Deep Analysis of New Poison Ivy/PlugX Variant - Part II

blog.fortinet.com/2017/09/15/deep-analysis-of-new-poison-ivy-plugx-variant-part-ii

September 15, 2017

Threat Research

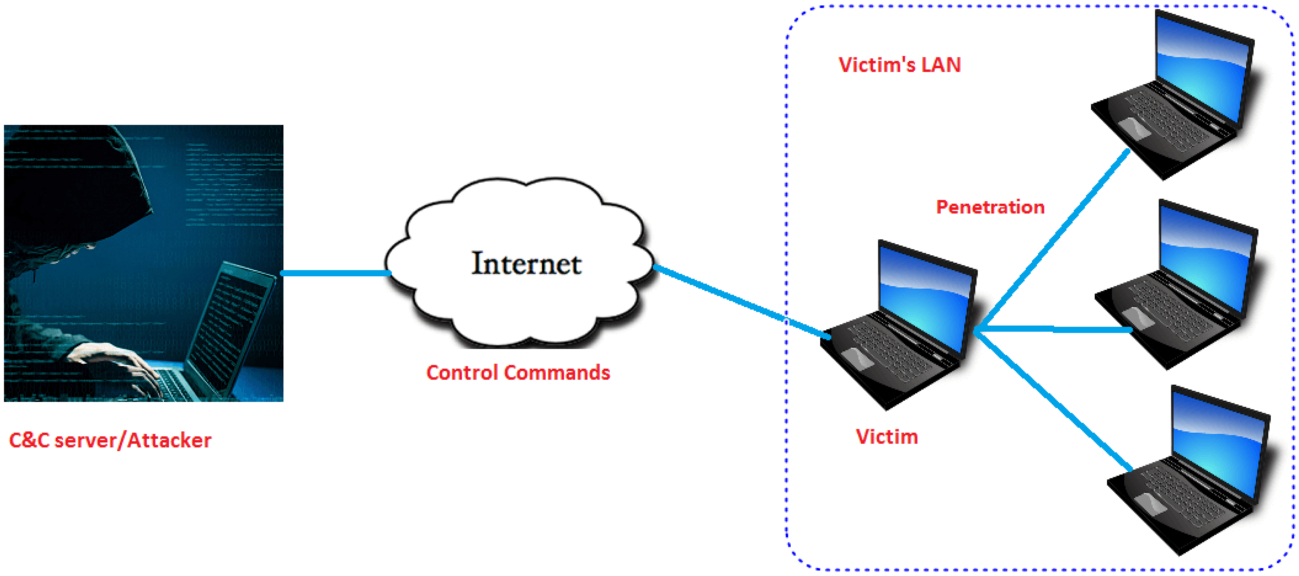
By Xiaopeng Zhang | September 15, 2017

Background

This is the second part of the FortiGuard Labs analysis of the new Poison Ivy variant, or PlugX, which was an integrated part of Poison Ivy's code. In the first part of this analysis we introduced how this malware was installed onto victim's systems, the techniques it used to perform anti-analysis, how it obtained the C&C server's IP&Port from the PasteBin website, and how it communicated with its C&C server.

What we didn't talk much about in that first blog was the control-commands that are used by this malware, partly because only a few of those commands were used during our analysis. However, as you may know, RAT malware usually has many control-commands so that attackers can effectively remotely control a victim's machine.

So, after our initial analysis, we monitored the C&C servers and captured their packets. Fortunately, we were able to successfully collect enough attacks and packets so that we could obverse and document its behavior. In this analysis, I'm going to focus on the control-commands used by the C&C server as it attempts to penetrate the victim's network by exploiting vulnerabilities.



Although the C&C servers have now been shut down, we found a way to decrypt the communication data from the captured packets in order to analyze its behavior.

As per my analysis, this variant of Poison Ivy eventually launches the MS17-010 (Eternal Blue) attack against the machines located inside the victim's LAN. Let's now take a look at how it performs this exploit.

Manage multiple modules

Before going on, however, we have to talk about how the decrypted modules are managed. From Part I we know that there are six modules in the svchost.exe program, which are connected by a doubly linked list. There is a module node in each of modules, as well as in svchost.exe. The module node is added into the doubly linked list when its module code is initialized. The header of the doubly linked list is in a global variable located in svchost.exe's memory space (qword_2345D0 with base address 0x220000 in my case). Below is a module node's structure, along with some corrections to the one shown in the Part I of this analysis.

```
Offset  Size  Description
+00H  8 bytes pointer to next object in the list
+08H  8 bytes pointer to previous object in the list
+10H  4 bytes a flag that tells if the module being used
+14H  4 bytes a constant 0x1B1844DF
+18H  4 bytes module's index
+28H  8 bytes the base address of the module
+30H  8 bytes pointer to export function table
```

The first module (which was injected into svchost.exe when svchost.exe started) is executed in svchost.exe, and was the first one added into the doubly linked list. I call it the host module.

I named these module1, module2, etc. according to the order in which they are added into the doubly linked list, The six modules are decrypted by the host module.

Figure 1 shows a view of the module node of the host (svchost.exe) in memory.

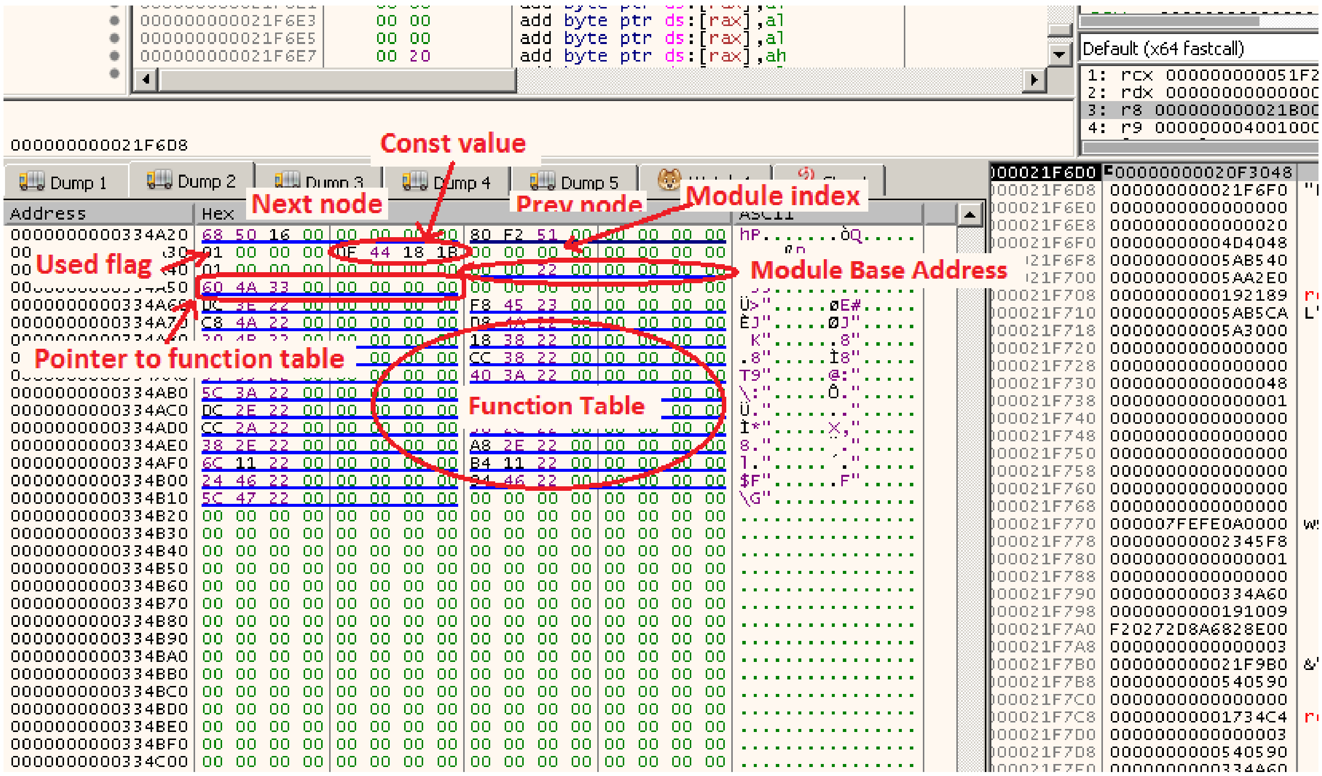


Figure 1. View of the host module node in memory

The host module node's address is 0x334A20. The previous node's address is 0x165068, and the next one is 0x51F280. The host module's index is 0, and its module base address is 0x220000. Finally, the function table's address is 0x334A60. Module index is important because it is also a part of the Control-Commands. We will talk more about this later.

Several functions in the host module are used to manage this doubly linked list. To manage the doubly linked list between these different modules, the author of the malware designed a named sharing memory (by calling API *CreateFilemappingA*) where the addresses of the manager functions are saved. So whenever it wants to manage the doubly linked list, it only needs to access all these functions from the sharing memory. BTW, the name of this sharing memory is created by calculating two current process IDs (by calling API *GetCurrentProcessID*, i.e. svchost.exe PID).

In Figure 2, you can see how the named sharing memory is created, and where the manager functions are saved in the sharing memory. The functions in [rax+8] and [rax+18] are called frequently during handling C&C commands. [rax+18] is the function that gets the module node from the doubly linked list using the module index, and sets module flag. [rax+8] is used to restore the module flag.

```
                mov     [rsp+arg_8], rbx
                push    rdi
                sub     rsp, 90h
                call    cs:call_GetCurrentProcessId ; GetCurrentProcessId
                mov     edi, eax
                xor     edi, 0FFFFFFFCh
                call    cs:call_GetCurrentProcessId ; GetCurrentProcessId
                lea     rdx, unk_2254D8 ; ;; "%p%p"
                mov     ebx, eax
                lea     rcx, [rsp+98h+var_68]
                mov     r8d, 1EA441Dh
                xor     ebx, 40A0668h
                call    Decrypt_String_fun
                mov     rcx, rax
                call    sub_221000      ; ;WideCharToMultiByte
                lea     rcx, [rsp+98h+var_48] ; ;;target buf
                mov     r9d, edi
                mov     rdx, rax        ; "%p%p"
                mov     r8d, ebx
                call    cs:call_wsprintfA ; wsprintfA(target ecx, fmt_str edx, %p 1, %p 2)  %p address
                lea     rcx, [rsp+98h+var_68]
                call    sub_224C54
                mov     ebx, 28h
                lea     r11, [rsp+98h+var_48] ;"000000000040A0BAC00000000FFFFF238" as CreateFileMappingA name
                mov     [rsp+98h+var_70], r11
                lea     r8d, [rbx-24h]
                xor     r9d, r9d
                xor     edx, edx
                or      rcx, 0FFFFFFFFFFFFFFFFh
                mov     dword ptr [rsp+98h+var_78], ebx
                call    cs:call_CreateFilemappingA
                test    rax, rax
                jnz     short loc_222A5A

                ●●●

loc_222A5A:                             ; CODE XREF: sub_2229C4+8C↑j
                xor     r9d, r9d
                xor     r8d, r8d
                mov     rcx, rax
                lea     edx, [r9+2]
                mov     [rsp+98h+var_78], rbx
                call    cs:call_MapViewOfFile ; MapViewOfFile
                test    rax, rax
                jz      short loc_222A52 ;
                lea     rcx, sub_222F20 ; ; add module node into doubly linked list.
                mov     [rax], rcx
                lea     rcx, sub_223004 ; ;;; restore module used flag
                mov     [rax+8], rcx
                lea     rcx, sub_223180
                mov     [rax+10h], rcx
                lea     rcx, sub_223348 ; ;;;get module node from doubly linked list by moudle index.
                mov     [rax+18h], rcx
                lea     rcx, sub_2234A0
                mov     [rax+20h], rcx
                mov     rcx, rax
                call    cs:call_UnmapViewOfFile
                xor     eax, eax

loc_222AB8:                             ; CODE XREF: sub_2229C4+94↑j
                mov     rbx, [rsp+98h+arg_0]
                add     rsp, 90h
                pop     rdi
                retn
sub_2229C4      endp
```

It saves 5 functions of managing doubly linked list into named sharing memory.

Figure 2. Code snippet of adding management functions into the named sharing memory

Here is the modules' information in my test environment:

Name	Base address	Size	Module index
Host	0x220000	0x11E000	0x00
Module1	0x160000	0x00F000	0x01
Module2	0x170000	0x011000	0x02
Module3	0x190000	0x010000	0x03
Module4	0x4D0000	0x00E000	0x04
Module5	0x4E0000	0x00E000	0x10
Module6	0x4F0000	0x00F000	0x11

Control-Command Packet Structure

In order to easily understand the C&C packets, I will explain the packet structure here. As I explained in the first blog, the packet payload is encrypted. Through analyzing its decryption function, I was able to write a python function to decrypt the data. This is the same function that the host module used to decrypt those six modules, as well as the C&C server IP&Port from the PasteBin website, but different decryption keys are used.

Python decryption function:

```
def decrypt_fun(buf, size, key):
    target = []
    key1 = key
    key2 = key
    for cnt in range(size):
        key1 *= 0x13379c8
        key2 *= 0x13
        key1 ^= 0x5397fc2
        key2 -= 0x17
        cl = (key1&0xff)
        cl -= (key2&0xff)
        val = ((cl) ^ ord(buf[cnt]))&0xff
        target.append((val))

    return "".join(map(chr, target))
```

The decrypted packet consists of two parts. The first 14H bytes are the header, and the data starts at offset 14H. The packet structure looks like this:

Offset	Size	Description
+00H	4 bytes	Decryption key
+04H	4 bytes	Control-Command
+08H	4 bytes	Sub-command, data depends on control-command
+0CH	4 bytes	the size of data part
+10H	4 bytes	
+14H	variable	the data part starts here

In the first blog I introduced commands "030001" and "030003". Please refer here for more details. By the way, the malware uses big-endian byte order to save its data. The control command is a Dword value, whose high 16 bits are the module index, and the low 16 bits is a kind of code branch switch. Once the malware gets the command it retrieves the module node from the doubly linked list by matching the module index. It then calls the functions of that module to handle this command data.

```
loc_192313:                             ; CODE XREF: sub_1921E8+117↑j
                mov     rcx, [rbp+arg_10] ; ;;socket
                lea     rdx, [rbp+arg_8]
                call    sub_193370      ; ;It calls recv to receive C&C server data. It then decrypts it.
                mov     edi, eax
                test    eax, eax
                jnz     loc_1923C1
                mov     rax, [rbp+arg_8] ; It holds the decrypted data's address.
                mov     ecx, [rax+4]    ; It gets control command.
                call    cs:call_htonl
                mov     ebx, eax
                shr     ebx, 10h        ; It gets high 16bits as the module index of the control command.
                                        ; ;
                call    sub_191C44      ; It retrieves the linked list management functions from named sharing memory
                                        ; ;
                mov     ecx, ebx        ; ebx is the command's high 16 bit. it's the module's index.
                call    qword ptr [rax+18h] ; It obtains the module node from doubly linked list by its index.
                mov     rbx, rax
                test    rax, rax
                jz      loc_1923FD
                mov     r8, [rax+30h]   ; at module node offset 30H saves the address of function table.
                test    r8, r8
                jz      loc_1923FD
                cmp     qword ptr [r8], 0
                jz      loc_1923FD
                mov     rdx, [rbp+arg_8]
                lea     rcx, [rbp+arg_10]
                call    qword ptr [r8]  ; Going to different code branch according to module index.
                mov     edi, eax
                call    sub_191C44      ; Get the linked list management functions from named sharing memory.
                mov     rcx, rbx
                call    qword ptr [rax+8] ; restore module node used flag.
                cmp     edi, 0FFFFFFFFh
                jnz     short loc_1923F2
                mov     ecx, 7Fh
```

Figure 3. All packets from C&C server are dispatched from here

Figure 3 shows the code snippet used for dispatching the C&C packets to the correct module for processing. After “call sub_193370” we got the decrypted C&C server packet in [rbp+arg_8]. “call sub_191C44” is used to get the management functions in rax from the named sharing memory. “call qword ptr [rax+18h]” is used to call one management function to get the module node from the doubly linked list using the module index in rcx i.e. high 16 bits of command. “call qword ptr [r8]” calls the first function of the function table to process the received packet.

From the above analysis you should now be able to clearly see the entire process of how the malware processes the C&C server’s packets.

Installing the “00000025” module

In my captured traffic, I was able to see many control commands. They include “00030001”, “00030002”, “00030003”, “00030004”, “00000003”, “00000001”, “00250000”, etc.

So let's now take a look at what the "00000003" command is used for. Figure 4 shows the original received packet and the decrypted data.

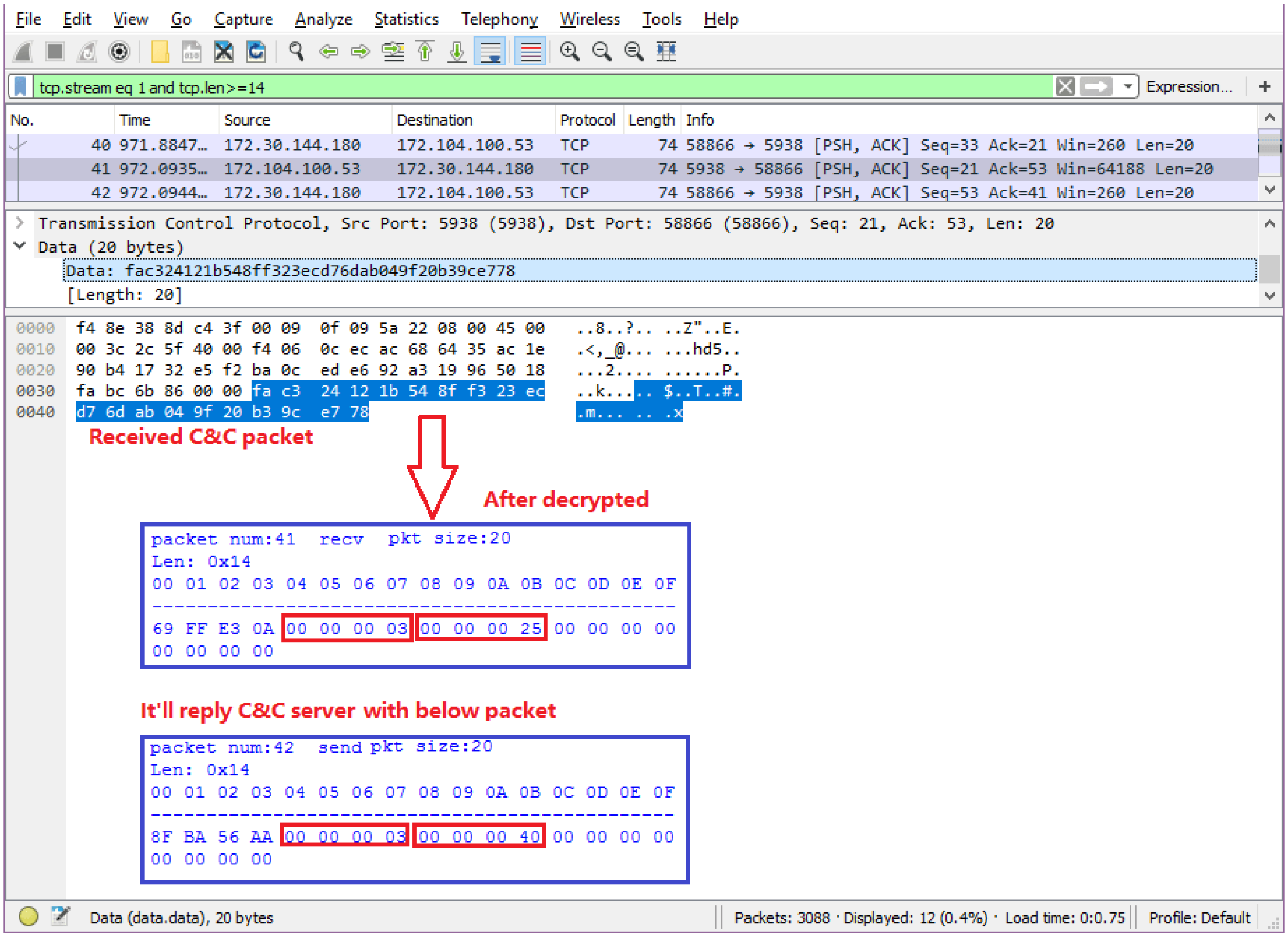


Figure 4. "00000003" command data

From the command "00000003" details we know that this packet is going to be passed to the host module (its index is 0), and then be processed by the first function in the function table and the "0003" branch.

It gets the sub-command ("00000025") as the module index to look for in that doubly linked list. So far, no module's index is 0x25. It then replies to the C&C server with sub-command "00000040". If the 0x25 module node exists, the sub-command is "00000000".

The C&C server then sends back command "00000001" with a new module attached. Below is part data of this packet after decryption, where you can see that the sub-command is "00000025". In code branch "0001" it decompresses the received module, then gets its code initialized, and finally adds it into the doubly linked list. This module's index is 0x25, so I call it Module25.

```
Len: 0x420
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----------------------------------------------
29 72 ED 38 00 00 00 01 00 00 00 25 00 00 08 DE
00 00 12 00 47 DE 08 00 00 00 12 00 00 00 20 23
C0 87 D3 0F 93 24 0A 3A EF 00 50 00 00 00 04 00
80 01 04 00 03 00 20 21 00 54 00 10 00 00 0B 02
00 00 04 02 00 AB ED 1F 80 01 01 03 40 08 01 00
20 01 00 0C 01 00 42 00 30 01 00 06 00 40 01 00
02 01 21 00 01 00 00 00 FF 00 00 FF 00 00 FF 00
00 9C 40 53 48 83 EC 20 48 8B D9 85 00 02 88 80
.....
```

It later sends command “00000001” with sub-command “00000000” to the C&C server to let it know that the 0x25 module was installed successfully. This module will be used to penetrate the victim’s network.

BTW, this module’s information in my test machine is:

```
Name       Base address    Size     Module index
Module25   0x20f0000       0xD000   0x25
```

Penetrating the victim’s LAN using EternalBlue

I’m sure that the C&C server sent commands to get the victim’s network configuration (my local IP, Gateway, DNS server), though I did not catch them.

Figure 5 is the screenshot of the network configuration of my test machine.

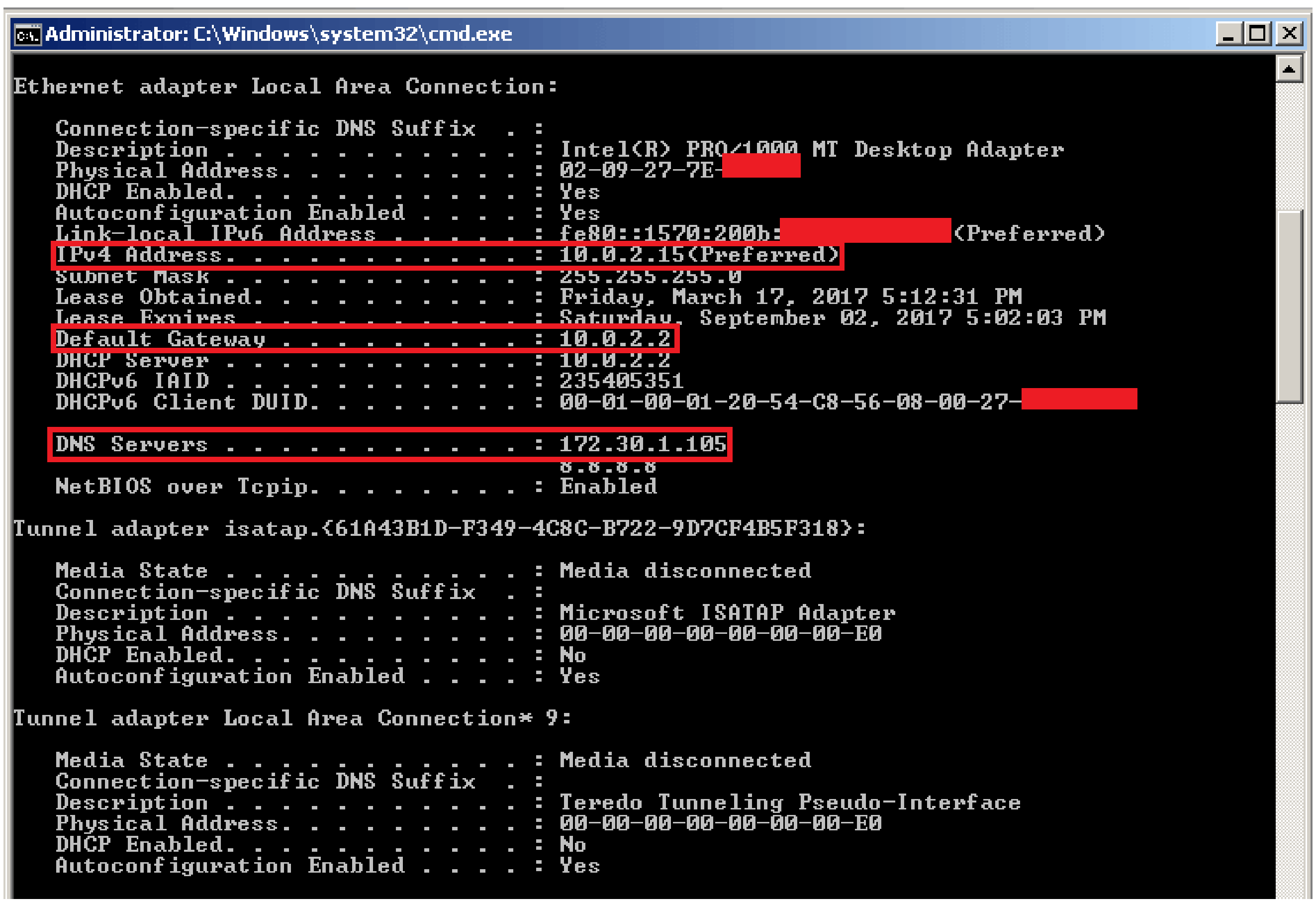


Figure 5. Network information

The C&C server controls the malware to scan the victim's network segment, including local IP, Gateway, and DNS server. For example, because my DNS server is 172.30.1.105 it's going to scan the 172.30.1.105/24 network segment.

The C&C server sends the "00000025" command with the destination IP and Port for further attack. By decrypting "00000025" packets we are able to see its data, shown below.

```
packet num:1194  recv pkt size:40
Len: 0x28
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
------------------------------------------------
7B A5 F7 94 00 25 00 00 00 00 01 BD 00 00 00 14   {?÷? %     ?
00 00 00 0D 45 14 0D 00 00 00 80 31 37 32 2E 33       E     ?172.3
30 2E 31 2E 31 32 35 00                           0.1.125

packet num:1195  recv pkt size:40
Len: 0x28
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
------------------------------------------------
EA F6 12 82 00 25 00 00 00 00 01 BD 00 00 00 14   ê? ? %     ?
00 00 00 0D 45 14 0D 00 00 00 80 31 37 32 2E 33       E     ?172.3
30 2E 31 2E 31 32 32 00                           0.1.122

packet num:1196  recv pkt size:40
Len: 0x28
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
------------------------------------------------
45 50 F3 B8 00 25 00 00 00 00 01 BD 00 00 00 14   EPó? %     ?
00 00 00 0D 45 14 0D 00 00 00 80 31 37 32 2E 33       E     ?172.3
30 2E 31 2E 31 32 33 00                           0.1.123
```

From this data it is easy to see that there are IP addresses from three local machines. The sub-command "000001BD" refers to port 445.

Module25 processes this packet, pulls the IP and port information from the packet, and then makes a connection to it. If any error occurs, it sends the status to the C&C server.

Once successfully connected to the destination machine, the malware then serves as a middleman that keeps transferring the two sockets' data between the C&C server and the destination machine (like man-in-the-middle does). In module3 we also see its debug output strings "*SoTransfer(%p<=>%p)...\r\n*" and "*SoTransfer(%p<=>%p) quit!\r\n*". Figure 6 and 7 show the attack view in Wireshark.

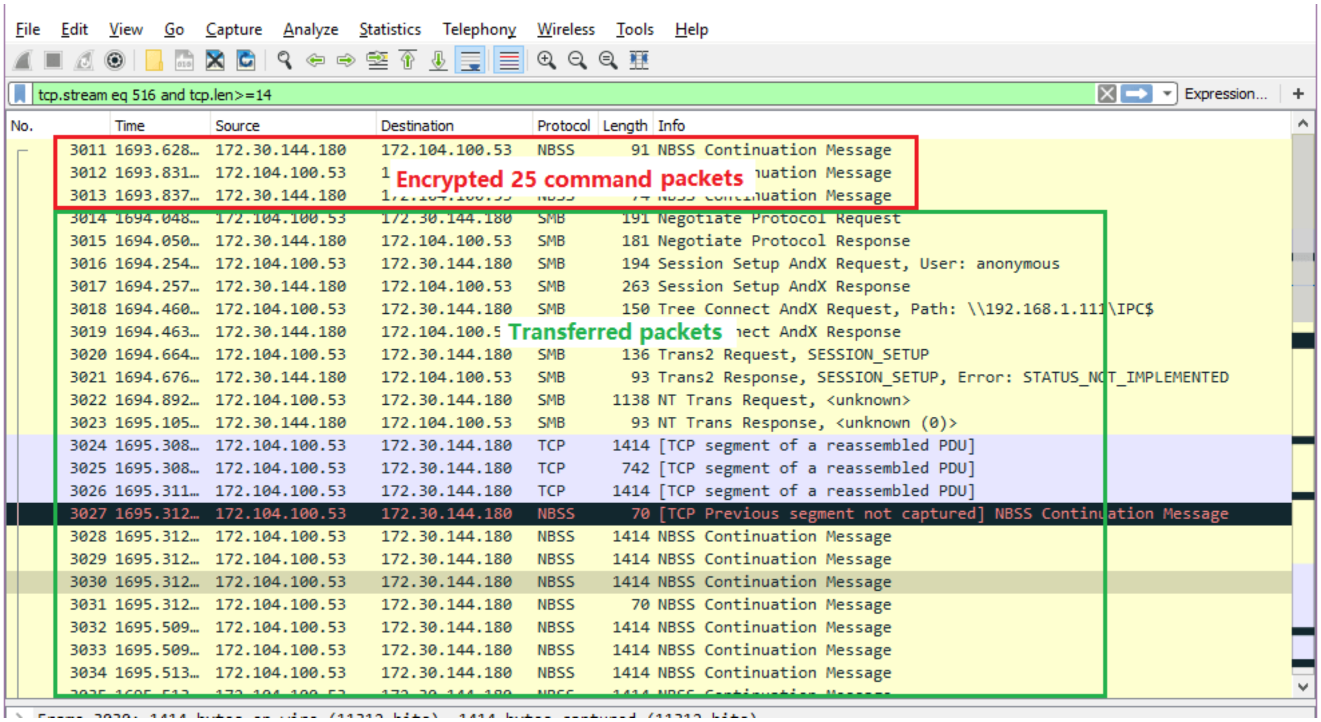


Figure 6. EternalBlue attack packets

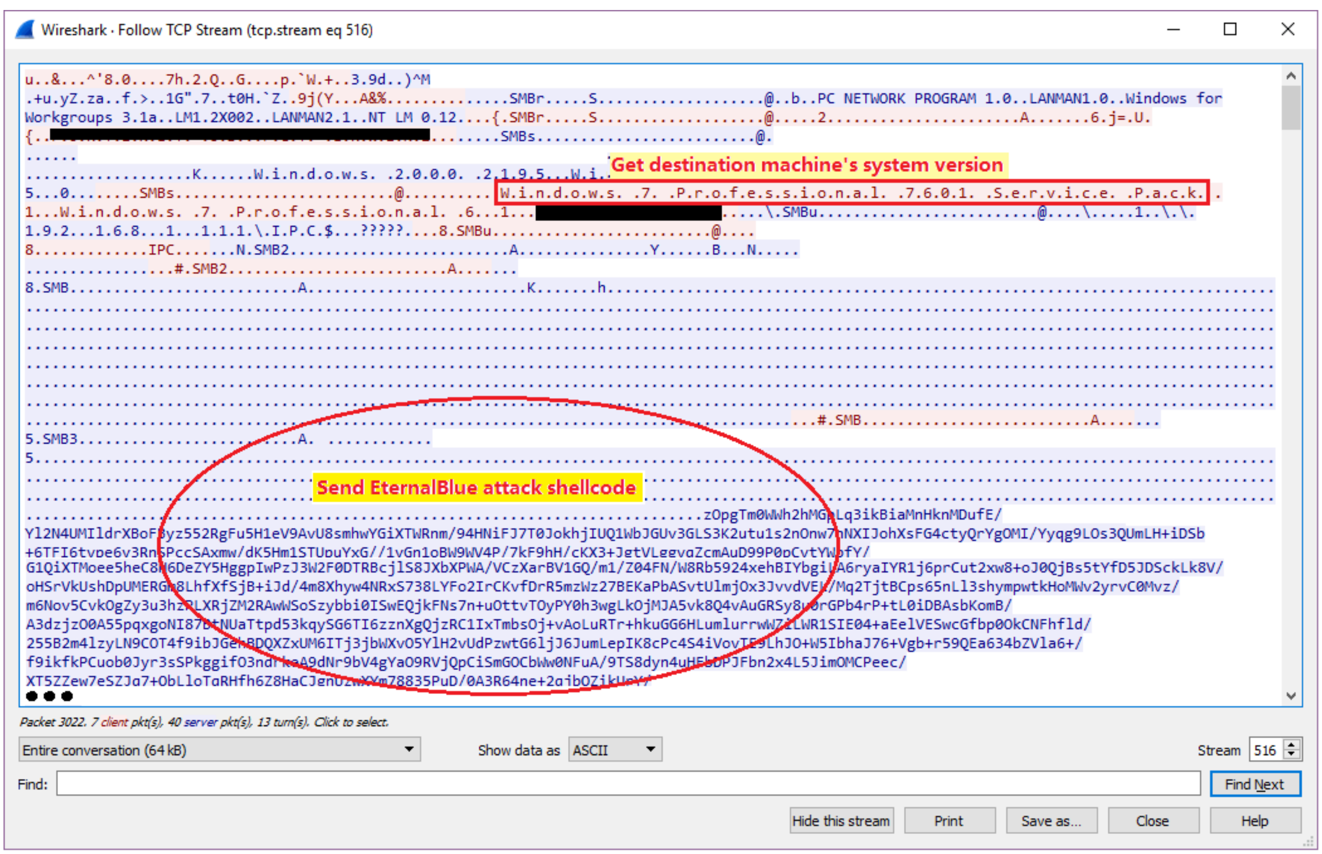


Figure 7. EternalBlue attack packet payload

Module25 makes the connection to the destination IP and then calls module3's function to perform the transfer work by calling the recv() and send() functions. In module3 function *sub_1935A8* it creates two threads to do that. One thread receives data from the C&C socket and sends it to the destination machine, and another one receives data from the destination machine and forwards it to the C&C server. Figure 8 shows the code snippet for what I explained about the two threads.

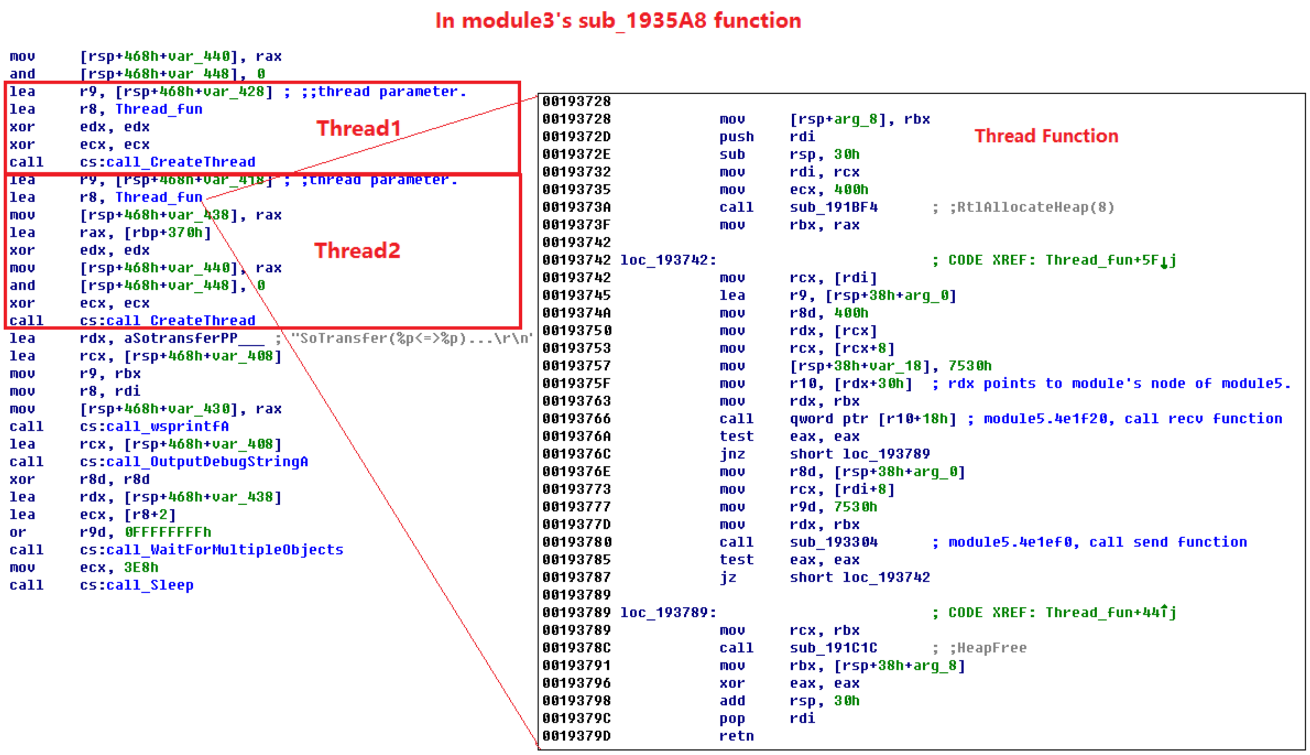


Figure 8. Two threads to transfer packets

Conclusion

Based on our analysis, this new Poison Ivy variant takes advantage of the EternalBlue exploit to spread. Once one system is infected by this variant, other systems on the same network are likely to be infected by the compromised system.

Solution

Users should apply Microsoft's patch for MS17-010.

Fortinet IPS signature MS.SMB.Server.SMB1.Trans2.Secondary.Handling.Code.Execution was released in March 2017 to protect our customers against the EternalBlue attack.

Sign up for weekly Fortinet FortiGuard Labs Threat Intelligence Briefs and stay on top of the newest emerging threats.

Related Posts