Avast Threat Labs analysis of CCleaner incident

blog.avast.com/avast-threat-labs-analysis-of-ccleaner-incident

Technical update and ongoing analysis of the APT security incident

Experts at Avast Threat Labs have been analyzing the CCleaner advanced persistent threat (APT) continuously for the past few days and apart from the information in recent blog posts (CCleaner and Avast posts), we are starting a series of technical blog posts describing details and technical information that we encountered during our analysis. Today, we will cover the ongoing analysis of the CnC server and the 2nd stage payload.

Just 4 days of data?

Shortly after receiving the initial notification about the incident from Morphisec, we reached out to law enforcement agencies to help us take down the Command and Control (CnC) server and get access to its contents. While analyzing the data, we noticed that there were only a few days' worth of data in the logs, and we wondered why? We knew the server was installed on July 31st so there had to be more than a month's worth of data since then:

```
Jul 31 06:32:53 seassdvz3.servercrate.com systemd[1]: Started First Boot Wizard.
```

Although the server was up and running since the end of July, data gathering started on August 11th, in preparation of the release of the compromised CCleaner executable file:

```
Aug 11 07:36:52 seassdvz3 mariadb-prepare-db-dir[10729]: Initializing MySQL
database
```

```
Aug 11 07:36:52 seassdvz3 mariadb-prepare-db-dir[10729]: Installing
MariaDB/MySQL system tables in '/var/lib/mysql' ...
```

The database didn't contain data older than September 12th, so we originally thought someone might have deleted the data to avoid being traced, but then we found this log:

```
170830 20:36:17 [Note] /usr/libexec/mysqld: ready for connections.
Version: '5.5.52-MariaDB'  socket: '/var/lib/mysql/mysql.sock'  port: 3306
 MariaDB Server
170910  8:47:40  InnoDB: Error: Write to file ./ibdata1 failed at offset 11
2854223872.
InnoDB: 1048576 bytes should have been written, only 0 were written.
InnoDB: Operating system error number 122.
InnoDB: Check that your OS and file system support files of this size.
InnoDB: Check also that the disk is not full or a disk quota exceeded.
InnoDB: Error number 122 means 'Disk quota exceeded'.
```

The MariaDB (fork of MySQL) database—which stored the data acquired by the backdoor—ran out of disk space. Not coincidentally, there was a connection to the machine just a few hours after the database died:

```
root     pts/0        Sun Sep 10 20:59 - 23:34  (02:34)     124-144-xxx-
xxx.rev.home.ne.jp
```
(actual address was redacted)

The user behind this connection came to free up some disk space. He (or she) started by erasing all the logs in the hope that this would quickly fix the issue, but the logs show the database also encountered some serious issues and was corrupted:

```
170910  8:47:43 [ERROR] mysqld got signal 6 ;
```

Two days later, another connection was made, and this time, the attacker decided to resurrect the database by a complete reinstall:

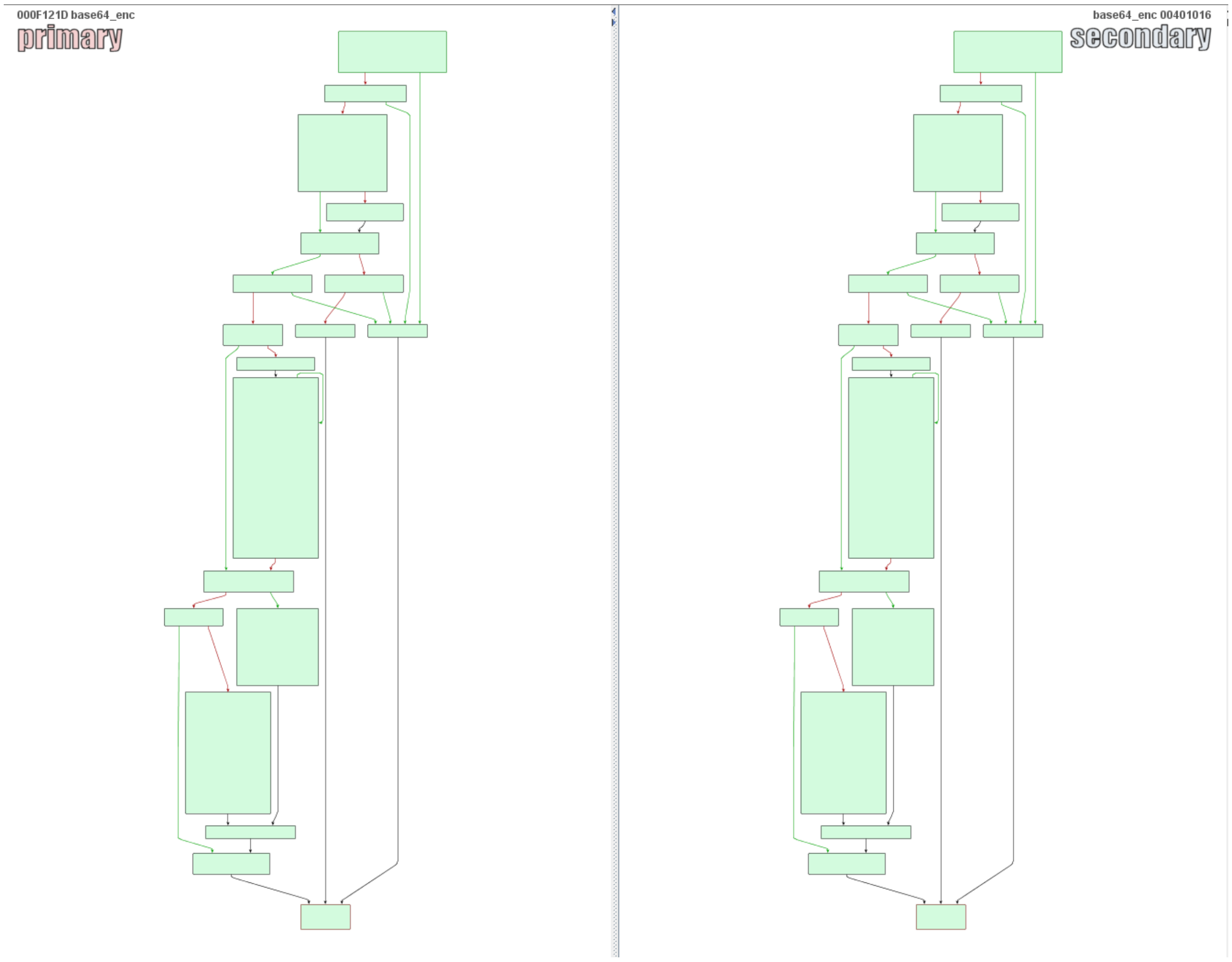
```
Sep 12 07:56:13 Erased: 1:mariadb-server-5.5.52-1.el7.x86_64
Sep 12 07:56:13 Erased: 1:mariadb-5.5.52-1.el7.x86_64
Sep 12 08:02:43 Installed: 1:mariadb-5.5.52-1.el7.x86_64
Sep 12 08:02:44 Installed: 1:mariadb-server-5.5.52-1.el7.x86_64
```

It is unfortunate that the server was a low-end machine with limited disk capacity, because if weren't for this (just 5 days before we took the server down), we would likely have a much clearer picture of exactly who was affected by the attack as the entire database would have been intact from the initial launch date.

Where did the attackers come from?

To figure out who the attackers were, we looked for any breadcrumbs the attackers might have left for us to follow. As Costin Raiu pointed out on Twitter (https://twitter.com/craiu/status/910148928796061696), there are some striking similarities between the code injected into CCleaner and APT17/Aurora malware created by a Chinese APT group in 2014/2015.

Indeed, the similarity between the code linked to group APT17 and the recent payload is quite high. Some of the functions are almost identical (e.g. the base64 encoding function on the following image) while other functions have a partial match, but the structure is overall very similar.



Next, we looked at where the attacker was connecting from to the CnC server.

```
 root      pts/0         Tue Sep 12 18:11 - 18:50  (00:39) xxxx.ap.so-net.ne.jp
 root      pts/0         Tue Sep 12 09:23 - 14:14  (04:51) xxxxx.bbtec.net
```

```
root     pts/0        Sun Sep 10 20:59 - 23:34  (02:34) 124-144-xxx-
xxx.rev.home.ne.jp
```
(+ other 32 connections)

Interestingly enough, most of the connections came from Japanese networks. Although these addresses are likely just infected PCs and servers used as proxies, it suggests that the attackers might be familiar with Asian networks. The list of targeted companies contained quite a few Asian companies but none from China. Lastly, the time zone in the PHP scripts feeding the database were set to PRC (People’s Republic of China) although the system clock is in UTC.

Even with all of these clues, it is impossible at this stage to claim which country the attack originated from, simply because all of the data points could easily be forged to hide the true location of the perpetrator.

Targeted companies - South Korea or Slovakia?

In addition to the domain names of targeted companies already published, there were four more domains belonging to two more companies that haven’t been mentioned publicly (we don’t want to disclose the names of these companies as they were potentially subjected to the attack). These domains were commented out in the scripts, which can indicate the list of targeted companies had changed repeatedly over time. This is further supported by the fact that some of the 20 computers that we know received the 2nd stage payload were in domains that were also not included in the original list.

As a side note, the attackers seem to have made a mistake with the domain name of one company specified as “<company>.sk”. We suppose they wanted to target the South Korean users (i.e. the headquarters), but the domain .sk actually belongs to the Slovak Republic so they were unknowingly trying to infect users from the Slovakian branch of the company!

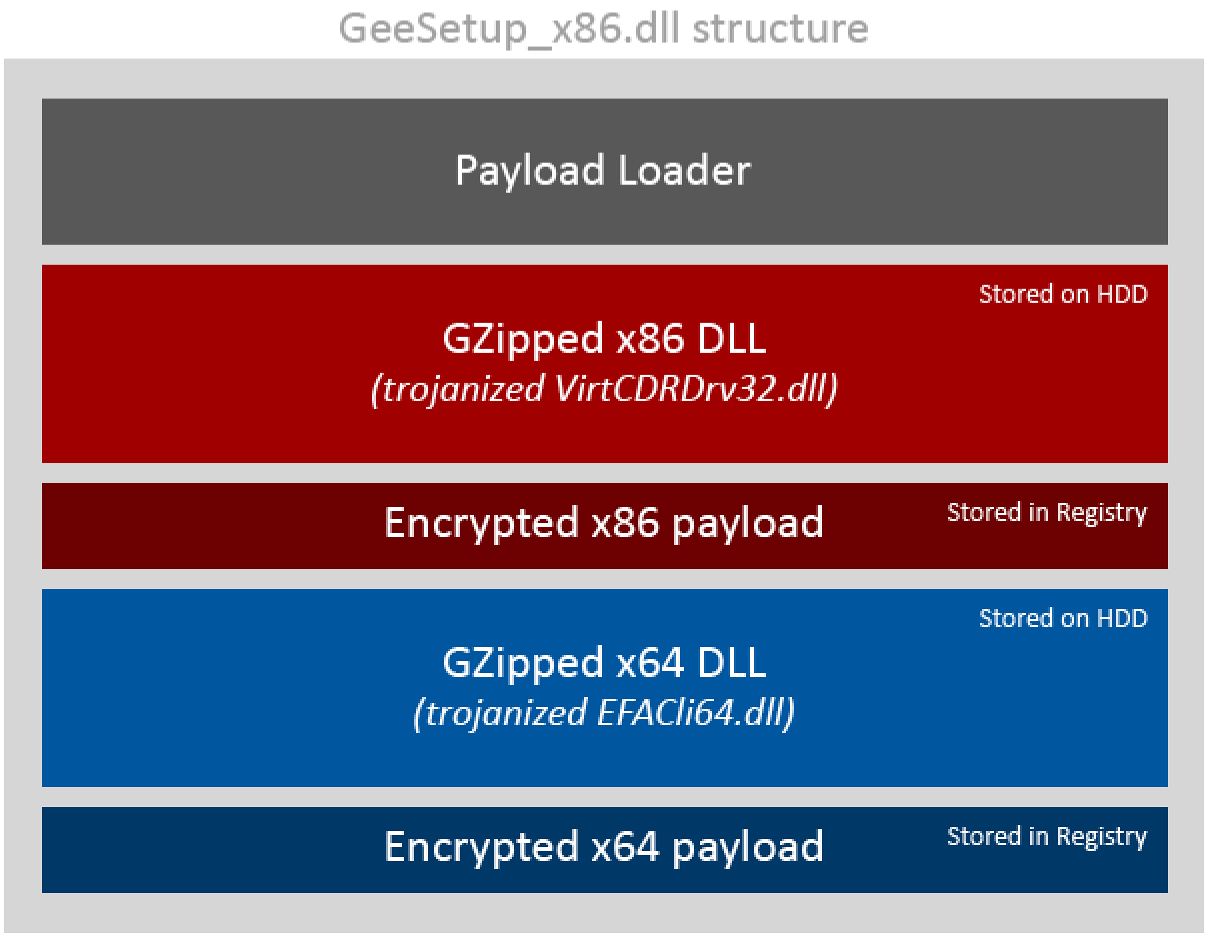
Matryoshkas

Ever heard of the Russian nesting doll Matryoshka? It’s a set of dolls of decreasing size placed one inside another, and while analyzing the 2nd stage payload binary, we had a sense of playing with Matryoshkas ourselves as there were multiple levels of indirection that we had to go through.

The backdoor in CCleaner called home to receive the second stage payload which we found in the server dump under the name GeeSetup_x86.dll.

Opening the first Matryoshka, we see two more containing 32- and 64-bit payloads, each of which piggybacked on a different legitimate binary along with additional malware for the appropriate architecture. The 32-bit version used patched TSMSISrv.dll which originally is a

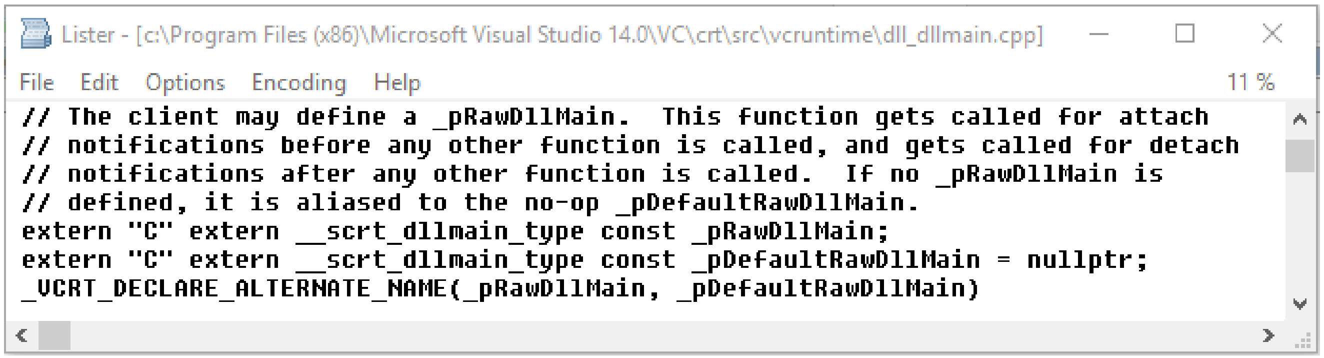
VirtCDRDrv32.dll created by Corel. The 64-bit version used a patched EFACli64.dll originally developed by Symantec. The 2nd stage is illustrated in the next figure:



When the DLLs were loaded, they saved the embedded malware into the registry and used elaborate tactics to extract the registry loader routine and run it. This procedure is described in the following section.

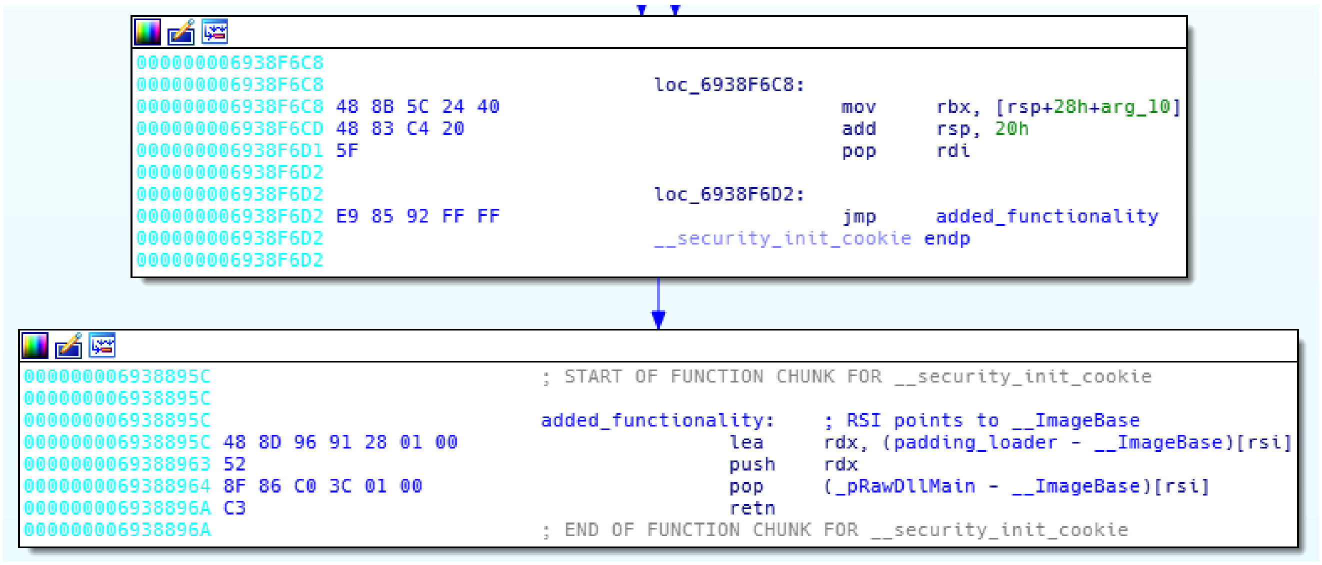
Hacked CRT

One way to show how sophisticated the attackers were is to look at the way they modified the C runtime (CRT). We will demonstrate this on the 64-bit version. CRT is a piece of code that contains important functions needed for the program to run. The modified CRT code can be found in the second stage payload which is embedded in the original Symantec code. The modifications are performed by adding a few instructions to the function __security_init_cookie which is ironically responsible for securing the code from buffer overflows—the well-known “canary”. The added instructions change the _pRawDllMain function pointer to point to the special function that extracts a hidden registry payload loader. The following figure explains the functionality behind the _pRawDllMain.



```
// The client may define a _pRawDllMain.  This function gets called for attach
// notifications before any other function is called, and gets called for detach
// notifications after any other function is called.  If no _pRawDllMain is
// defined, it is aliased to the no-op _pDefaultRawDllMain.
extern "C" extern __scrt_dllmain_type const _pRawDllMain;
extern "C" extern __scrt_dllmain_type const _pDefaultRawDllMain = nullptr;
_VCRT_DECLARE_ALTERNATE_NAME(_pRawDllMain, _pDefaultRawDllMain)
```

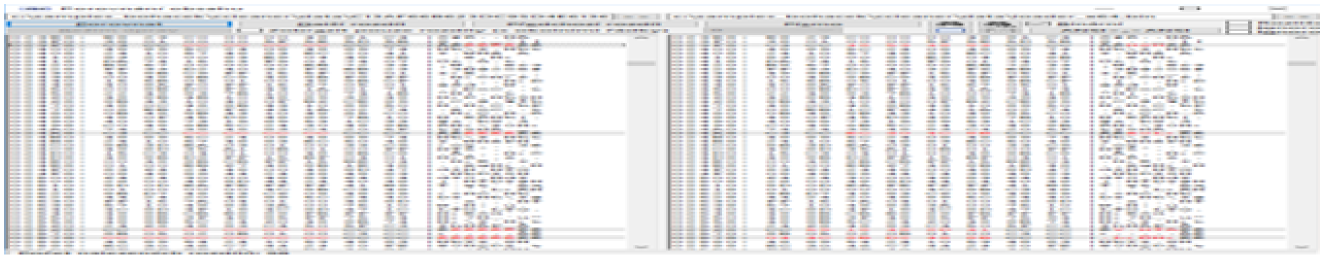
There is an added jump and a few instructions at the end of CRT function __security_init_cookie.



```
000000006938F6C8
000000006938F6C8                            loc_6938F6C8:
000000006938F6C8 48 8B 5C 24 40                             mov     rbx, [rsp+28h+arg_10]
000000006938F6CD 48 83 C4 20                                add     rsp, 20h
000000006938F6D1 5F                                         pop     rdi
000000006938F6D2
000000006938F6D2                            loc_6938F6D2:
000000006938F6D2 E9 85 92 FF FF                             jmp     added_functionality
000000006938F6D2                            __security_init_cookie endp
000000006938F6D2
```

```
000000006938895C                            ; START OF FUNCTION CHUNK FOR __security_init_cookie
000000006938895C
000000006938895C                            added_functionality:    ; RSI points to __ImageBase
000000006938895C 48 8D 96 91 28 01 00                   lea     rdx, (padding_loader - __ImageBase)[rsi]
0000000069388963 52                                     push    rdx
0000000069388964 8F 86 C0 3C 01 00                      pop     (_pRawDllMain - __ImageBase)[rsi]
000000006938896A C3                                     retn
000000006938896A                            ; END OF FUNCTION CHUNK FOR __security_init_cookie
```

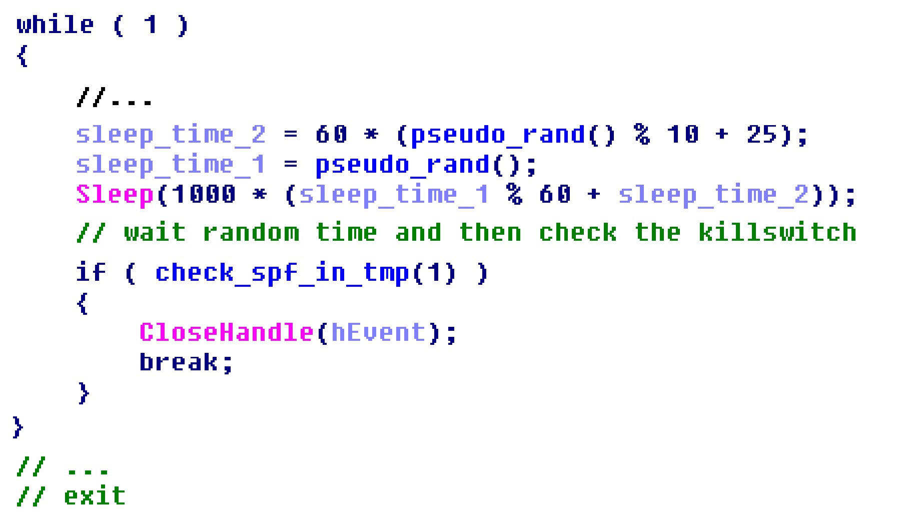
The attackers found a nice way to hide the loader within the second stage. There are just a few bytes woven into each padding which is commonly put in place by a compiler between function code. The following figure shows the difference between the original file by Symantec (left) and the weaponized one (right):



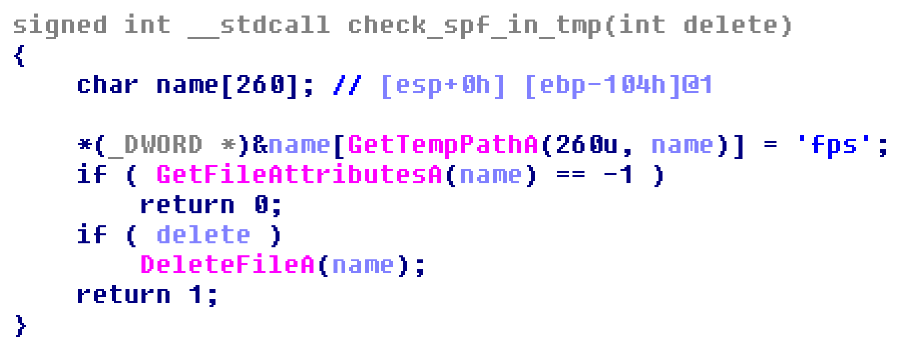
The (not so good) kill switch

Many of you may have heard about the WannaCry ransomware and its weak spot called the kill switch. The good news in our case is that one of the payloads delivered by the backdoored CCleaner also contains a code mechanism similar to a kill switch. The bad news is that it is not as powerful as a kill switch in a WannaCry attack, i.e. no silver bullet.

The most important outcome of the analysis is definitely the discovery of a kill switch. More details about this finding: The second stage payload checks for the presence of a file %TEMP%\spf. If the file exists, the payload will terminate.



```
while ( 1 )
{
    //...
    sleep_time_2 = 60 * (pseudo_rand() % 10 + 25);
    sleep_time_1 = pseudo_rand();
    Sleep(1000 * (sleep_time_1 % 60 + sleep_time_2));
    // wait random time and then check the killswitch
    if ( check_spf_in_tmp(1) )
    {
        CloseHandle(hEvent);
        break;
    }
}
// ...
// exit
```



```
signed int __stdcall check_spf_in_tmp(int delete)
{
    char name[260]; // [esp+0h] [ebp-104h]@1

    *(_DWORD *)&name[GetTempPathA(260u, name)] = 'fps';
    if ( GetFileAttributesA(name) == -1 )
        return 0;
    if ( delete )
        DeleteFileA(name);
    return 1;
}
```

As you may notice from the code above, this payload is running in an endless loop. Within this loop, the payload tries to communicate with one of its CnC servers. The previously described kill-switch can be used to exit the loop and thus the whole program. In other words, this will prevent any other connections to the CnC server. Sadly, the kill-switch is checked after a communication attempt is made, so if the server responded, the user has already received and ran the second stage payload which renders the kill-switch almost useless.

Getting to Stage Three

Similar to the first stage payload, the second stage also relies on communication with CnC servers. However, there is no hardcoded IP address nor any DGA (domain generation algorithm) like there was in the former payload. Instead, there are three different approaches to retrieve the CnC IP address and one of them is picked randomly each time by the algorithm. The three approaches are:

1. Via a hidden message stored in user profile details on a GitHub page (this doesn't exist anymore; it was probably deleted after the attackers realized something was going on). The URL string is parsed by payload in the reply from GitHub and by using simple binary operations. The result is an IP address of another CnC, which the payload uses for communication over TCP port 443 (usually HTTPS). In other code parts, there is also a UDP communication over port 53, which mimics DNS protocol.

2. Alternatively to approach 1, it also tries to retrieve an IP address from a WordPress-hosted page. Once again, this web page is not active at the moment so the payload is unable to retrieve any information from it.

3. Finally, there is a third way to get a CnC IP address and it is very similar to the approach used in the DGA of the first payload. It tries to read DNS records for a domain "**get.adxxxxxx.net**" (exact domain name redacted). It requires at least two IP addresses from which it computes the target CnC address.

To illustrate the algorithm, we demonstrate how it would work for the avast.com domain and its two IP addresses 77.234.43.52 and 77.234.45.78.

At first the IP addresses are represented as hexadecimal numbers:

77.234.43.52 = 4D EA 2B 34
77.234.45.78 = 4D EA 2D 4E

Then the bytes of these numbers are XORed together:

4D EA 2B 34 = /xor/ = C1 79
4D EA 2D 4E = /xor/ = C7 03

These four bytes are grouped together forming four octets of the CnC address C1 79 C7 03. (Note: we omit displaying the resulting IP address because this was only an illustrative example using the avast.com domain). In the case of the **get.adxxxxx.net** domain, there are no IP addresses registered to this domain right now, thus the algorithm is once again not working and no CnC IP address is served at the time of writing this article.

As you can see, none of these three methods is able to retrieve the CnC IP address at this moment. If it could retrieve the CnC IP address, however, it would start a bidirectional socket-based communication with the CnC. It will once again upload some information (computer name, volume serial number of system drive, installed OS, etc.) about the victims of the attack. More importantly this second stage payload is capable of retrieving and executing additional code from the CnC - the 3rd stage payload.

Next steps

Our investigation and hunt for the perpetrators continues. In the meantime, we advise users who downloaded the affected version to upgrade to the latest version of CCleaner and perform a scan of their computer with a good security software, to ensure no other threats are lurking on their PC.

The companies who we believe were introduced to the second stage payload were notified. If there are any other companies who believe they encountered this malware, please contact us through our legal department at legal@avast.com.