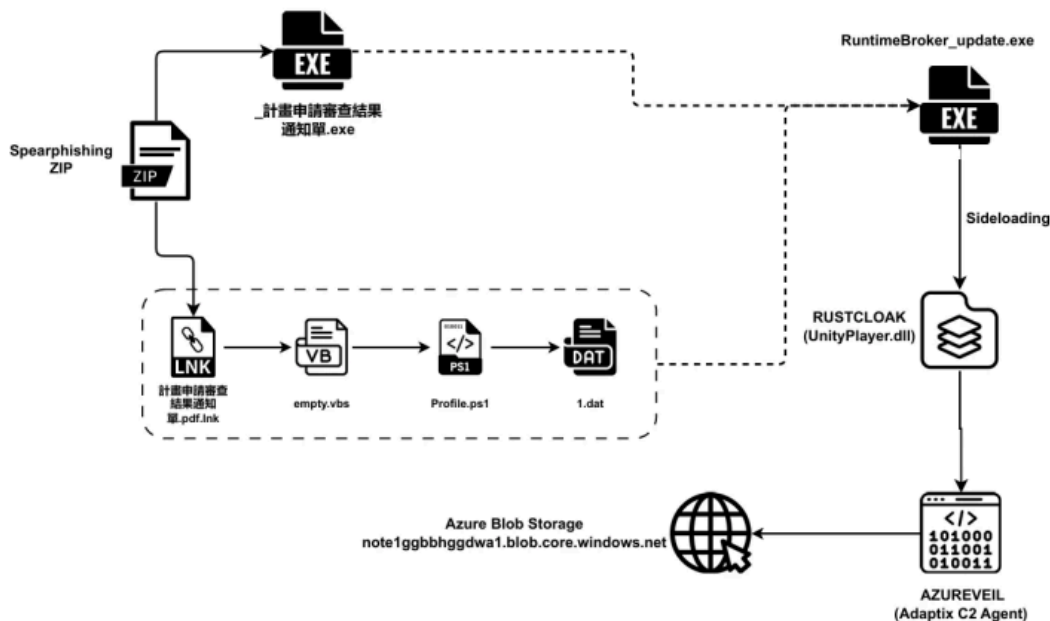


Operation Dragon Weave : Uncovering a China-Linked Campaign Targeting Czech Republic and Taiwan Using Azure Cloud C2

seqrite.com/blog/operation-dragon-weave-uncovering-a-china-linked-campaign-targeting-czech-republic-and-taiwan-using-azure-cloud-c2

Priya Patel

May 29, 2026



SEQRITE APT-Team

[Home](#) / [Technical](#) / Operation Dragon Weave : Uncovering a China-Linked Campaign Targeting Czech Republic and Taiwan Using Azure Cloud C2

Contents

- Introduction
- Key Targets
 - Industries Affected
 - Geographical focus
- Infection Chain
- Initial Findings
 - Looking into the Decoy Document
- Technical Analysis
 - Stage 1 – Initial Delivery
 - Path A: LNK-Based Execution
 - Path B: Executable-Based Delivery
 - Stage 2 – Script-Based Dropper Chain
 - Stage 3 – RUSTCLOAK (Rust Loader)
 - Stage 4 – AZUREVEIL (Adaptix C2 Agent)
- Infrastructure & Attribution
- Conclusion
- SEQRITE Protection
- Indicators of Compromise (IOCs)
- MITRE ATT&CK Mapping
- Authors

Introduction

The Seqrite APT Team has been actively tracking threats across the globe. During our recent analysis, we identified a spearphishing campaign targeting officials and citizens in the Czech Republic and Taiwan. We observed a single lure document along with multiple supporting artifacts that strongly suggest the campaign is specifically targeting these regions, as the files closely mimic official communications.

The attack begins with a ZIP attachment. When extracted, the archive contains multiple files that appear legitimate but are actually part of a structured infection chain designed to execute malicious payloads in the background.

This blog provides a detailed walkthrough of the full infection process and explains how the attacker uses two different delivery paths to execute the payload. It also examines how trusted services such as Microsoft Azure Blob Storage are abused for command-and-control communication, and how the Adaptix agent is used for data exfiltration and remote control.

In addition, we analyze the multi-layer encryption used to protect the payload and how it helps the attacker evade detection. The blog also presents key indicators of compromise (IOCs), highlights important findings from our analysis, and maps the observed behavior to MITRE ATT&CK techniques to support attribution of this campaign to a China-linked threat actor.

Key Targets

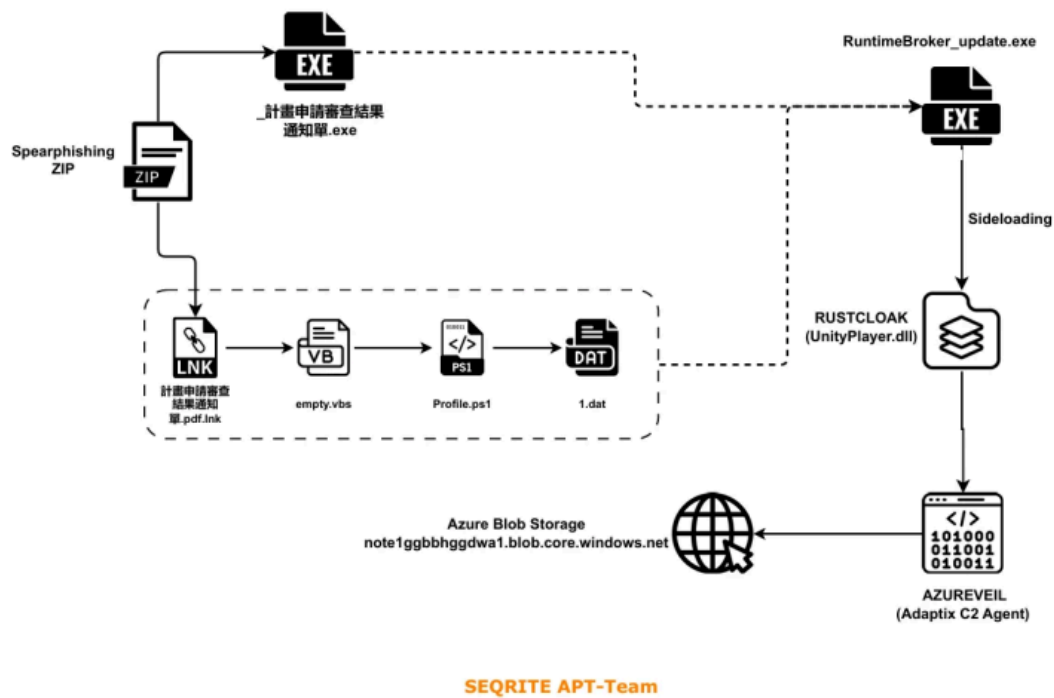
Industries Affected

- Government & Public Sector
- Research & Academia
- Technology & Software
- Financial Services

Geographical focus

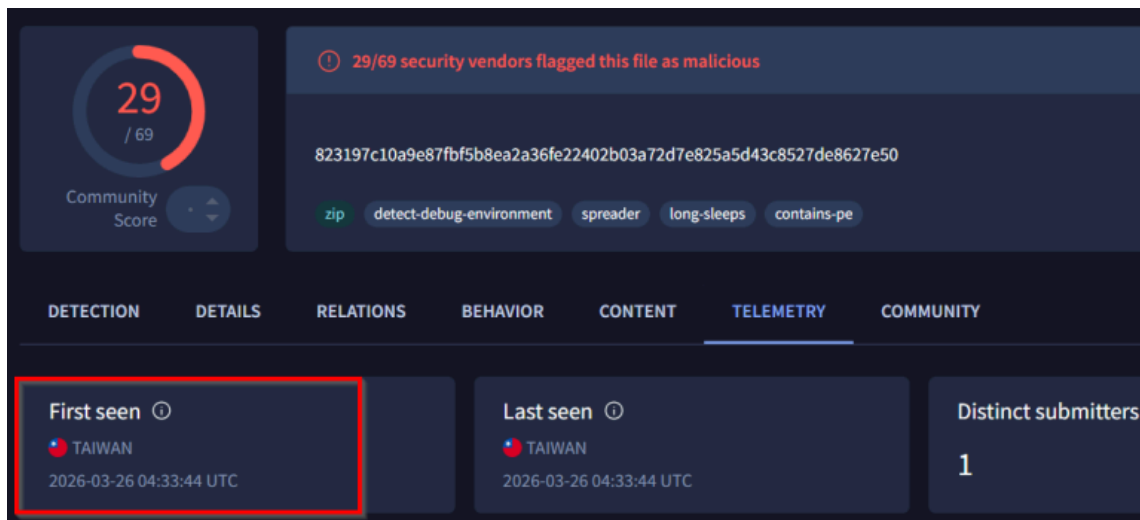
- Czech Republic
- Taiwan

Infection Chain



Initial Findings

As part of our ongoing threat monitoring efforts, the SEQRITE Labs team identified a suspicious ZIP archive on the threat intelligence platform [VirusTotal](#).



Based on the available telemetry, the file was first submitted from Taiwan on March 26, 2026. This initial submission provides an important clue, as it may indicate the region where the campaign was first observed or actively targeting.



data



計畫申請審查結果通知單.
pdf.lnk



_計畫申請審查結果通知單.
exe

After analyzing the contents of the extracted ZIP archive, we identified several suspicious files, including a folder named *data*, a shortcut file 計畫申請審查結果通知單.pdf.lnk masquerading as a PDF document, and an executable _計畫申請審查結果通知單.exe disguised as a legitimate file. The filename translates to “*Project Application Review Result Notification*,” suggesting an effort to appear as an official document. The filenames are written in Traditional Chinese, commonly used in Taiwan, which further suggests that the campaign is targeting users in that region.



000b67d70f3876965bb09fd
37164b7.pdf



1.dat



Com.dat



empty.vbs



Profile.ps1



UnityPlayer.dll

After analyzing the contents of the *data* folder, we found multiple files involved in different stages of the infection chain, including encrypted payload containers (1.dat and Com.dat), scripts files (Profile.ps1 and empty.vbs), a malicious DLL used for sideloading (UnityPlayer.dll), and a decoy PDF file used to distract the victim.

After analyzing the executable file, we observed that it spawns an additional lure document designed to resemble an official document from the Czech Republic. This document will be analyzed in detail later in the blog, and its presence further indicates that the campaign is also targeting Czech officials.

Looking into the Decoy Document

After examining the contents of the ZIP archive, we discovered a lure document inside the *data/* folder named 000b67d70f3876965bb09fd37164b7.pdf. Additionally, upon analyzing the main executable _計畫申請審查結果通知單.exe, we found another lure document embedded within the binary itself. At first glance, both documents seem similar due to their naming, but their contents are different.

 欢迎使用 WPS PDF

自由编辑 PDF

WPS PDF 提供了强大的 PDF 编辑能力, 你可以像编辑 Word 文档一样修改 PDF 内的文字和图片。



PDF 转换 / 压缩 / 拆分

格式转换、PDF 与图片互转、PDF 压缩与拆分等, 旨在满足不同需求。



AI 助手: 高效阅读利器

包括超长文档内容提炼、OCR 纠偏及裁剪、AI 增强显示, 以及 AI 全文翻译功能。



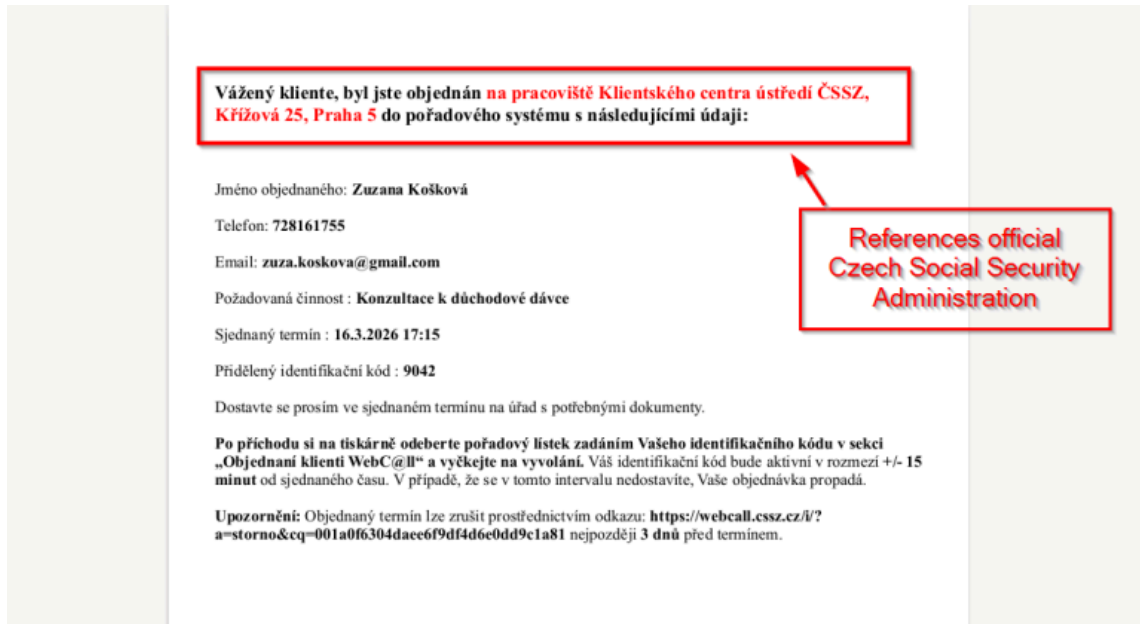
提取内容

提取 PDF 中的内容、页面、表格、文字和图片。简单操作, 即可实现高效的文件处理。



The file 000b67d70f3876965bb09fd37164b7.pdf was identified within the data/ folder and is encrypted using a single-byte XOR with the key 0xBE. Upon decryption, we found the document is written in Chinese and contains content related to a WPS PDF software advertisement. Despite having a naming pattern similar to the lure document embedded within the executable, this file is not referenced or used at any stage of the infection chain.

This suggests that the document may have been unintentionally included by the threat actor, possibly as a leftover artifact from a different campaign, where such a document could have served as a decoy.



The decoy file is embedded inside the executable 計畫申請審查結果通知單.exe and is the document shown to the victim during execution. The document name is 000b67d70f3876965bb09fd37164b7ccrezervaci.pdf, which combines a hash-like string with a Czech word (“rezervaci”). The document is written in Czech and appears to be an appointment notice from the Czech Social Security Administration (ČSSZ), likely taken from the official website cssz.cz. It includes details such as a scheduled appointment for a person named Zuzana Košková on March 16, 2026, making it appear as a genuine government-issued document.

The final section of the document provides additional instructions to the recipient, guiding them on what to do upon arrival at the office. It explains how to obtain a queue number using the provided identification code and instructs the user to wait for their turn, similar to standard appointment procedures.

The document also includes a link to the official Czech Social Security Administration (ČSSZ) website for managing or canceling the appointment. It appears to have been created by saving or printing the original website as a PDF.

In the next section we will look into the technical aspects of this campaign.

Technical Analysis

While going through the ZIP contents we noticed something interesting, there were actually **two ways** the infection chain would ultimately side load the malicious DLL file which contains the payload, depending on which file the victim interacted with first.

In Path A, the infection begins when the victim clicks on the malicious LNK file 計畫申請審查結果通知單.pdf.lnk. This action silently triggers a VBScript (empty.vbs), which then launches a PowerShell script (Profile.ps1). The PowerShell script is responsible for decrypting 1.dat, writing the decrypted content as RuntimeBroker_update.exe, and executing it.

In Path B, the victim directly runs 計畫申請審查結果通知單.exe. This executable acts as a self-contained Rust-based dropper that extracts all required components on its own and then launches the same RuntimeBroker_update.exe.

Even though the initial steps are different, both paths end up at the same stage. RuntimeBroker_update.exe loads a malicious UnityPlayer.dll using DLL sideloading, which leads to the execution of a Rust-based loader we call **RUSTCLOAK**. This loader then decrypts and runs the final payload, **AZUREVEIL**, an Adaptix C2 agent.

Stage 1 – Initial Delivery

Path A: LNK-Based Execution

The first delivery method starts with a malicious Windows shortcut file named 計畫申請審查結果通知單.pdf.lnk, which translates to “Project Application Review Result Notice.pdf.” The use of a double extension (.pdf.lnk) is intentional, making the file look like a regular PDF document.

```
Volume label :  
Local path   : C:\Windows\System32\wscript.exe  
Relative path : C:\Windows\System32\wscript.exe  
Command line arguments : ".\data\empty.vbs"  
Icon location : %ProgramFiles(x86)%\Microsoft\Edge\Application\msedge.exe  
Environment variables location :
```

The LNK file is configured to execute wscript.exe from the Windows system directory, passing .\data\empty.vbs as an argument. Instead of opening a PDF, the shortcut silently runs this VBScript file in the background. At the same time, it uses the Microsoft Edge icon to appear like a normal document.

Path B: Executable-Based Delivery

The second delivery method uses a Rust-compiled executable named _計畫申請審查結果通知單.exe. This file acts as a complete dropper, with all required components embedded inside it, so it does not rely on any external files from the ZIP archive.

```
db 0C1h  
db 1  
db 0  
db 0  
db 10h  
db 0  
db 0  
db 0  
db 0  
aLocalappdataw db 'LOCALAPPDATAWebViewFixUtilityBrowserViewUtility.exeUnityPlayer.dll'  
                ; DATA XREF: sub_1400028D0+24+ro  
                ; sub_1400028D0+151+o  
db 'lCom.dat001a0f6304dae6f9df4d6e0dd9c1a81rezervaci.pdfMZ'  
db 90h  
db 0  
db 3  
db 0  
db 0  
db 4  
db 0  
db 0  
db 0  
db 05Fh
```

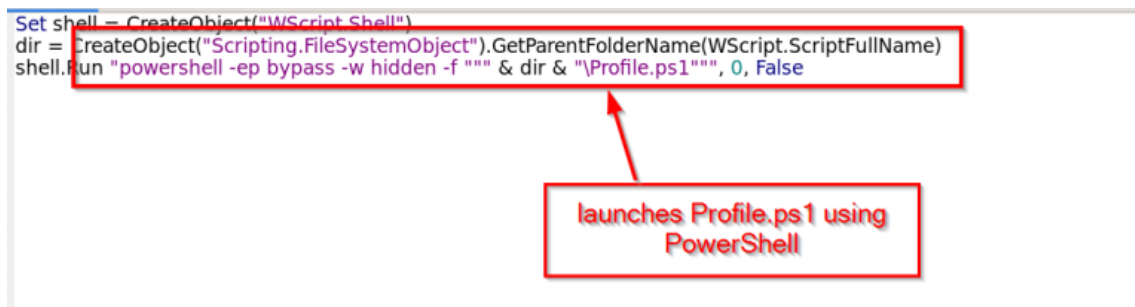
All dropped file names and the staging directory %LOCALAPPDATA%\WebViewFixUtility\

During dynamic analysis, we found the binary performs several actions on its own. It first creates a directory at %LOCALAPPDATA%\WebViewFixUtility to store its files. It then drops multiple components, including BrowserViewUtility.exe (a legitimate executable used for sideloading), a malicious UnityPlayer.dll (RUSTCLOAK), an encrypted configuration file, a decoy document (000b67d70f3876965bb09fd37164b7ccrezervaci.pdf), and RuntimeBroker_update.exe.

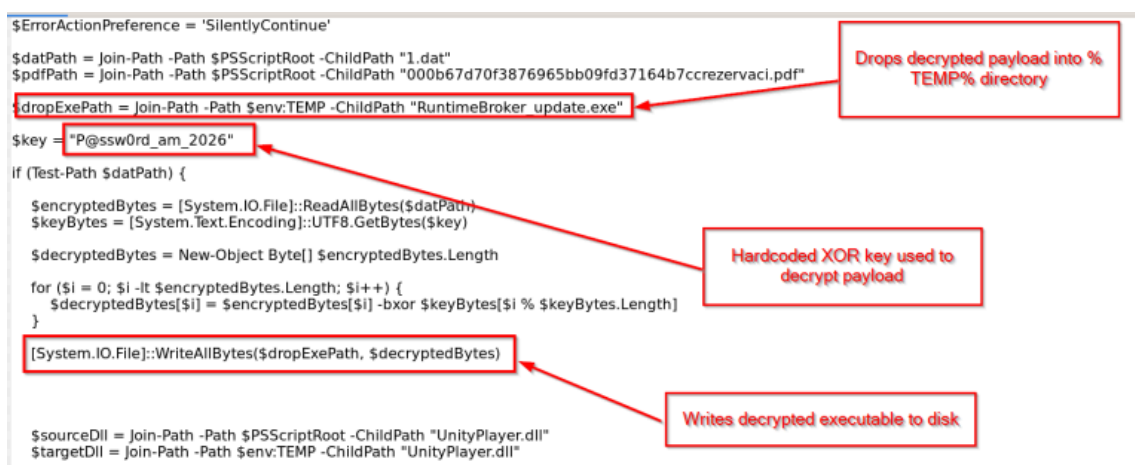
Stage 2 – Script-Based Dropper Chain

Path B handles everything internally so from here we focus only on **Path A**, the script-based chain that moves through three additional components: empty.vbs, Profile.ps1, and 1.dat. Each of these plays a role in preparing and executing the final binary, RuntimeBroker_update.exe, which is later used for DLL sideloading.

Once the victim clicks on the LNK file, *empty.vbs* gets executed and the script-based chain begins. This flow runs quietly in the background, passing control from one component to the next without showing anything suspicious to the user.



After analysing the VB Script we found it is a very small VBScript with a single purpose that is to launch Profile.ps1 using PowerShell. It runs PowerShell with execution policy bypass enabled and in a hidden window, so nothing is visible to the user. The script itself does not contain any real logic and simply acts as a bridge to the next stage.



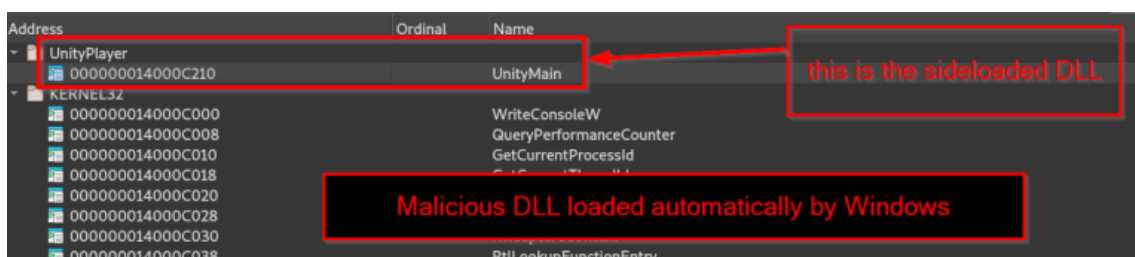
The main logic of the infection chain is handled by Profile.ps1. This script reads *1.dat* from the *data/* folder and decrypts it using a simple XOR operation with the hardcoded key *P@ssw0rd_am_2026*. After decryption, the file becomes *RuntimeBroker_update.exe*, which is then written to disk.

The script also moves *UnityPlayer.dll* and *Com.dat* to the *%TEMP%* directory so that all required components are placed together. Once everything is ready, it executes *RuntimeBroker_update.exe* in the background and opens the decoy document to keep the user distracted.

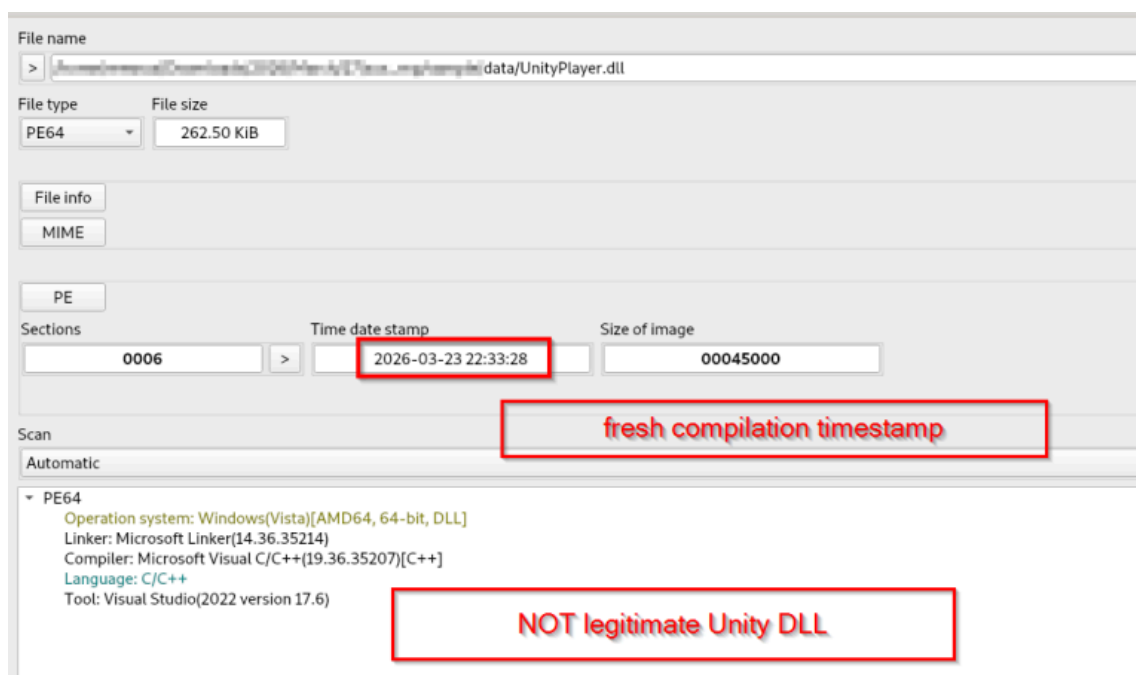
The file *1.dat* is not a regular executable. It is an XOR-encrypted container that holds *RuntimeBroker_update.exe*. The encryption is simple and uses a short key, likely to avoid basic detection. Once decrypted by *Profile.ps1*, the actual executable *RuntimeBroker_update.exe* is dropped and executed.

As described earlier, the campaign uses two different entry points depending on how the victim interacts with the archive. Both paths eventually deliver the same payload, *RuntimeBroker_update.exe*, but the way they do it is different. Path A follows a multi-step script-based chain that starts when the victim clicks the LNK file, while Path B is self-contained and only requires the victim to run the executable.

Stage 3 – RUSTCLOAK (Rust Loader)



Both paths converge here and RuntimeBroker_update.exe is now running. The first thing Windows does is look for UnityPlayer.dll in the same folder. The attacker takes advantage of this by placing a malicious UnityPlayer.dll in the same folder. Because of this, Windows loads the attacker's DLL instead of the legitimate one.



This DLL, which we call RUSTCLOAK, is a Rust-based loader responsible for continuing the execution chain and preparing the final payload.

Developer Artifacts

```

db 0
db 0
db 0
aUsersDell2Car_0 db 'C:\Users\dell2\.cargo\registry\src\index.crates.io-1949cf8c6b5b55'
; DATA XREF: .rdata:off_180029340;io
; .rdata:off_180029358;io ...
db '7f\base64-0.21.7\src\engine\general_purpose\decode.rs',0
align 20h
off_180029340 dq offset aUsersDell2Car_0
; DATA XREF: sub_1800022B0+B03;io
; "C:\\Users\\dell2\\.cargo\\registry\\src"...
db 77h ; w
db 0
db 0
db 0

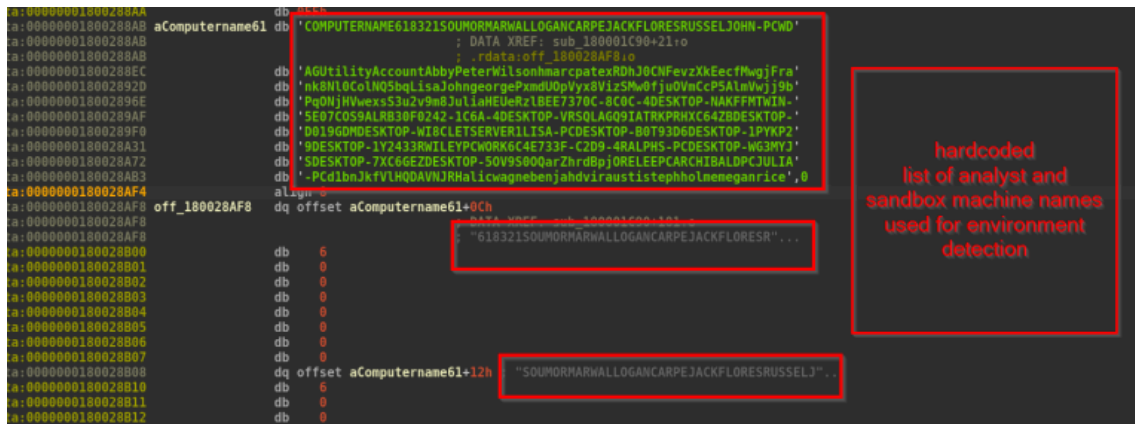
```

One of the key findings during our analysis of **RUSTCLOAK** was an operational security mistake made by the threat actor. The Rust build path `C:\Users\dell2\.cargo\registry\src\index.crates.io-1949cf8c6b5b557f\src\decrypt_SM4.rs` was left inside the binary as plaintext, exposing details from the developer's system:

This exposes the developer's Windows username as `dell2` and the specific library versions used such as `libsm-0.5.1` for SM4 encryption and `base64-0.21.7` for Base64 operations.

Anti-Analysis: Sandbox Detection

Before carrying out any malicious activity, **RUSTCLOAK** checks whether it is running in a sandbox or analysis environment. It retrieves the system's computer name and compares it against a hardcoded list of over 100 known sandbox and analyst machine names.



If a match is found, the loader exits immediately without executing the payload. The list includes commonly used names such as *DESKTOP-NAKFFMT*, *JULIA-PC*, and *ARCHIBALD-PC*, which are often seen in automated analysis setups. This check helps the malware avoid detection during analysis.

Triple-Layer Decryption Chain

Once **RUSTCLOAK** is loaded, it reads the encrypted payload (Com.dat) from disk and processes it through multiple layers of decryption before it can be executed.

Layer 1: Custom RC4 Decryption – The first stage uses a modified RC4 algorithm. The key used for this step is: F8 83 40 17 1D 66 AA C2 B0 25 A8 6C A0 DD C4 5A

Layer 2: Base64 Decoding – After the RC4 step, the output is still not directly usable. It is encoded in Base64 format, so the loader decodes it to get the actual binary data for the next stage.

Layer 3: SM4-CBC Decryption – In the final step, the data is decrypted using SM4 in CBC mode. The key and initialization vector used are:

Key: CD CE 4F DB 3E 6A F2 44 AC 62 8C F4 96 1F 6B FB

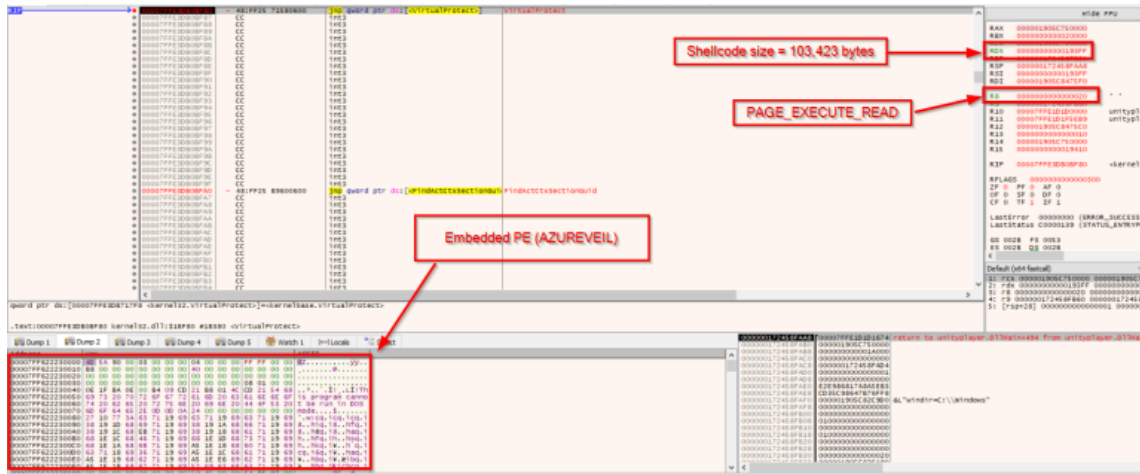
IV: FA 70 B1 81 A0 BA 5D 46 7A 5D 40 DD 99 B6 9B 42

In-Memory Execution via Windows fibers



Once the decryption process is complete, **RUSTCLOAK** allocates memory using `VirtualAlloc` and copies the decrypted shellcode into it. It then changes the memory permissions to executable using `VirtualProtect` before running the payload.

Instead of creating a new thread, which is often monitored by security tools, it uses Windows fibers for execution. The loader calls `CreateFiberEx` and then `SwitchToFiber` to pass control to the shellcode.



While debugging the loader, we observed that it allocates a large memory region and copies the decrypted payload into it. On closer inspection of the memory region, we identified a valid PE header (MZ), confirming that the shellcode actually contains a full executable embedded within it. The size of this in-memory payload was observed to be approximately 103 KB.

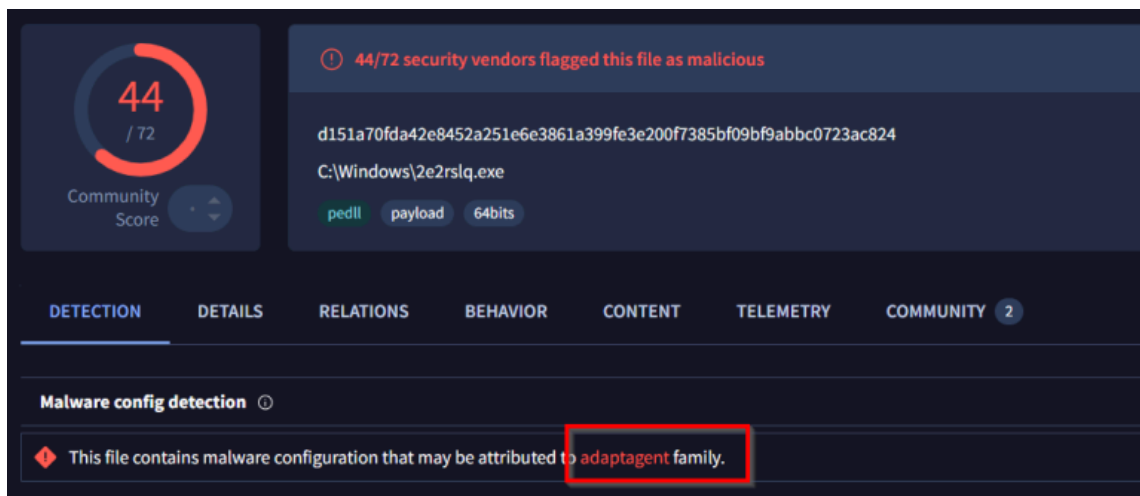
We extracted this in-memory PE file during debugging, which turned out to be the final payload, **AZUREVEIL**. This shows that **RUSTCLOAK** is not just executing raw shellcode directly, instead it decrypts and loads a full executable in memory and then transfers execution to it.

Stage 4 – AZUREVEIL (Adaptix C2 Agent)

AZUREVEIL is the final payload and the most technically interesting part of this campaign.



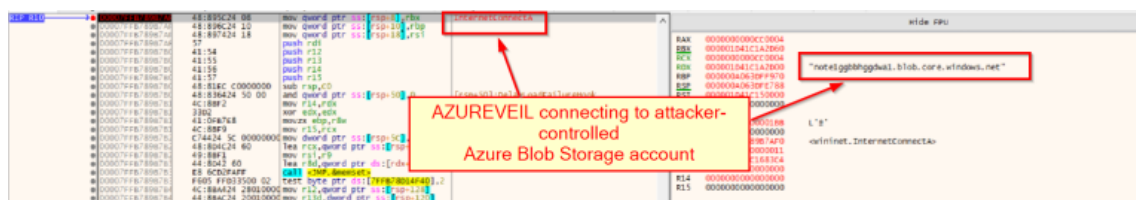
After analyzing the file using DIE, we found that **AZUREVEIL** is a fully featured Adaptix C2 agent compiled as a 64-bit DLL using MinGW C++.



During analysis on VirusTotal, the sample was flagged as malicious by multiple vendors, with detection pointing to embedded configuration linked to the Adaptix malware family.

We observed that **AZUREVEIL** resolves around 87 Windows APIs at runtime using a djv2-based hashing method. It loads these APIs from core libraries such as *wininet.dll*, *Ws2_32.dll*, *Advapi32.dll*, *lphlpapi.dll*, and *msvcrt.dll*.

Azure Blob Storage C2 Communication



During our analysis, we observed that **AZUREVEIL** uses Microsoft Azure Blob Storage for command-and-control communication. The interesting part is the C2 mechanism because there is no traditional C2 server at all. The malware just talks to Azure Blob Storage, the same service used by thousands of legitimate enterprises worldwide.

C2 Endpoint: note1ggbhggdwa1[.]blob[.]core[.]windows[.]net

Instead of using a traditional pull-based C2 model, **AZUREVEIL** follows a dead-drop approach. The attacker and the infected system never communicate directly. Instead, both sides use the same Azure storage container to exchange data.

The agent periodically uploads a small encrypted beacon (around 124 bytes) to signal that it is active. The attacker then places commands in the same container. **AZUREVEIL** retrieves these commands, decrypts them, executes them, and uploads the results back as encrypted blobs.

Command Capabilities of AZUREVEIL



During our static and dynamic analysis, we identified 36 commands supported by **AZUREVEIL**, covering a wide range of post-exploitation activities. These capabilities allow the attacker to fully control the infected system, perform data theft, and move laterally across the network.

File System Operations

- List directory contents and logical drives
- Read, move, rename, and delete files
- Exfiltrate files to Azure Blob Storage
- Drop files from C2 to the victim system
- Queue file downloads

Process and Shell Control

- Execute shell commands
- List running processes and named pipes

- Configure and terminate processes
- Kill processes or close pipes

Network and Pivoting

- Port forwarding and SOCKS proxy control
- TCP and UDP pivot connections
- Named pipe communication
- Network adapter enumeration (MAC, IP, type)

C2 Management

- Reconfigure C2 settings at runtime
- Control file transfer state
- Retrieve system uptime

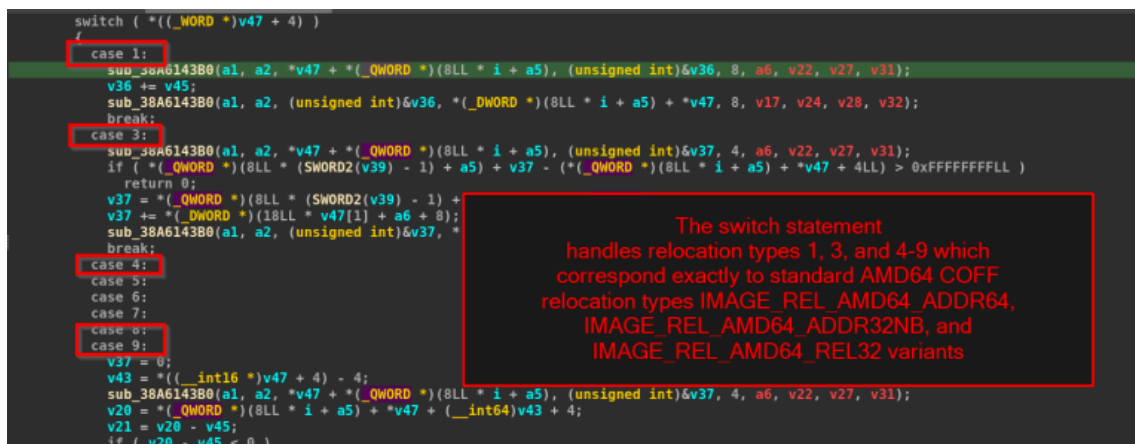
In-Memory Code Execution

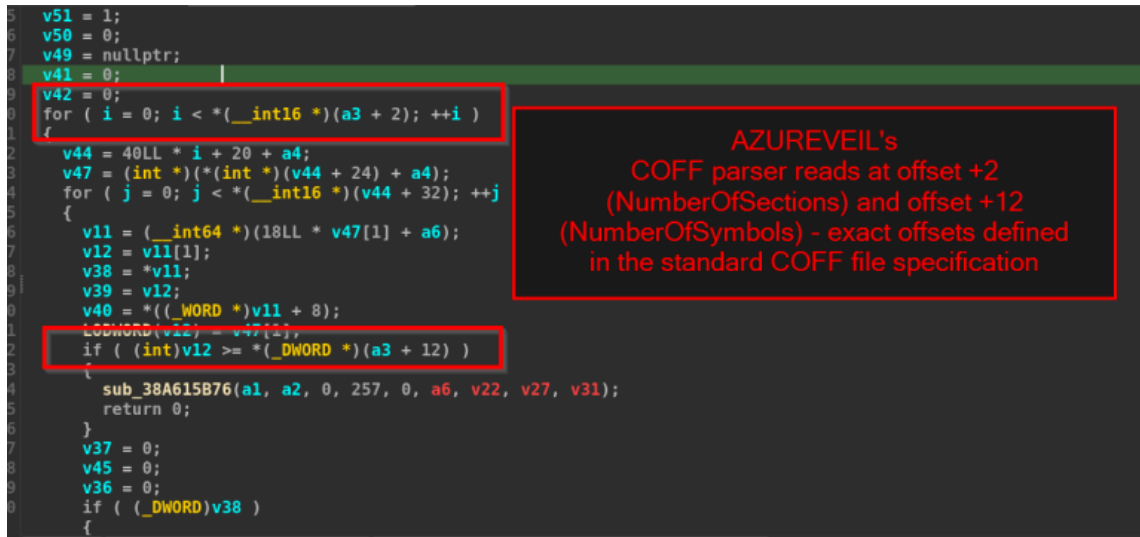
Execute Beacon Object Files (BOF) entirely
in memory without writing to disk

BOF Execution Engine



Command 0x32 caught our attention during static analysis. At first it looked like a generic injection routine, but after analyzing the relocation code we found it was more than that.





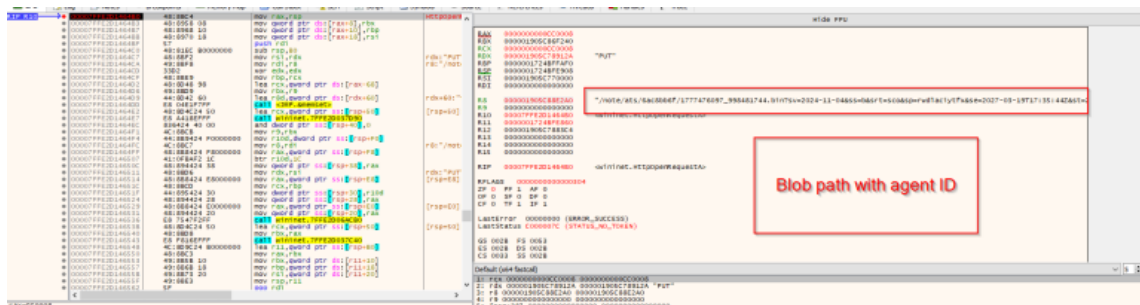
The function was reading data at offset +2 for section count and offset +12 for symbol count, the exact same offsets defined in the standard COFF file format. To confirm this, we compiled a real AMD64 COFF object file on our analysis machine and compared its structure against the parser.

When a BOF is received, AZUREVEIL first allocates memory and loads the different sections of the object file into it. It then processes the required relocations and resolves function references using its internal hash-based lookup. After setting everything up, the loader executes the BOF entry point directly within the agent process.

Any output generated during execution is captured using a named pipe and then sent back to the attacker through Azure Blob Storage. This lets the attacker run additional functions on the system and get the results back, all while remaining completely in memory.

Infrastructure & Attribution

During our analysis, we observed that **AZUREVEIL** communicates entirely through Microsoft Azure Blob Storage, abusing legitimate cloud infrastructure for command-and-control. Instead of using a traditional C2 server, the attacker relies on a dedicated Azure storage account: note1ggbbhggdwa1[.]blob[.]core[.]windows[.]net



Further analysis revealed the following details about the storage setup:

Container: /note/ats/
 Blob format: {agent_id}/{timestamp1}_{timestamp2}.bin
 Agent ID: 345831bc
 Port: 443 (HTTPS)

All communication happens over HTTPS on port 443, which makes the traffic blend in with normal Azure activity. Since blob.core.windows.net is widely used by legitimate applications, this approach helps the attacker avoid network-level detection.

We also identified a hardcoded Shared Access Signature (SAS) token found inside encrypted_blob which is dropped by the 計畫申請審查結果通知單.exe executable, which is used to authenticate all operations on the storage account:

```
sv=2024-11-04&ss=b&srt=sco&sp=rwdlaciytfx&st=2026-03-19T09:20:44Z&se=2027-03-19T17:35:44Z&spr=https&sig=ECJjJIIE9Ou75dwiHhliC4fWccdBpLX9u580AX9TGwY=
```

The token is valid for one year, from March 19, 2026 to March 19, 2027. This long validity period suggests that the attacker intended to maintain persistent access over an extended time. The permissions associated with the token allow full interaction with the storage container, including read, write, delete, and upload operations.

Based on the TTPs, tooling, and targeting observed throughout this campaign, we assess with moderate confidence that it is linked to a China-based threat actor. However, we do not attribute it to any specific known group, as the campaign includes techniques that have not been previously documented in publicly known Chinese APT activity.

Conclusion

Operation Dragon Weave is a targeted espionage campaign. One of the unique aspects of this campaign is its use of Microsoft Azure Blob Storage as a dead-drop C2 channel. Instead of talking to a normal C2 server, the malware blends its traffic with regular cloud activity, which makes it much harder to notice. The final payload, AZUREVEIL, is a fully featured Adaptix C2 agent with 36 post exploitation commands. Along with the in-memory BOF execution capability, it gives the attacker a lot of flexibility to run additional code and control the system without leaving much behind on disk.

We also observed that the campaign targets both the Czech Republic and Taiwan, using region-specific lure documents. This indicates that the operation is planned and targeted rather than random. Based on the use of distinct tools and techniques, we assess that this campaign is linked to a China-based threat actor.

SEQRITE Protection

Lnk.Trojan.50646.GC

Trojan.Agent.S38943638

Script.Trojan.50655.SL

Script.Trojan.50650.SL

Indicators of Compromise (IOCs)

SHA-256File Name

096372d19b4787e989f44e04c5ecc29885aa927c34ae8666628d6c0eb20bb447	計畫申請審查結果通知單.pdf.lnk
1c56228cbd1bdebb9e5ea55c2749150fee06c865ede4a3754e8bd6843e51d2d4	計畫申請審查結果通知單.exe
080ab9bc2893ba7bad354551604a667af40ed2ae2d042d2323c2bd9ad3122192	UnityPlayer.dll
5ed14c2b7f7433a1a72dd6b668413f935a217ba10b69d89b774a82990fa12fe1	BrowserViewUtility.exe
61f7d9cd2d8ce7df950639b23ce90085b300b0c6dd0d8d934bba8fdecbb670f15	RuntimeBroker_update.exe
24aa4e780ccd66cef13da9ef98c32954105cf2a32ec643efab0ba1aa2d6352f4	Com.dat
02542a49b3bd6bd2795afb67840acb4557b17e017f7503dd03ebe3aeeb28720e	000b67d70f3876965bb09fd37164b7ccrezervaci.pdf
8ae7c82a3e4f742777e590b25a1c563d19bd9bcba2a387d004aae72c4b2828f9	000b67d70f3876965bb09fd37164b7.pdf
047687548605734348792e2a9d771b6cba42facd0d0d7d44d778290a25848574	1.dat

a4e9f9919d62589b57cfa08c9ccb89e386b09f683271373413cd8e8c8c7d1c5a	empty.vbs
823d5969db3f3b72ebbdce1b78752717ea849884a0fb40d86146416c38e128de	Profile.ps1
783661d0f7edb338d2d50be087764d82dbbc9ee7989ddc57db1801e4ec9045b0	azureveil.exe

Network Indicator

IndicatorDescription

note1ggbbhggdwa1[.]blob[.]core[.]windows[.]net Azure Blob Storage C2

MITRE ATT&CK Mapping

Tactic	Technique ID	Technique Name
Initial Access	T1566.001	Spearphishing attachment
Execution	T1204.002	Malicious file – user execution
	T1059.001	PowerShell
	T1059.005	Visual Basic
Defence Evasion	T1574.002	DLL side-loading
	T1027	Obfuscated files or information
	T1055	Process injection
	T1497.001	Virtualisation/sandbox evasion
	T1620	Reflective code loading
Discovery	T1083	File and directory discovery
	T1057	Process discovery
	T1016	System network configuration discovery
	T1082	System information discovery
Command & Control	T1102.001	Web service – dead drop resolver
	T1573	Encrypted channel
	T1090	Proxy
	T1105	Ingress tool transfer
Exfiltration	T1041	Exfiltration over C2 channel

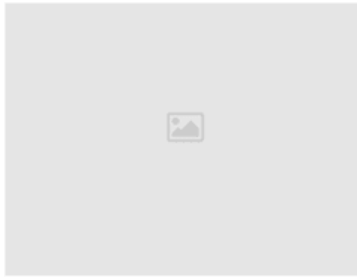
Authors

Priya Patel

Kartikkumar Jivani

Related Posts

-



-



-

