry under `HKEY_CURRENT_USER\offlinekey` with keys `clipboard` (enable/disable feature), `charac` (search string), `characLen` (search length), and `newcharac` (replacement string).



Clipboard hijacker setup through C2 command

It registers a window class named `ClipboardListener_Class_Toggle` and creates a hidden window titled `ClipboardMonitor` to receive clipboard change notifications. The window procedure handles `WM_CLIPBOARDUPDATE (0x31D)` messages by verifying clipboard sequence numbers with `GetClipboardSequenceNumber` to detect genuine changes, then invoking the core manipulation routine, which swaps the clipboard content via `EmptyClipboard` and `SetClipboardData`.



ClipboardMonitor setup, responsible for the actual clipboard swap

Malware and MITRE ATT&CK

Elastic uses the [MITRE ATT&CK](#) framework to document common tactics, techniques, and procedures that advanced persistent threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- [Execution](#)
- [Persistence](#)
- [Privilege Escalation](#)
- [Defense Evasion](#)
- [Credential Access](#)
- [Discovery](#)

- [Collection](#)
- [Command and Control](#)

Techniques

Techniques represent how an adversary achieves a tactical goal by performing an action.

- [Command and Scripting Interpreter: Windows Command Shell](#)
- [System Services: Service Execution](#)
- [Create or Modify System Process: Windows Service](#)
- [Abuse Elevation Control Mechanism: Bypass User Account Control](#)
- [Access Token Manipulation](#)
- [Impair Defenses: Disable or Modify Tools](#)
- [Impair Defenses: Disable or Modify System Firewall](#)
- [Indicator Removal: Clear Windows Event Logs](#)
- [Hijack Execution Flow: DLL Side-Loading](#)
- [Process Injection](#)
- [Masquerading: Match Legitimate Name or Location](#)
- [Modify Registry](#)
- [Subvert Trust Controls: Code Signing Policy Modification](#)
- [Input Capture: Keylogging](#)
- [Clipboard Data](#)
- [Process Discovery](#)
- [System Information Discovery](#)
- [System Owner/User Discovery](#)
- [Software Discovery: Security Software Discovery](#)
- [Non-Application Layer Protocol](#)
- [Encrypted Channel: Symmetric Cryptography](#)

Mitigations

Detection

- [Potential Evasion via ClipUp Execution](#)
- [Suspicious Remote Memory Allocation](#)
- [Potential Suspended Process Code Injection](#)
- [Remote Memory Write to Trusted Target Process](#)
- [Remote Process Memory Write by Low Reputation Module](#)
- [Process Memory Write to a Non Child Process](#)
- [Unbacked Shellcode from Unsigned Module](#)
- [UAC Bypass Attempt via WOW64 Logger DLL Side-Loading](#)
- [Network Connect API from Unbacked Memory](#)
- [Rundll32 or Regsvr32 Loaded a DLL from Unbacked Memory](#)
- [Network Module Loaded from Suspicious Unbacked Memory](#)

YARA

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify RONINGLOADER and the final implant:

- [Windows.Trojan.RoningLoader](#)
- [Windows.Trojan.DragonBreath](#)

Observations

The following observables were discussed in this research.

Observable	Type	Name	Reference
da2c58308e860e57df4c46465fd1cfc68d41e8699b4871e9a9be3c434283d50b	SHA-256	k1klznuah.msi	Initial NSIS installer
82794015e2b40cc6e02d3c1d50241465c0cf2c2e4f0a7a2a8f880edaee203724	SHA-256	Snieoatwtregoable.exe	Malicious installer unpacked from initial installer
c65170be2bf4f0bd71b9044592c063eaa82f3d43fcbd8a81e30a959bcaad8ae5	SHA-256	Snieoatwtregoable.dll	Stage 1 - loader for stage 2
2515b546125d20013237aeadec5873e6438ada611347035358059a77a32c54f5	SHA-256	ollama.sys	Stage 2 - driver for process termination
1613a913d0384cbb958e9a8d6b00fffaf77c27d348ebc7886d6c563a6f22f2b7	SHA-256	tp.png	Stage 2 - encrypted core payload
395f835731d25803a791db984062dd5cfdcade6f95cc5d0f68d359af32f6258d	SHA-256	1.bat	Stage 2 - UAC bypass script
1c1528b546aa29be6614707cbe408cb4b46e8ed05bf3fe6b388b9f22a4ee37e2	SHA-256	fhq.bat	Stage 2 - script to block networking for AV processes
4d5beb8efd4ade583c8ff730609f142550e8ed14c251bae1097c35a756ed39e6	SHA-256	1.dll	Stage 2 - AV processes termination
96f401b80d3319f8285fa2bb7f0d66ca9055d349c044b78c27e339bcfb07cdf0	SHA-256	{31351756-3F24-4963-8380-4E7602335AAE}.cip	Stage 2 - WDAC policy
33b494eaaa6d7ed75eec74f8c8c866b6c42f59ca72b8517b3d4752c3313e617c	SHA-256	goldendays.dll	Stage 3 - entry point
fc63f5dfc93f2358f4cba18cbdf99578fff5dac4cdd2de193a21f6041a0e01bc	SHA-256	trustinstaller.bin	Stage 3 - loader for Enpug.bin
fd4dd9904549c6655465331921a28330ad2b9ff1c99eb993edf2252001f1d107	SHA-256	Enpug.bin	Stage 3 - loader for final payload
3dd470e85fe77cd847ca59d1d08ec8ccebe9bd73fd2cf074c29d87ca2fd24e33	SHA-256	6uf9i.exe	Stage 4 - final payload

Observable	Type	Name	Reference
qaqkongtiao[.]com	domain-name		Stage 4 - final payload C2

References

The following were referenced throughout the above research:

- https://nsis.sourceforge.io/Main_Page
- <https://learn.microsoft.com/en-us/windows-server/storage/file-server/volume-shadow-copy-service>
- <https://github.com/Jemmy1228/HookSigntool>
- <https://www.safebreach.com/blog/process-injection-using-windows-thread-pools/>
- <https://hijacklibs.net/entries/microsoft/built-in/wow64log.html>
- https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function
- <https://www.zerosalarium.com/2025/08/countering-edrs-with-backing-of-ppl-protection.html>
- <https://github.com/TwoSevenOneT/EDR-Freeze/blob/ceffd5ea7b813b356c77d469561dbb5ee45aeb24/PPLHelp.cpp#L43>
- <https://news.sophos.com/en-us/2023/05/03/doubled-dll-sideload-dragon-breath/>
- <https://ti.qianxin.com/blog/articles/operation-dragon-breath-%28apt-q-27%29-dimensionality-reduction-blow-to-the-gambling-industry/>
- <https://github.com/sin5678/gh0st>

Jump to section

- [Introduction](#)
- [Key takeaways](#)
- [Discovery](#)
- [RONINGLOADER code analysis](#)
- [Stage 1](#)
- [Stage 2 - tp.png](#)
- [AV process termination via injected remote process](#)
- [AV process termination](#)
- [Embedded archives](#)
- [Batch scripts to bypass UAC and AV networking](#)

Elastic Security Labs Newsletter

[Sign Up](#)