

Fake Installers to Monero: A Multi-Tool Mining Operation

 elastic.co/security-labs/fake-installers-to-monero

Jia Yu Chan, Cyril François, Remco Sprooten

March 31, 2026

• [Jia Yu Chan](#) • [Cyril François](#) • [Remco Sprooten](#)

Elastic Security Labs dissects a long-running operation deploying RATs, cryptominers, and CPA fraud through fake installer lures, tracking its evolution across campaigns and Monero payouts.

 18 min read  [Malware Analysis](#), [Threat Intelligence](#), [Enablement](#)

 Fake Installers to Monero: A Multi-Tool Mining Operation

Introduction

Elastic Security Labs has been tracking a financially motivated operation, designated REF1695, that has been active since at least late 2023. The operator deploys a combination of RATs, cryptominers, and custom XMRig loaders through fake installer packages. Across all observed campaigns, the infection chains share a consistent packing technique, overlapping C2 infrastructure, and common social engineering patterns, linking them to a single operator.

Beyond cryptomining, the threat actor monetizes infections through CPA (Cost Per Action) fraud, directing victims to content locker pages under the guise of software registration. In this report, we trace the operation's evolution across multiple campaign builds, analyze the C2 communication protocols, document a previously unreported .NET implant (CNB Bot), and track the operator's financial returns via public Monero mining pool dashboards.

Key takeaways

- Financially motivated campaigns have been active since late 2023, deploying various RATs and cryptominers through fake installer packages.
- Operator monetizes infections through both cryptomining and CPABuild fraud.
- Stages use a consistent Themida/WinLicense + .NET Reactor packing combination
- CNB Bot is a previously undocumented .NET implant with RSA-2048 signed task authentication
- A custom XMRig loader evades detection by killing the miner whenever analysis tools are running and deploys WinRing0x64.sys
- Over 27.88 XMR paid out across four tracked wallets, with active workers at the time of writing
- We leveraged a Claude-driven agentic pipeline to automate the extraction of payload stages and implant configurations

Campaign 1 (CNB Bot)

The most recent campaign involves dropping CNB Bot, using an ISO file as the infection vector. The ISO image contains 2 files: a single-stage .NET Reactor-protected loader further packed with Themida/WinLicense 3.x, and a ReadMe.txt. Associated ISO samples:

- [460203070b5a928390b126fcd52c15ed3a668b77536faa6f0a0282cf1c157162](#)
- [b8b7aecce2a4d00f209b1e4d30128ba6ef0f83bbdc05127f6f8ba97e7d6df291](#)
- [9977b9185472c7d4be22c20f93bc401dd74bb47223957015a3261994d54c59fc](#)
- [9fa23382820b1e781f3e05e9452176a72529395643f09080777fab7b9c6b1f5c](#)
- [27db41f654b53e41a4e1621a83f2478fa46b1bbffc1923e5070440a7d410b8d3](#)

The ReadMe.txt serves as a social engineering lure, framing the unsigned binary as the product of a small non-profit team that cannot afford EV code-signing, then provides explicit instructions to bypass SmartScreen via "More Info" → "Run Anyway."



Using the open-source Themida/Winlicense unpacker project, [Unlicense](#), we automatically extracted the .NET Reactor-protected loader and then passed it through [NETReactorSlayer](#) for deobfuscation. The majority of campaigns were observed to use this combination of protection in both the initial and subsequent stages.

The loader first invokes PowerShell with `-WindowStyle Hidden`, to register broad Microsoft Defender exclusions via `Add-MpPreference -ExclusionPath` and `Add-MpPreference -ExclusionProcess`, covering the loader itself, staging directories (`%TEMP%`, `%LocalAppData%`, `%AppData%`) and a set of LOLBin process names the malware later utilizes.



It then extracts an embedded .NET assembly resource and writes it to disk at `%TEMP%\MLPCInstallHelper.exe` (filename varies by build), then executes it via PowerShell. This embedded resource is a .NET Reactor-protected CNB Bot instance, discussed in detail in the **Code Analysis - CNB Bot** section below.

Since no legitimate software is installed at any point, the loader presents a fake error dialog to the user, attributing the installation failure to unmet system requirements.



Campaign 2 (PureRAT)

Pivoting on the ReadMe.txt lure content, we discovered a campaign dropping PureRAT v3.0.1. This campaign uses a very similar initial-stage loader as campaign 1 and introduces a second-stage loader.

Example ISO samples employing this chain:

- 7bb0e91558244bcc79b6d7a4fe9d9882f11d3a99b70e1527aac979e27165f1d7
- c6c4a9725653b585a9d65fc90698d4610579b289bcfb2539f7a5f7e64e69f2e4

- a3f84aa1d15fd33506157c61368fd602d0b81f69aff6c69249bf833d217308bb
- 82c03866670b70047209c39153615512f7253f125a252fe3dcd828c6598fdf86
- 542d2267b40c160b693646bc852df34cc508281c4f6ed2693b98147dae293678

We will be using the first sample from this list as an example for our analysis.

The initial-stage loader applies Microsoft Defender exclusions to the same directory set (%TEMP%, loader path, %LocalAppData%, ...), but process exclusions are limited to the loader executable only. The Stage 2 payload is extracted from the embedded resource to %TEMP%\<...>InstallHelper.exe and launched via hidden PowerShell Start-Process. Stage 2 is protected with the same Themida + .NET Reactor packing technique.

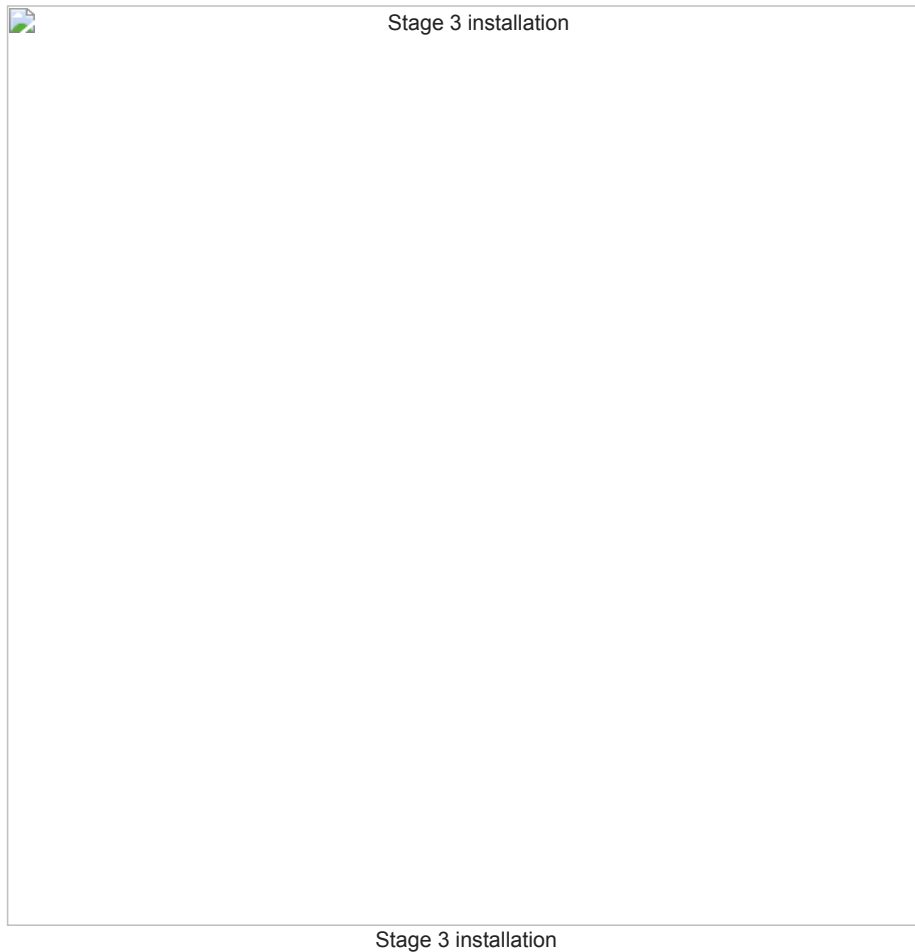
Stage 2 registers only process-level Microsoft Defender exclusions.



Setting up Microsoft Defender exclusions

Setting up Microsoft Defender exclusions

The loader then extracts four embedded resources into the install directory at `%SystemDrive%\Users\%UserName%\AppData\Local\SVCData\Config`, dropping 3 unused, benign DLLs and a malicious `svchost.exe` binary, which is the 3rd stage. Stage 3 is launched through PowerShell, and a scheduled task named `SVCConfig` is registered via `schtasks.exe` with an `ONLOGON` trigger and `HIGHEST` privilege.



Following payload launch, Stage 2 writes a temporary `.bat` file to `%TEMP%` with a polling loop that forcefully deletes the installer binary until successful, then deletes the batch file itself.



Self-delete installer binary



Self-delete installer binary

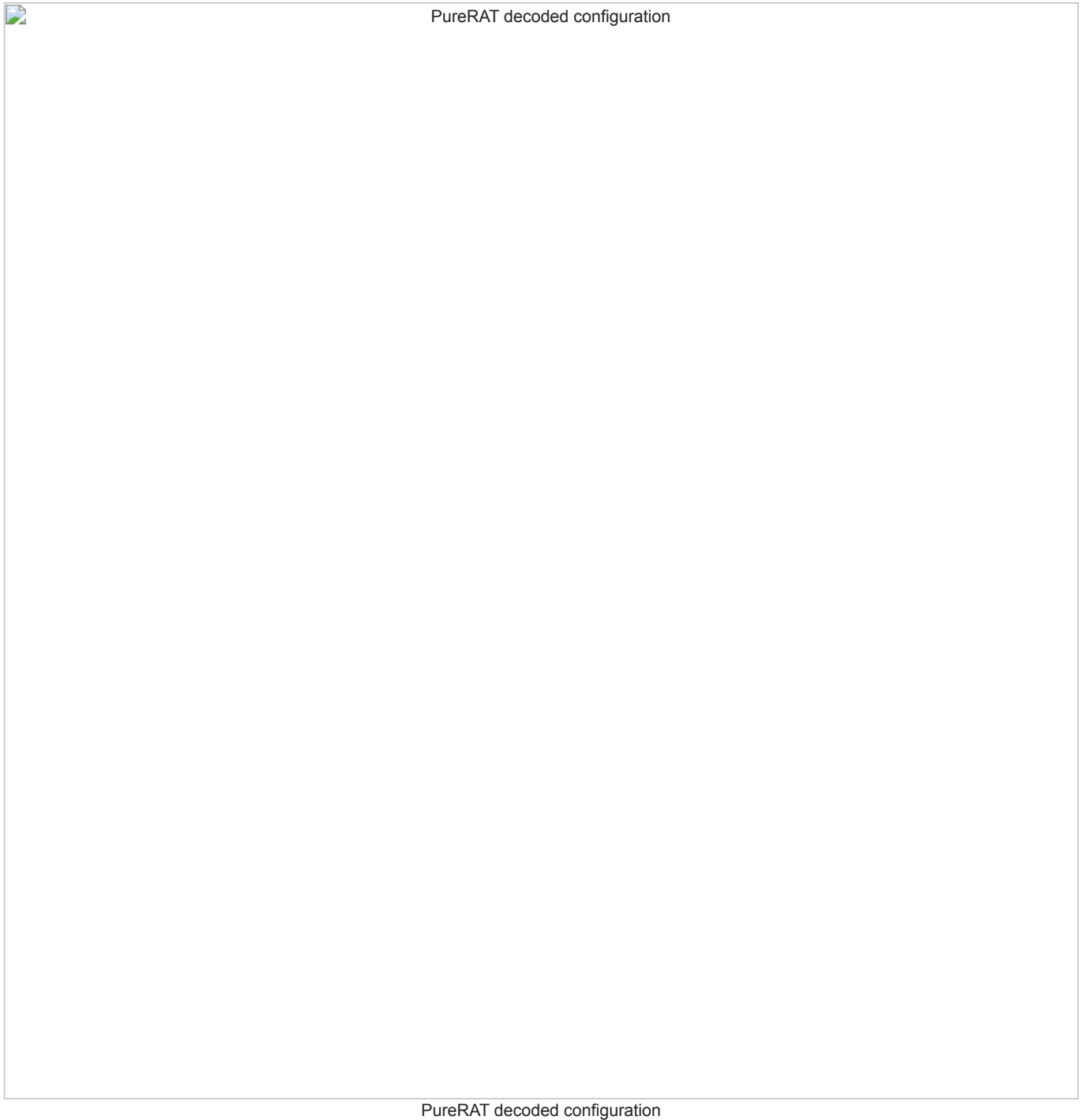
Stage 3 is a Themida + .NET Reactor-protected, in-memory PE loader, which is also the beginning of the PureRAT component. The encrypted next-stage module is stored as a .NET resource and decrypted via Triple DES (3DES) in CBC mode using an embedded key and IV. The decrypted output is a GZip-compressed PE: the first 4 bytes encode the decompressed size as a little-endian integer, followed by the GZip stream.



PureRAT next-stage decryption

The PureRAT v3.0.1 configuration is decoded by base64-decoding an embedded string and deserializing the result as a Protobuf message:

- 23-01-26 (build / campaign date)
- windirautoupdates[.]top (C2 #1)
- winautordr.itemdb[.]com (C2 #2)
- winautordr.ydns[.]eu (C2 #3)
- winautordr.kozow[.]com (C2 #4)
- Aesthetics135 (mutex and C2 comms key)



The C2 communication protocol uses key derivation function - `PBKDF2-SHA1("Aesthetics135", embedded_salt=010217EA2530863FF804, iter=5000)` to derive 96 bytes, split into an AES-256-CBC key and an HMAC-SHA256 key. Incoming messages are authenticated by verifying the HMAC over `[IV | ciphertext]` stored in the first 32 bytes; the IV is then read from byte offset (32- 48) and used to decrypt the remaining ciphertext, yielding a [Protobuf](#)-encoded command message.

By decrypting traffic captured in VirusTotal sandboxes, we observed that the C2 server at [windirautoupdates\[.\]top](#) was automatically issuing a download-and-execute task directing the implant to fetch an XMR mining payload from [https://github\[.\]com/lebnabar198/Hgh5gM99fe3dG/raw/refs/heads/main/MnrsInstllr_240126\[.\]exe](https://github[.]com/lebnabar198/Hgh5gM99fe3dG/raw/refs/heads/main/MnrsInstllr_240126[.]exe).



PureRAT initial task decryption

PureRAT initial task decryption

Campaign 3 (PureRAT, PureMiner, XMRig loader)

The third campaign variant shares the same initial-stage loader design as Campaigns 1 and 2. Its Stage 2 resembles Campaign 2 but differs by dropping multiple embedded payloads from the resource section, including PureRAT, a custom XMRig loader, and PureMiner.

Example ISO sample:

[f84b00fc75f183c571c8f49fcc1d7e0241f538025db0f2daa4e2c5b9a6739049](#).



Installation of PureRAT, PureMiner, and a custom XMRig loader

To keep the machine awake and maximize mining uptime, the loader disables sleep and hibernation via Windows power management commands:

- `powercfg /change standby-timeout-ac 0`
- `powercfg /change standby-timeout-dc 0`
- `powercfg /change hibernate-timeout-ac 0`
- `powercfg /change hibernate-timeout-dc 0`

The PureRAT configuration matches Campaign 2, differing only in the build/campaign ID: `25-11-25`.

The PE loader component of PureMiner is similar to PureRAT, and the decrypted module is also obfuscated via .NET Reactor. Since the configuration is Protobuf-serialized, hooking `ProtoBuf.Serializer::Deserialize` allows inspection of the configuration data:

- `25-11-25` (build / campaign date)
- `wndlogon.hopto[.]org` (C2 #1)
- `wndlogon.itemdb[.]com` (C2 #2)
- `wndlogon.ydns[.]eu` (C2 #3)
- `wndlogon.kozow[.]com` (C2 #4)
- `4c271ad41ea2f6a44ce8d0` (mutex and C2 comms key)



Additional behavioral indicators include the dynamic loading of AMD Display Library binaries ([atiadlxx.dll](#)/[atiadlxy.dll](#)) and the NVIDIA API library ([nvapi64.dll](#)), consistent with GPU hardware profiling techniques employed by PureMiner.



PureMiner loading atiadx.dll, atiadx.dll, and nvapi64.dll

PureMiner loading atiadx.dll, atiadx.dll, and nvapi64.dll

Custom .NET-Based Loader for XMRig

The following findings cover the custom XMRig loader deployed during this campaign. Analyzed samples:

- 0176ffaf278b9281aa207c59b858c8c0b6e38fdb13141f7ed391c9f8b2dc7630
- 9409f9c398645ddac096e3331d2782705b62e388a8ecb1c4e9d527616f0c6a9e
- f84b00fc75f183c571c8f49fcc1d7e0241f538025db0f2daa4e2c5b9a6739049

The Entry Point and Setup

Execution begins in the `Start()` method. The loader first calls `FetchRemoteConfig()`, which reaches out to a hardcoded URL (`https://autoupdatewinsystem[.]top/MyMNRconfigs/0226.txt`). The response is AES-encrypted JSON, which the loader decrypts using a hardcoded key (`AsyncPrivateInputx64`) and parses to extract the pool, wallet, and mining arguments. If the remote server is unreachable or decryption fails, it falls back to a hardcoded `ztbpVbABSx1jDIKnWGbx1d_0` configuration to ensure mining can still occur.



The hard-coded configuration when the online config is unavailable



The hard-coded configuration when the online config is unavailable

Resource Extraction

Simultaneously, an asynchronous task triggers `ExtractResources()`. The loader checks the `%TEMP%` directory for two files: `procsrv.exe` (the renamed XMRig payload) and `winRing0x64.sys` (a driver used by XMRig for direct hardware access). If either is absent, the loader unpacks them from its own assembly manifest.

Evasion Loop

After a 3-second sleep, the loader calls `StartEvasionTimer()`, initializing a timer that ticks every 1,000 milliseconds. On each tick, `IsAnalysisToolRunning()` compares all running process names against a hardcoded list of 35 security and monitoring tools (`Taskmgr`, `ProcessHacker`, `Wireshark`, `Procmon`, etc.).



Monitoring tools that are targeted



Monitoring tools that are targeted

If any analysis tool is detected, the loader immediately calls `KillMinerProcess()`, terminating `procsrv.exe`, effectively dropping the CPU usage back to normal.

If no analysis tool is detected, the loader calls `CheckAndRunMiner()`. If the miner is not currently running, it reconstructs the command-line arguments (using the remote or fallback config) and quietly launches the miner as a hidden background process via `LaunchMiner()`.

This creates a "hide and seek" scenario for the victim. Whenever they try to investigate why their PC is slow, the malware shuts down the miner.

WinRing0x64.sys and Ring 0 Access

The loader also drops and loads `WinRing0x64.sys`, a legitimate open-source driver frequently abused by cryptominers. The driver provides direct Ring 0 (kernel-level) hardware access, which XMRig uses to apply its Model Specific Register (MSR) modification, reconfiguring CPU prefetcher and L3 cache behavior to significantly boost RandomX (Monero) hash rates.

Campaign 4 - Umr_ (SilentCryptoMiner)

From the `autoupdatewinsystem[.]top` domain, we identified another GitHub account <https://github.com/ugurlutaha6116> hosting another loader variant whose executable name is prefixed with `Umr_`. This loader is a Themida-packed SilentCryptoMiner loader that installs persistently on the victim machine, injects a watchdog payload into `conhost.exe`, and a miner payload into `explorer.exe`, mining ETH or XMR depending on the build configuration.

SilentCryptoMiner is a closed-source Win32 64-bit malware released for free on [GitHub](#). The samples we analyzed are older versions than the latest [release](#):

- `1f7441d72eff2e9403be1d9ce0bb07792793b2cb963f2601ecfdf8c91cd9af73`
- `468441d32f62520020d57ff1f24bb08af1bc10e9b4d4da1b937450f44e80a9be`
- `4e6b8fdd819293ca3fe8f8add6937bf6531a936955d9ac974a6b231823c7330e`
- `6492e50e79b979254314988228a513d5acbd9a950346414955dc052ae77d2988`
- `ce90cb3a9bfb8a276cb50462be932e063ed408af8c5591dd2c50f1c6d18c394c`

Direct Syscalls

To evade detection, SilentCryptoMiner uses direct syscalls instead of `NTDLL` functions. To do this, it parses `NTDLL` exports to locate the target function by a hash of its name, extracts the syscall number, and manually executes the syscall instruction sequence.



Direct syscall procedure

Direct syscall procedure

Disable Sleep and Hibernate

To ensure it can use the host machine for as long as possible, SilentCryptoMiner disables Windows sleep and hibernation by executing a shell command.



Disable windows sleep and hibernate

Disable windows sleep and hibernate

Install Persistence

After copying itself to its installation folder (in this case, configured to masquerade as legitimate software named “Appdata/Local/OptimizeMS/optims.exe”), SilentCryptoMiner proceeds to establish persistence. If the process is running with administrator privileges, it creates a scheduled task configured via an XML file.



Schtask task creation for persistence

Schtask task creation for persistence

The XML file is dropped onto the disk in the `AppData/Local/Temp` folder and contains the task configuration. One interesting setting is `AllowHardTerminate = False`, which prevents the task from being forcibly terminated via `schtasks`.



Malware XML task configuration



Malware XML task configuration

If the process lacks administrator rights, it instead adds a **Run** key to the registry.



Malware adds a run key for persistence if not running as administrator

Malware adds a run key for persistence if not running as administrator

After initial installation, the process terminates. On subsequent execution by the persistence mechanism, it verifies that it is running from its installation directory before proceeding to the process injection phase.

Inject watchdog and miner payloads

In the samples we analyzed, the builds contain four payloads:

- A **Winring0.sys** driver
- A watchdog process
- A Monero miner

- An Ethereum miner

We know that the malware can contain multiple miners; however, in our tests, we only observed the Monero miner injected into a process. In the code, only one of the two miners is injected, which we assume depends on the configuration.

SilentCryptoMiner initiates injection by creating a new suspended process with a spoofed parent process. It obtains a handle to `explorer.exe` using `NtQuerySystemInformation` and `NtOpenProcess`, then configures a `PS_ATTRIBUTE_LIST` structure with the handle for parent spoofing and passes it to `NtCreateUserProcess`.

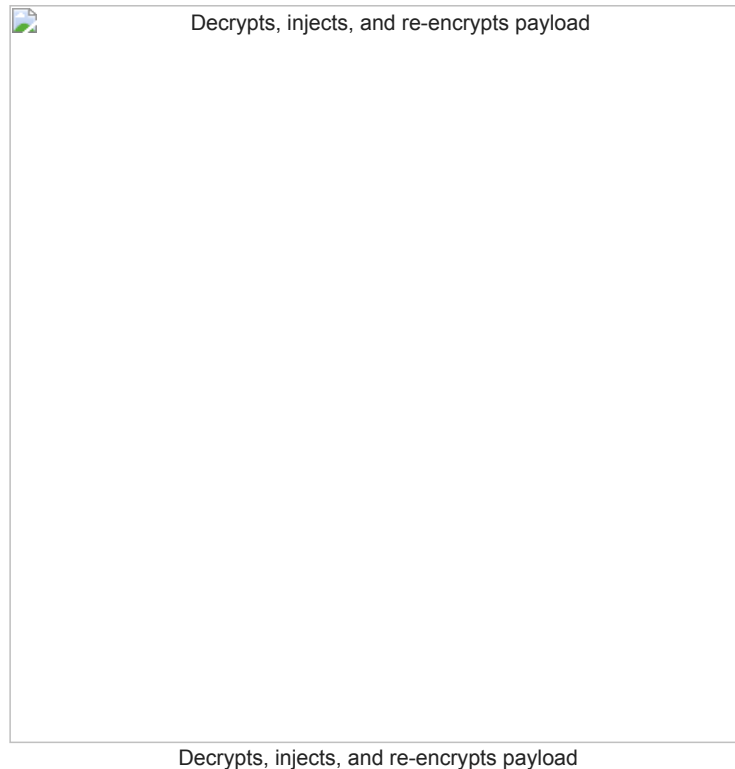
The payload is written to disk via `NtCreateFile` and `NtWriteFile`, then mapped into the target process's memory space through `NtCreateSection` and `NtMapViewOfSection`. Execution flow is hijacked by modifying the suspended process's entry point (in the `RCX` register) to point to the payload's image base using `NtGetContextThread` and `NtSetContextThread`. The process's PEB (in `RDX` register) image base is also set to the payload's address using `NtWriteVirtualMemory`. Finally, the process is resumed with `NtResumeThread`.



Process injection procedure

Process injection procedure

The payload data is decrypted from a hardcoded blob in the binary using a simple XOR cipher with a hardcoded key. After injection, the blob is re-encrypted in memory to reduce forensic traces.



In the analyzed samples, SilentCryptoMiner utilizes two distinct processes for payload injection: the watchdog component is injected into `conhost.exe`, while the miner payload targets `explorer.exe`. The `WinRing0.sys` driver is also written to disk, then loaded and used by the miner. This is likely to optimize the CPU for mining operations.

Watchdog and Miner Processes

The watchdog is responsible for monitoring the loader file in its persistence folder: it rewrites the file to disk if it is deleted and reinstalls the persistence mechanism if the scheduled task or registry key is deleted.

The miner downloads its configuration from `(/UWP1)?/*CPU.txt` endpoints and communicates with its C2 via `[UWP1|UnamWebPanel7]/api/endpoint.php` API, depending on the version.

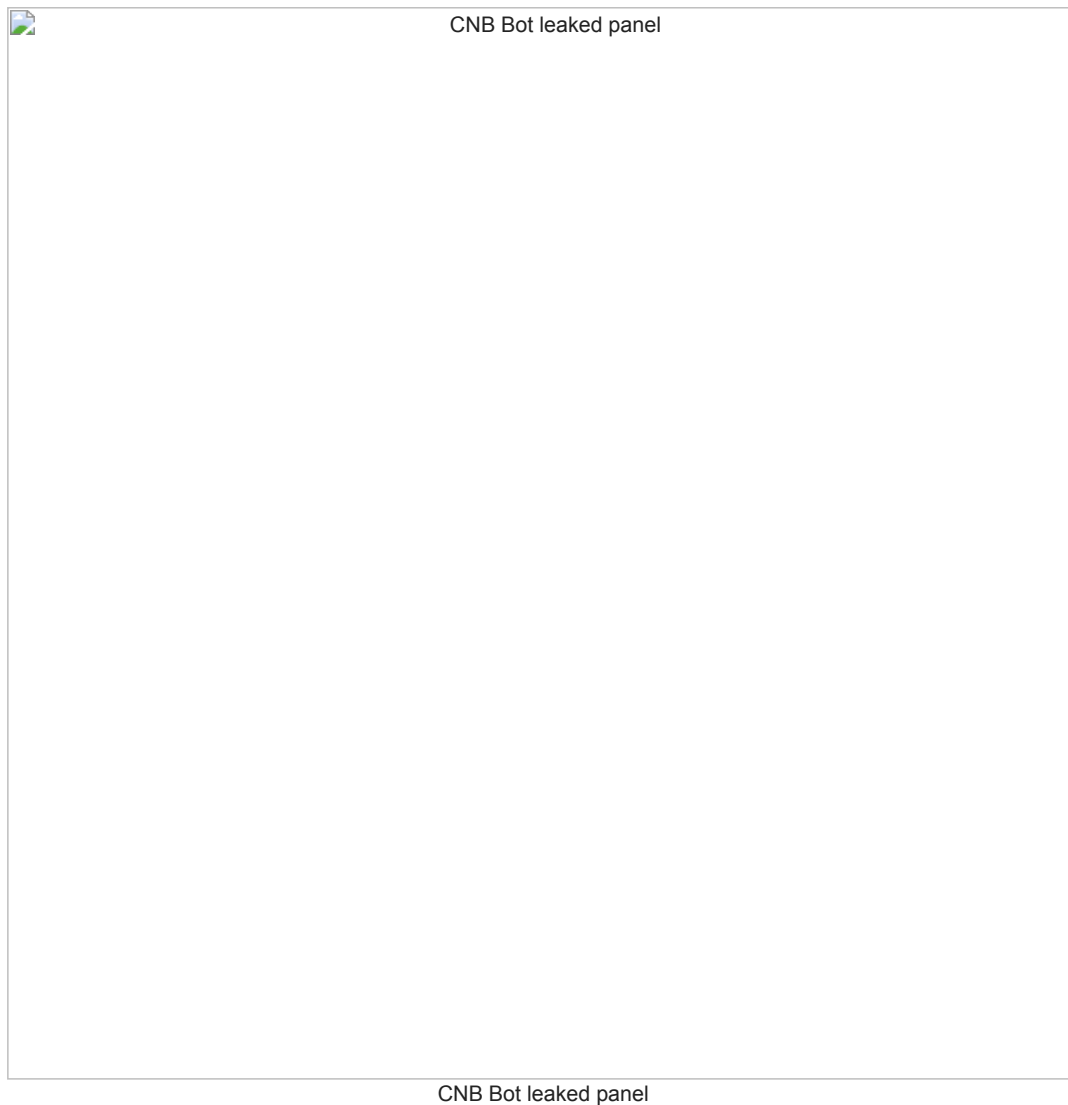
Based on the documentation and memory strings, we know that the miner includes supplementary protection measures: Like the .NET miner detailed previously, it halts mining operations when it detects specific blocklisted processes. These processes encompass a variety of tools, including those used for process monitoring, network monitoring, antivirus protection, and reverse engineering.

Code analysis - CNB Bot

CNB Bot is a .NET implant with integrated loader capabilities. It implements a command-polling loop against its configured C2 servers, and supports 3 operator commands:

- download-and-execute arbitrary payloads
- self-update
- uninstall/cleanup

On Jan 31, 2026, malware researcher [@ViriBack](#) discovered a related C2 panel that was exposed at [https://win64autoupdates\[.\]top/CNB/10g1n234\[.\]php](https://win64autoupdates[.]top/CNB/10g1n234[.]php), which has since been taken offline.



Configuration

Some configuration values for CNB Bot are not encrypted, such as the bot version (**1.1.6.**), campaign date (**03_26**), and the scheduled task name for persistence (**HostDataPlugin**).



Bot version and campaign ID in plaintext

Bot version and campaign ID in plaintext

Sensitive strings (C2 URLs, mutex name, auth token, comms key) are stored AES-256-CBC encrypted with a hardcoded 32-byte key, which differs across campaign batches.

Strings can be decrypted through the following formula:

```
x = base64.decode(data)
decrypted = AES256CBC(key=hard_coded_key, iv=x[0:16]).decrypt(x[16:])
```

Extracted configuration:

Field	Value
Mutex Name	MTXCNBV11000ERCXSWOLZNBVRGH
C2 URL	https://tabbysbakescodes[.]ws/CNB/gate.php
C2 URL fallback #1	https://tommysbakescodes[.]ws/CNB/gate.php
C2 URL fallback #2	https://tommysbakescodes[.]cv/CNB/gate.php
Auth Token	0326GJSECMHSHOEYHQMKDZ
Comms AES Key (input)	AnCnDai@4zDsXP!a3E
Scheduled Task	HostDataProcess
Install Dir	%APPDATA%\HostData\
Marker File	%APPDATA%\HostData\install.dat
Executable	sysdata.exe
Group / Campaign	03_26
Bot Version	1.1.6.

Execution Flow

At startup, CNB Bot uses five different methods to check for VM detection:

Check	Technique
WMI ComputerSystem	Manufacturer/Model: "vmware", "virtualbox", "vbox", "qemu", "xen", "parallels", "innotek", "microsoft corporation" (manufacturer) + "virtual machine" (model)
WMI BIOS	Version/Serial: "vmware", "virtualbox", "vbox", "qemu", "bochs", "seabios"
Process list	"vmtoolsd", "vmwaretray", "vmwareuser", "vboxservice", "vboxtray", "xenservice"
Registry	VMware Tools / VirtualBox Guest Additions keys: "SOFTWARE\VMware, Inc.\VMware Tools", "SOFTWARE\Oracle\VirtualBox Guest Additions", "SYSTEM\CurrentControlSet\Services\VBBoxGuest", "SYSTEM\CurrentControlSet\Services\VBBoxSF"
MAC Address	"00:0C:29", "00:50:56", "00:05:69", "08:00:27", "0A:00:27", "00:16:3E", "00:1C:14"

Each check returns zero or one and is summed against a threshold. When the detection threshold is reached, the first process instance acquires a named mutex and enters an infinite sleep (`Thread.Sleep(int.MaxValue)`), appearing hung rather than terminating cleanly. Any subsequent instance finding the mutex already held exits immediately.

Otherwise, on first execution, the implant checks for `%APPDATA%\HostData\install.dat`. If absent, it performs the initial installation:

- Generates a random 5-character alphabetic subdirectory name under `%APPDATA%\HostData\`
- Copies itself to `%APPDATA%\HostData\<random>\sysdata.exe`
- Writes the installed path to `install.dat`
- Extracts benign dependencies `DiagSvc.dll` and `sdrsvc.dll` into the same directory
- Writes a VBScript wrapper `sysdata.vbs` alongside the binary: `CreateObject("WScript.Shell").Run "<installed_path>""", 0, False`

- Creates a scheduled task named `HostDataProcess` via `schtasks.exe`, configured to run `wscript.exe //nologo sysdata.vbs` every 10 minutes at `HIGHEST` privilege
- Launches the installed copy as a hidden process with `%TEMP%` as the working directory
- Self-deletes the original copy via a self-deleting BAT script (`timeout /t 3, loop-del`)

On subsequent runs, when `install.dat` exists, and the running path matches its contents, the implant proceeds to active operation:

- Sets the current working directory to `%TEMP%`
- Repairs persistence: checks if `sysdata.vbs` exists (recreates if absent) and verifies the scheduled task is configured with `wscript.exe`, re-registering it if necessary
- Acquires a named mutex (`MTXCNBV11000ERCXSWOLZNBVRGH`) - exits if already running
- Instantiates the victim profiler, C2 comms, and command dispatcher
- Issues a single POST to the C2 with `payload: "fetch"`, handles any returned task
- Exits - next execution is driven entirely by the 10-minute scheduled task trigger



CNB Bot main code logic

CNB Bot main code logic

C2 Communication

The malware communicates with its C2 by issuing HTTP POST requests with the Content-Type set to `application/x-www-form-urlencoded`. Each field value is independently AES-256-CBC encrypted with a random IV. The AES key is derived as the SHA-256 hash of the hardcoded communications passphrase (`AnCnDai@4zDsXP!a3E`). The IV is prepended to the ciphertext, and the entire blob is base64-encoded; C2 responses follow the same format.

```
encrypted_field_value = base64_encode(random_iv + AES-256-CBC_encrypt\  
(key: SHA-256('AnCnDai@4zDsXP!a3E'), iv: random_iv, data: plaintext_field_value))
```

Fields sent on every request:

Field	Value
<code>desktop</code>	machine name
<code>username</code>	username
<code>os</code>	Windows version
<code>version</code>	bot version (<code>1.1.6.</code>)
<code>privileges</code>	user OR admin
<code>cpu</code>	processor name from the registry
<code>gpu</code>	GPU name(s) from registry
<code>gpu_type</code>	yes (discrete) / no (integrated)
<code>group</code>	group / campaign ID (<code>03_26</code>)
<code>client_path</code>	full path of running executable
<code>local_ipv4</code>	external IP via <code>ipify[.]org</code> / <code>icanhazip[.]com</code> / <code>ident[.]me</code>
<code>auth_token</code>	authentication token (<code>0326GJSECMHSHOEYHQMKDZ</code>)
<code>timestamp</code>	Unix epoch (UTC)
<code>payload</code>	Command string (“fetch”, “completed”)

A server response decrypts to either a task string, `“NO TASKS”`, or `“REGISTERED/UPDATED”`. When the client requests a task through `payload: “fetch”`, if a task exists for the client, the C2 response decrypts to a `<sep>`-delimited task string: `task_id<sep>command<sep>argument<sep>RSA_sig`.



CNB Bot dispatcher function

CNB Bot dispatcher function

Prior to dispatch, each task undergoes RSA-SHA256 signature verification. The signed message is the concatenated string `task_id<sep>command<sep>argument`, and the signature is the base64-decoded `RSA_sig` field. A hardcoded RSA-2048 public key is used for verification.



Tasks failing verification are silently dropped. Without the operator's RSA private key, third parties cannot issue commands to infected hosts even with full C2 access.

Supported Commands

3 commands are supported, described in the table below:

Command	Behavior
download_execute	Downloads URL argument to %TEMP%\<random>.<ext>. Execute: .exe (hidden), .bat/.cmd (cmd /c), .vbs (wscript.exe), other (ShellExecute).
update	Downloads URL argument to staging location %TEMP%\tmp_updt236974520367.exe. Runs BAT to: kill current PID, overwrite installed binary with staged download, delete staging file, and self-delete BAT.
uninstall	Deletes scheduled task, removes install.dat, self-deletes via BAT, rmdir install dir, and %APPDATA%\HostData\.

Earlier Campaigns

Pivoting on the PureRAT mutex `Aesthetics135`, we discovered an earlier wave of the operation that presented a different fake installer UI.



Fake installer interface from early 2025

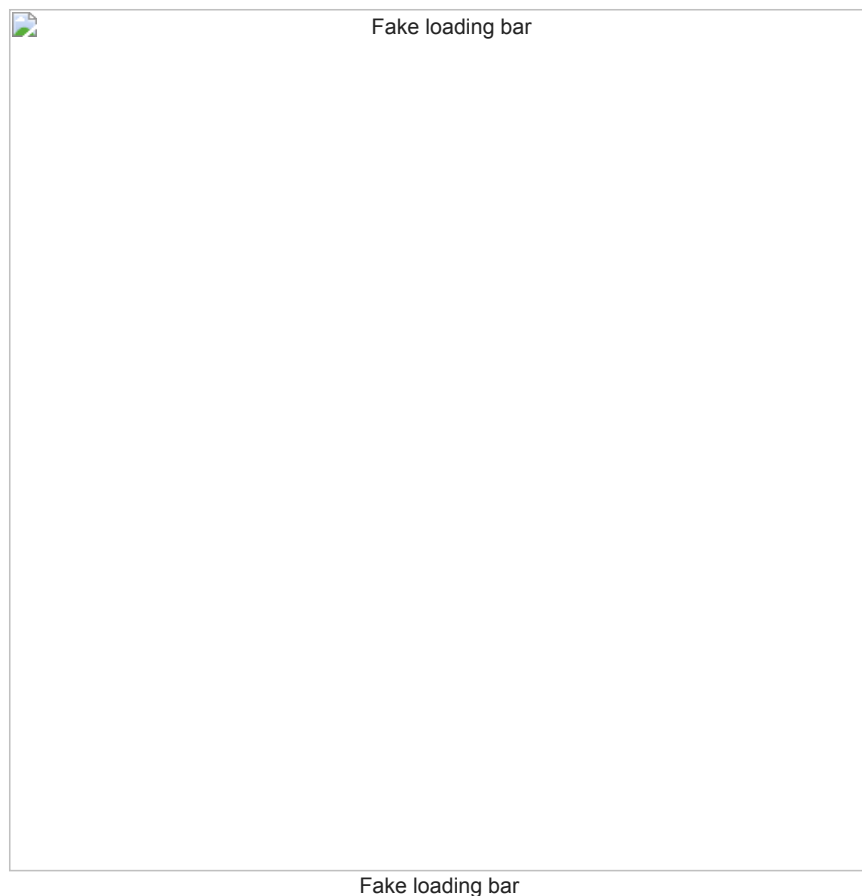
Early 2025 Build

The sample `bb48a52bae2ee8b98ee1888b3e7d05539c85b24548dd4c6acc08fbe5f0d7631a` (first seen 2025-01-30) is a Themida and .NET Reactor-protected Windows Forms application that drops PureRAT v0.3.9.

It consists of 3 classes: `Fooo1rm` (the ApplicationContext entry point), `Form2` (the installer UI and the PureRAT dropper), and `Form3` (a fake registration lure). The code structure closely resembles the more recent campaigns.

On initialization, it immediately invokes a hidden PowerShell one-liner to add itself to Microsoft Defender exclusions before any UI appears: `powershell.exe -WindowStyle Hidden Add-MpPreference -ExclusionPath '<self_path>'; Add-MpPreference -ExclusionProcess '<self_path>'`. A timer with a 2,846 ms interval fires, instantiating and showing `Form2`.

`Form2` presents a progress bar dialog titled “Getting things ready” with a 12-step timer ticking every 1,000 ms, simulating a legitimate installation.



A second PowerShell exclusion command covers `%LocalAppData%`, `%AppData%`, the drop directory `%LocalAppData%\winbuf`, and process names including `winbuf.exe`, `wintrs.exe`, and `AddInProcess.exe`. The PureRAT v0.3.9 payload is extracted from the assembly manifest resource and written to `%LocalAppData%\winbuf\winbuf.exe`. Persistence is established via `schtasks.exe`.

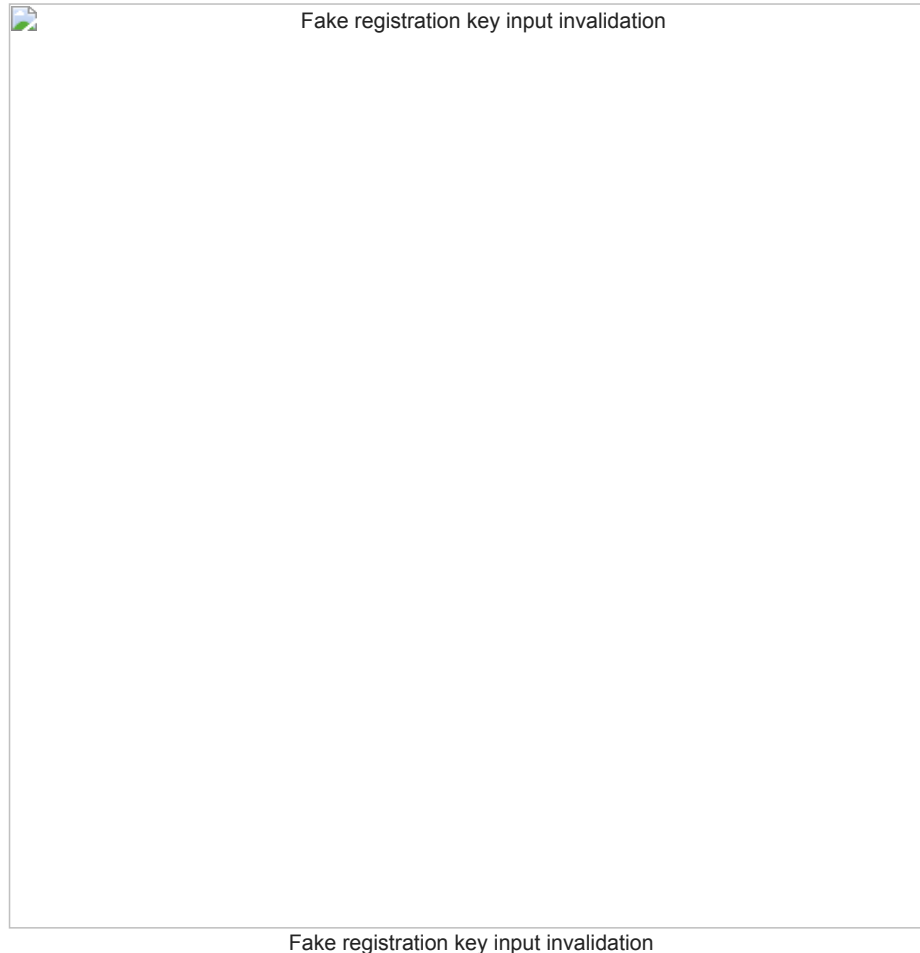
Extracted PureRAT config:

- `wndlogon.hopto.org` (C2 #1)
- `wndlogon.itemdb.com` (C2 #2)
- `wndlogon.kozow.com` (C2 #3)
- `wndlogon.ydns.eu` (C2 #4)
- `Aesthetics135` (mutex and C2 comms key)
- `29-01-25` (build / campaign date)

`Form3` serves purely as a social engineering mechanism to drive [Cost Per Action](#) (CPA) offer completions through a content locker.

Content lockers are a monetization technique in which access to a resource is gated behind completing CPA (Cost Per Action) offers, such as filling out a survey or signing up for a service. The malware operator earns a commission each time a victim completes one of these offers.

It presents a fake "Registration Required" dialog with a key entry field, a "Validate" button, and a hyperlink labeled "here" that opens [https://tinyurl\[.\]com/cmvt944y](https://tinyurl[.]com/cmvt944y). Key validation is entirely fake. Regardless of input, the handler introduces a hardcoded 2-second delay, then always returns "Invalid key. Please try again."



The TinyURL shortlink [tinyurl\[.\]com/cmvt944y](https://tinyurl[.]com/cmvt944y) redirects to the lure page at [rapidfilesdatabase\[.\]top/files/z872d515ea17b4e6c3abca9752c706242/](https://rapidfilesdatabase[.]top/files/z872d515ea17b4e6c3abca9752c706242/).

The page used to host a minimal HTML document titled "Registration Key is Ready", designed to trick the victim into interacting with the CPA content locker. It presents a download icon and a fake file link labeled [Registration_Key.txt](#), alongside a unique campaign tracking ID ([z872d515ea17b4e6c3abca9752c706242](#)) displayed in the page body.

The content locker JavaScript ([3193171.js](#)) is loaded from [d3nxbjuv18k2dn.cloudfront\[.\]net](#), and clicking the [Registration_Key.txt](#) link triggers the offer wall under the pretext of unlocking a license key.



Content at rapidfilesdatabase[.]top/files/z872d515ea17b4e6c3abca9752c706242/

Content at rapidfilesdatabase[.]top/files/z872d515ea17b4e6c3abca9752c706242/



CPA content locker JS (3193171.js)

CPA content locker JS (3193171.js)

Late 2023 Build

An older sample - [6a01cc61f367d3bae34439f94ff3599fccb66d05a8e000760626abb9886beac](#) (first seen 2023-11-09) presented a similar fake installer UI. This represents the earliest activity we attributed to this threat actor based on shared infrastructure and tooling.



This campaign build dropped PureRAT v0.3.8B, in which the in-memory PE loader component used a SmartAssembly-protected PureCrypter.

Extracted PureRAT config:

- wndlogon.hopto.org (C2 #1)
- wndlogon.itemdb.com (C2 #2)
- wndlogon.kozow.com (C2 #3)
- wndlogon.ydns.eu (C2 #4)
- [Aesthetics135](#) (mutex and C2 comms key)
- [09.11.23](#) (build / campaign date)

On the installation window, the “go here” hyperlink opens a short link <https://t.ly/MQXPm> that redirects to the lure page [https://softwaredlfast\[.\]top/files/n71fGbs2b7XceW3op71aQsrx41Rkeyd1/](https://softwaredlfast[.]top/files/n71fGbs2b7XceW3op71aQsrx41Rkeyd1/), which presents 2 outgoing fake download links:

- [https://rapidfilesbaze\[.\]top/z78fGbs2b7XceWop21aQsrx41Rkeydsktp/](https://rapidfilesbaze[.]top/z78fGbs2b7XceWop21aQsrx41Rkeydsktp/)
- [https://rapidfilesbaze\[.\]top/z78fGbs2b7XceWop21aQsrx41Rkeymb1/](https://rapidfilesbaze[.]top/z78fGbs2b7XceWop21aQsrx41Rkeymb1/)



Content at [https://softwaredlfast\[.\]top/files/n71fGbs2b7XceW3op71aQsrx41Rkeyd/](https://softwaredlfast[.]top/files/n71fGbs2b7XceW3op71aQsrx41Rkeyd/)



Content at [https://softwarefast\[.\]top/files/n71fGbs2b7XceW3op71aQsrx41Rkeydl/](https://softwarefast[.]top/files/n71fGbs2b7XceW3op71aQsrx41Rkeydl/)

Both links were offline at the time of analysis. However, historical data indicates that [rapidfilesbaze\[.\]top](#) has been used consistently for CPA-style offer lures.

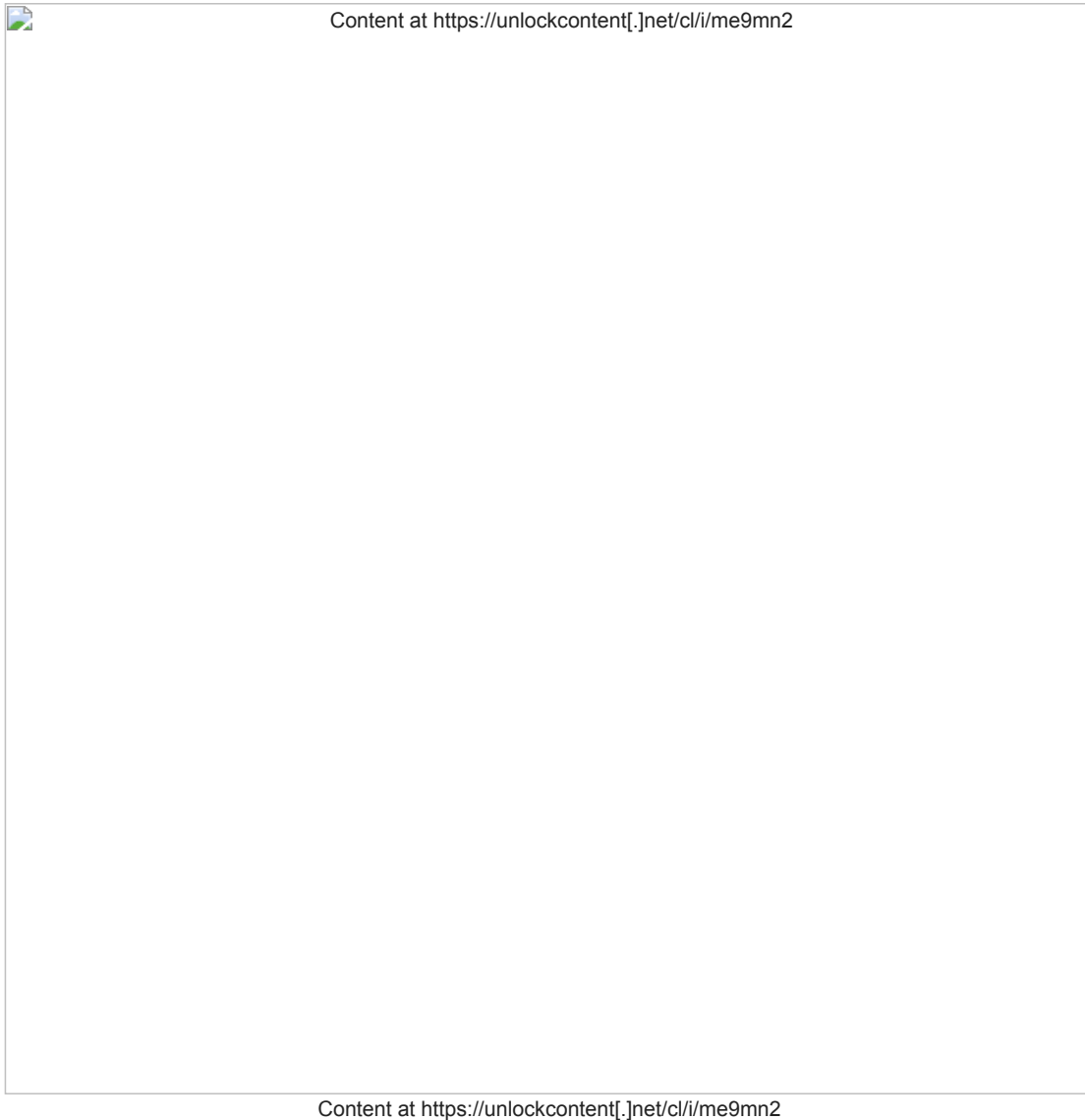
A [URLScan.io](#) archived response for a related path ([rapidfilesbaze\[.\]top/h74fGbs2b7XceWop71aQsrx41-Registration-Key-Mobile/](#)) confirms the site's use as a lure landing page.



Content at rapidfilesbaze[.]top/h74fGbs2b7XceWop71aQsrx41-Registration-Key-Mobile/

Content at rapidfilesbaze[.]top/h74fGbs2b7XceWop71aQsrx41-Registration-Key-Mobile/

The downstream unlocker site at [https://unlockcontent\[.\]net/c1/i/me9mn2](https://unlockcontent[.]net/c1/i/me9mn2) remains active as of this writing.



GitHub Profiles

Beyond the C2 infrastructure, the threat actor abuses GitHub as a payload delivery CDN, hosting staged binaries across two identified accounts. This technique shifts the download-and-execute step away from operator-controlled infrastructure to a trusted platform, reducing detection friction. Both profiles were confirmed through decrypting C2 task traffic captured by VirusTotal sandboxes, which issued download-and-execute tasks pointing directly to raw GitHub content URLs. The operator routinely deletes individual binaries and entire repositories; the files documented below were captured via VirusTotal submissions or direct retrieval from GitHub prior to deletion.

The first profile, [https://github\[.\]com/lebnabar198](https://github[.]com/lebnabar198), surfaced during analysis of Campaign 2. After decrypting the C2 traffic from the [windirautoupdates\[.\]top](#) server, we observed the PureRAT implant being instructed to fetch a payload from this account, specifically the custom XMRig loader [MnrsInst11r_240126.exe](#). This establishes a direct operational link between the PureRAT C2 and this GitHub profile.

The second profile, [https://github\[.\]com/ugurlutaha6116](https://github[.]com/ugurlutaha6116), was identified by decrypting traffic from a PureRAT loader (SHA-256: [e1e87d11079d33ec1a1c25629cbb747e56fe17071bde5fd8c982461b5baa80a4](#)), which used the same PBKDF2 key derivation structure with the comms key [Aesthetics152](#). The decrypted task pointed to the hosted payload [PM3107.exe](#).

The hosted files map to the following payloads:

Filename	Associated payload
CNB-v112-zUpdt-inPmnr.exe	CNB Bot
MyXMRmnr_Instllr_0302.exe	Custom XMRig loader
MnrsInstllr_240126.exe, MnrsInstllr_030126.exe	Custom XMRig loader
PM2311.exe, PM1109.exe, ...	PureMiner
Pmnr_1303_wALL.exe, Pmnr_Instllr_1303.exe, ...	PureMiner
A_Instllr_250525.exe	AsyncRAT
U_n_P_Installer_220725.exe, U_n_P_Installer_110725.exe, ...	Loader for SilentCryptoMiner & PureMiner
umnr_120525.exe, Umnr_1403_frPmnr.exe, ...	SilentCryptoMiner
plsr_instllr_1804.exe	Pulsar RAT

Monero Wallet Analysis

During our analysis of the cryptominer payloads, we successfully extracted four active Monero (XMR) wallet addresses from the malware's configuration. Because the threat actor is routing their compromised hosts through public mining pools, we can query the pool's public dashboards using these wallet addresses. It provides information about the operational scale and profitability of the campaigns.



Tracking mining activity through a public dashboard



Tracking mining activity through a public dashboard

Based on the telemetry available at the time of writing, here is the current status of the attacker's mining operations:

- **Wallet 1:** `87NnUp8GKVBZ8pFV75Gas4A5nMMH7gEeo8AXBhm9Q6vS5oQ6SZCYf1bJr7Lib35VN2UX271PAXeqRFDmjo5Sxm3zFDfDSWD`
 - **Active Workers:** 7
 - **Estimated Hashrate Return:** ~0.0172 XMR / day
 - **Total Paid Out:** 2.2 XMR
- **Wallet 2:** `89FYoLrFXwEDAVAsVYbhAfg3mATUtBzNAK2LG8wwDKfNTRhmNRTBn1VbwpFxEpJ8h5fQa2A4CS1tpRv7amUdJ3ZbUoVu6T1`
 - **Active Workers:** 3
 - **Estimated Hashrate Return:** ~0.02 XMR / day
 - **Total Paid Out:** 4.23 XMR
- **Wallet 3:** `89WoZKYohhcNEFRV8jjB6nDqzjiBtQqyp4agGfyHwED1XyVAoknfVsVY1CwEHG6nwZFJGFTF5XbqC4tAQbnoFFCX8UQof3G`
 - **Active Workers:** 2
 - **Estimated Hashrate Return:** ~0.0057 XMR / day
 - **Total Paid Out:** 11.69 XMR
- **Wallet 4:** `83Q1PKZ5yXsP8SCqjV3aV7B3UoBB3skPp49G1VnnGtv5Y5EubFQTXvzR9cZshBYBBfd8Dm1snkkud431pdzEZ2uJTad1CiC`
 - **Active Workers:** 2
 - **Estimated Hashrate Return:** ~0.0036 XMR / day
 - **Total Paid Out:** 9.76 XMR

With a combined total of over 27.88 XMR (~ USD\$ 9392) already successfully paid out to the attacker, it proves that low-and-slow cryptojacking operations can yield consistent financial returns over time.

Agentic Payload and Configuration Extraction Pipeline

In this research, we examined several hundred infection chains across the campaigns we described. For each chain, we have samples, mainly .NET, which are either loaders or final payloads layered with .NET Reactor obfuscation and often Themida packing.

The large number of these chains makes manual configuration and unpacking time-consuming and difficult to scale across all the chains we discovered. This is why, as part of this research, we used the Claude Opus 4.5 model to quickly vibecode a payload and configuration extraction pipeline. In this section, we provide details on the choices we made and the results we obtained with this method.

Triage

To optimize processing time, this phase focuses on extensively exploring infection chains using VirusTotal. We begin by obtaining a list of hashes from VirusTotal based on a specific pivot. For instance, using the README.txt content as a pivot to identify other ISOs.





VirusTotal ISO pivot

Claude is instructed to use a Python script to perform a recursive download. This process involves gathering information about embedded binaries and dropped files associated with each file hash. Claude then uses its “intelligence” to identify the most subsequent link in the chain and continues its investigation until it reaches what it considers the final binary in that chain. After exploring all chains, Claude analyzes the patterns and creates chain types to group them. Finally, the results are compiled into a CSV file for subsequent analysis.

The data we obtained includes the starting hash from VirusTotal and the final hash, representing the last file Claude successfully tracked. This demonstrates that, with the right guidance, Claude can effectively track entire chains using only information from VirusTotal.



Triaged data



Triaged data

Download and Extraction

Once the triage file was created, we downloaded the intermediate payloads and instructed Claude to start the automatic payload/configuration extraction process. To do this, we installed an OpenSSH server on a Windows virtual machine, then created a Claude skill containing instructions to connect to this machine and use the installed tools to perform the reverse engineering and extraction workflow.

The workflow is simple: Claude connects to the machine, uploads the sample, detects whether it is obfuscated or packed with Detect It Easy, and applies the appropriate deobfuscation tool until the sample is no longer obfuscated (Unlicense, .NET Reactor Slayer). It then runs the developed extraction scripts to identify what the sample is and determine the next step: either continue extraction with the child payload if the parent is a loader, or store the configuration information for the final report.



If all the extraction scripts fail, Claude must enter Research Mode. This mode is the most enjoyable part of the skill because it gives Claude a workflow to either automatically develop a new extraction script or identify why the existing script doesn't work with the variant. Claude's Research Mode consists of using the [dnSpyEx](#) tool installed on the machine to compile the sample's C# code, perform a complete code analysis, identify how to extract the payload or configuration, then develop a script with this knowledge to work directly with the raw binaries to be more efficient and finally store the knowledge for the next time it has to work on the same malware family.



Research mode instruction

Research mode instruction

Results

Using the Claude Opus 4.5 model, the results were really good. Not only did Claude succeed in handling the obfuscation layers, but it also completely researched and developed, on its own, the methods and scripts (based on the CIL of .NET binaries) to extract the final payloads and their configurations without having encountered them before.

It also demonstrated robust failure handling without requiring additional instruction. For example, when it encountered samples that could not be fully deobfuscated due to issues with Reactor Slayer, which made static extraction too difficult, it stopped processing, documented the problem, and proceeded to the next sample.



Claude entering Research Mode on extraction failure

Claude entering Research Mode on extraction failure

Of course, it is not without drawbacks:

- Once its context started to fill up too much, it often diverged onto useless paths and required either micro-management or a reset, hence the usefulness of having a skill with reusable instructions and a knowledge base on the work already done.
- It takes a long time, every action requires it to “think”, however, it’s automatic and it’s definitely time recovered you can use to do something else.
- Its token consumption is particularly greedy, especially once you figure out it’s doing a lot of inefficient things.

Observations

The following tables consolidate malware configurations extracted across the builds we investigated, and are not exhaustive:

CNB Bot

Versions	1.1.1., 1.1.2., 1.1.3., 1.1.5., 1.1.6.
C2s:	tabbysbakescodes[.]ws/CNB/gate.php tommysbakescodes[.]ws/CNB/gate.php tommysbakescodes[.]cv/CNB/gate.php win64autoupdates[.]top/CNB/gate.php autoupdatewinsystem[.]top/CNB/gate.php
Campaign/Build ID	03_26, 25_02_26, 15_02_26, 1502_26, 0502_26, 01-26, frPmnr_0126
Auth tokens	0326GJSECMHSHOEYHQMKDZ 020226SNDLPXSHTCSURVQ 0226frBLKWNYHD0FS1YWE 0126HRAOLQEFNGGRCXMITREQC
Mutex	MTXCNBV11000ERCXSWOLZNBVRGH

PureRAT

Versions	0.3.8B. 0.3.9, 0.4.1, 3.0.1
C2s	windiraupdates[.]top winautordr.hopto[.]org winautordr.itemdb[.]com winautordr.ydns[.]eu winautordr.kozow[.]com wndlogon.hopto[.]org wndlogon.itemdb[.]com wndlogon.kozow[.]com wndlogon.ydns[.]eu
Campaign/Build IDs	23-01-26, 14-01-26, 03-01-26, 24-12-25, 25-11-25, 08-11-25, 29-01-25, 09.11.23
Mutex / C2 Comms key	Aesthetics135

PureMiner

Versions	7.0.6, 7.0.7
C2s	wndlogon.hopto[.]org wndlogon.itemdb[.]com wndlogon.ydns[.]eu wndlogon.kozow[.]com
Campaign/Build IDs	24-10-25, 23-11-25, 15-09-25-MassUpdt, 11-09-25, 08-08-RAM, 06-08-RAM, 04-08-RAM, 31-07-RAM, 03-08-RAM, 13-03-25, 25-07-RAMWALL, 25-11-25
Wallet Address	89WoZKY0HhcNEFRV8jjB6nDqzjiBtQqyp4agGfyHwED1XyVAoknfVsVY1CwEHG6nwZFJGFTF5XbqC4tAQbnoFFCX8UQof3G
Mutex / C2 Comms key	4c271ad41ea2f6a44ce8d0

Custom XMRig Loader

Wallet Address	87NnUp8GKVBZ8pFV75Gas4A5nMMH7gEeo8AXBhm9Q6vS5oQ6SzCYf1bJr7Lib35VN2UX271PAXeqRFDmjo5SXm3zFDfDSWD,83sDbPzoghAX45hA2Y26xvaDsKv8TLymAGKKyZwrCKB3T9kuuYBDzb64vfy9XQyrpUFQ4r8u3V2T1EzqE6CR27XmMCCwGu1
-----------------------	---

AsyncRAT

Versions	0.5.8
C2s	wndlogon.hopto[.]org wndlogon.itemdb[.]com wndlogon.ydns[.]eu wndlogon.kozow[.]com
Campaign/Build IDs	BL_Bckp_250525

PulsarRAT

Versions	1.5.1
C2s	wndlogon.hopto[.]org wndlogon.itemdb[.]com wndlogon.ydns[.]eu wndlogon.kozow[.]com
Campaign/Build IDs	18-04-25

SilentCryptoMiner

Mining Pool	gulf.moneroocean[.]stream:10128
Wallet	83Q1PKZ5yXsP8SCqjV3aV7B3UoBB3skPp49G1VnnGtv5Y5EUbFQTxvzR9cZshBYBBfd8Dm1snkkud431pdzEZ2uJTad1CiC
Password	CPUrig
Mining proxy/fallback	172.94.15[.]211:5443
Domain	softappsbase[.]top
Domain	autoupdatewinsystem[.]top
Domain	softwaredatabase[.]xyz
Configuration path	https://softappsbase[.]top/UnammnrsettingsCPU.txt
Configuration path	https://autoupdatewinsystem[.]top/UWP1/cpu.txt
Configuration path	https://softwaredatabase[.]xyz/UnammnrsettingsCPU.txt
Communication endpoint	https://softappsbase[.]top/UnamWebPanel7/api/endpoint.php
Communication endpoint	https://autoupdatewinsystem[.]top/UWP1/api/endpoint.php
Communication endpoint	https://softwaredatabase[.]xyz/UnamWebPanel7/api/endpoint.php

Here is a [GitHub Gist](#) of a list of sample hashes.

Jump to section

- [Introduction](#)
- [Key takeaways](#)
- [Campaign 1 \(CNB Bot\)](#)
- [Campaign 2 \(PureRAT\)](#)
- [Campaign 3 \(PureRAT, PureMiner, XMRig loader\)](#)
- [Custom .NET-Based Loader for XMRig](#)
- [The Entry Point and Setup](#)
- [Resource Extraction](#)
- [Evasion Loop](#)
- [WinRing0x64.sys and Ring 0 Access](#)

Elastic Security Labs Newsletter

[Sign Up](#)