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

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

September 15, 2017



### Threat Research

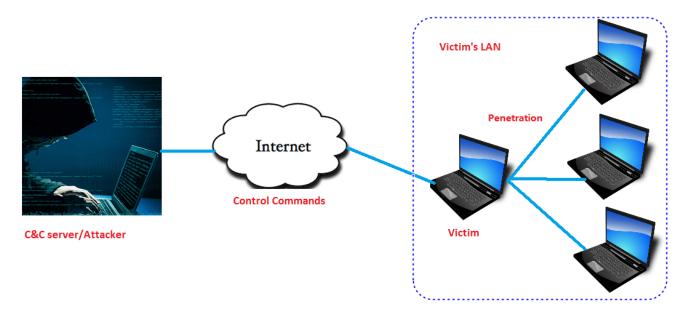
By Xiaopeng Zhang | September 15, 2017

# Background

This is the second part of the <u>FortiGuard Labs</u> analysis of the new Poison Ivy variant, or PlugX, which was an integrated part of Poison Ivy's code. In the <u>first part</u> 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 controlcommands 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 <u>MS17-010</u> (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	t	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.

		_		_			
Liauro 1	ahowo o v	iour of the	modulo podo	of the heat	(avabaat ava)	) in momo	m /
гюше і	Shows a v	iew oi ine	module node	or me nosi i	ISVENOSI EXE	) in memo	IV
						,	• • •

000000000000000000000000000000000000	add byte ptr ds:[rax],al		
00000000021F6E5     00 00     00000000021F6E7     00 20	add byte ptr ds:[rax],al add byte ptr ds:[rax],ah		Default (x64 fastcall)
•			1: rcx 000000000051F2
	J		2: rdx 00000000000000
			3: r8 00000000021B0C
Constant.	_		4: r9 0000000400100C
0000000021F6D8 Const valu	e		
💭 Dump 1 🔛 Dump 2 💷 Dump 3 🔛 Dump 4	💷 Dump 5 🗍 🎯	00021F6D	0 C00000000020F3048
		00021F6D	
Address Hex Next node	Prev node	000021F6E	
000000000334A20 68 50 16 00 00 00 00 80 F			
00		00021F6F	
oo Used flag 40 01 oo oo oo oo oo oo oo oo oo		dress (21F6F)	
0000000000000000000 60 4A 33 00 00 00 00 00 00 00		12IF/0	
000000000334A68 UC 3E 22 00100 00 00 00 F8 4	IS 23 00 00 00 00 00 Ü>"øE#	000021F70	
	<u>A 22 38 00 00 00 00</u> ÈJ"ØJ"	000021F71	
	18 22 00 00 00 00 00 K"8"	000021F72	
	<u>8 22 00 00 00 00 00</u> .8"Ì8"	000021F72	
	A 22 00 00 00 00 N T9"@:"	000021F73	
000000000334AB0 5C 3A 22 00 00 00 00 00	stien Table 00 00 \;"0."	000021F73	
	ction Table oo of U."	000021F74	
000000000334AD0 CC 2A 22 00 00 00 00 00 00	<u>, 22 00 00 00 00 01</u> 1*"	000021F74	
	E 22 00 00 00 00 00 8."	00021F75	000000000000000000000000000000000000000
0000000000334AF0 6C 11 22 00 00 00 00 00 B4 1 0000000000334B00 24 46 22 00 00 00 00 00 94 4	1 22 00 00 00 00 00 1	000021F75	B 0000000000000000
0000000000334B00 24 46 22 00 00 00 00 00 00 00 00 00 00 00 00		000021F76	
000000000334B20 00 00 00 00 00 00 00 00 00 00 00 00 0		00021F76	
		000021F77	
		000021F77	
		000021F78	
		000021F78	
000000000334870 00 00 00 00 00 00 00 00 00 00		000021F79	
000000000334880 00 00 00 00 00 00 00 00 00 00	0 00 00 00 00 00 00	000021F7A	
000000000334890 00 00 00 00 00 00 00 00 00 00 00		000021F7A	
000000000334BA0 00 00 00 00 00 00 00 00 00 00 00		000021F7B	
000000000334BB0 00 00 00 00 00 00 00 00 00 00 00		000021F7B	
	0 00 00 00 00 00 00	000021F7C	
	0 00 00 00 00 00 00	000021F7C	
000000000334BE0 00 00 00 00 00 00 00 00 00 00 00 00 0		000021F7D	
000000000334BF0 00 00 00 00 00 00 00 00 00 00 00 00 0		000021F7D	B 000000000540590
000000000334000 00 00 00 00 00 00 00 00 00 00		h00021E7E	000000000334660

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_0], rbx rdi
	push	
	sub call	rsp, 90h cs:call GetCurrentProcessId ; GetCurrentProcessIN
	moy	edi, eax
	xo	edi, OFFFFFFCh
	call	cs:call GetCurrentProcessId ; GetCurrentProcessId
	lea	rdx, unk 2254D8 ; ;; "%p%p"
	mov	ebx, eax
	lea	rcx, [rsp+98h+var_68]
	mov	r8d, 1EA4410h
	xor	ebx, 40A0668h
	call	Decrypt String fun
	mov	rcx, rax
	<mark>call</mark>	<pre>sub_221000 ; ;WideCharToMultiByte</pre>
	lea	rcx, [rsp+98h+var_48] ; ;;target buf
	mov	r9d, edi
	mov	rdx, rax ; "%p%p"
	mov	r8d, ebx
	<mark>call</mark>	cs:call_wsprintfA ; wsprintfA(target_ <u>ecx.fmt_ctr_edx, %p_1_%p_2)</u> %p address
	lea	rcx, [rsp+98h+var_68]
	<mark>call</mark>	sub_224054
	mov	ebx, 28h
	lea	r11, [rsp+98h+var 48] ;"00000000040A0BAC0000000FFFFF238" as CreateFileMappingA name
	mov	[rsp+98h+var_70], r11
	lea	r8d, [rbx-24h]
	xor	r9d, r9d
	xor	edx, edx
	0r	rcx, OFFFFFFFFFFFFFFF
	mov <mark>call</mark>	dword ptr [rsp+98h+var_78], ebx cs:call CreateFilemappingA
	test	rax, rax
	jnz	short loc 222A5A
	1	5161 € 100_222.051
		•
1oc_222A5A:		; CODE XREF: sub_2229C4+8Cīj
	xor	r9d, r9d
	xor	r8d, r8d
	mov los	rcx, rax
	lea mov	edx, [r9+2] [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 rcx, sub 223468 It saves 5 functions of managing doubly
	lea	rcx, sub_2234A0 It saves 5 functions of managing doubly
	mov	[rax+29h], rcx
	MOV	rcx, rax linked list into named sharing memory.
	<mark>call</mark>	cs:call_UnmapViewOfFile
	xor	eax, eax
100 000000		• CODE VDEE• cub 222000+00∲÷
loc_222AB8:	mou	; CODE XREF: sub_2229C4+941j
	mov add	rbx, [rsp+98h+arg_0] rsp, 90h
	pop	rsp, 900 rdi
	retn	- 02
sub 2229C4	endp	
	cuab	

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 <u>here</u> 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+arq 10] ; ;;socket
	lea	rdx, [rbp+arg 8]
	call	sub 193370 ; ;It calls recv to receive C&C server data. It then decrupts it.
	mov	edi, eax
	test	eax, eax
	inz	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	edı, eax
	call	<pre>sub_191C44 ; Get the linked list management functions from named sharing memory.</pre>
	MOV	rcx, rbx
	call	qword ptr [rax+8] ; restore module node used flag.
	cmp	edi, ØFFFFFFFh
	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", "0000003", "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.

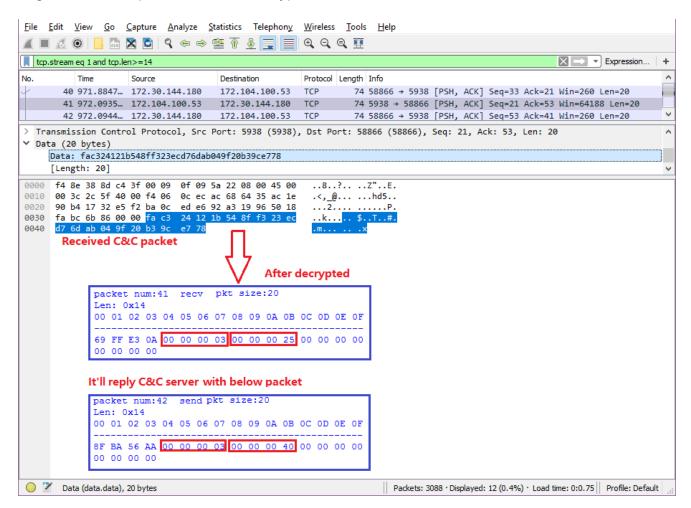


Figure 4. "00000003" command data

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.

Ler	n: 0	9x42	20												
00	01	02	03	04	05	06	07	08	09	ØA	ØB	ØC	ØD	ØE	ØF
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
CØ	87	D3	ØF	93	24	ØA	ЗA	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	8

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.

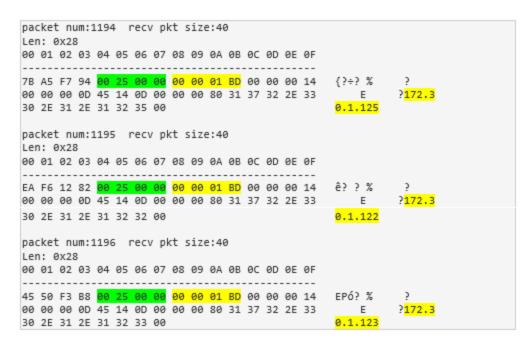
Figure 5 is the screenshot of the network configuration of my test machine.

```
_ 🗆 ×
🖼 Administrator: C:\Windows\system32\cmd.exe
                                                                                                                                      ٠
Ethernet adapter Local Area Connection:
    Connection-specific DNS Suffix
                                                         Intel(R) PRO/1000 MT Desktop Adapter
02-09-27-7E-
    Description
Physical Address....
DHCP Enabled....
Autoconfiguration Enable
Link-local IPv6 Address
                                                        02-0.
Yes
Yes
fe80::1570:200b
10-0.2.15(Prefer
                              Enabled
                                                                                                   (Preferred)
                                                   . : 10.0.2.15(Preferred)
    IPv4 Address. . . . .
                                                      : 255.255.255.0
: Friday, March 17, 2017 5:12:31 PM
: Saturday, September 02, 2017 5:02:03 PM
: 10.0.2.2
    Subnet Mask . .
Lease Obtained.
Lease Exnires .
    Default Gateway .
    DHCP Server . . . .
DHCPv6 IAID . . . .
DHCPv6 Client DUID.
                                                         10.0.2.2
235405351
00-01-00-01-20-54-C8-56-08-00-27-
                                                      : 172.30.1.105
    DNS Servers . . . . . .
                                                         8.8.8.8
Enabled
    NetBIOS over Topip. . . . . . .
Tunnel adapter isatap.{61A43B1D-F349-4C8C-B722-9D7CF4B5F<u>318</u>}:
    Media State .
                                                         Media disconnected
    Connection-specific DNS Suffix
    Description . . .
Physical Address.
DHCP Enabled. . .
                                                         Microsoft ISATAP Adapter
00-00-00-00-00-00-00-00-E0
                                                         No
Yes
    Autoconfiguration Enabled .
Tunnel adapter Local Area Connection* 9:
    Media State
                                                         Media disconnected
    Connection-specific DNS Suffix
    Description . . .
Physical Address.
DHCP Enabled. . .
                                                         Teredo Tunneling Pseudo-Interface
                                 .
                                                         00-00-00-00-00-00-00-E0
                                                         No
Yes
    Autoconfiguration Enabled
```

#### 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.



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.

	I 🖉 🛞 📙 🛅 🗙 🖾 I	🔍 🗢 🗢 🕾 🗿 🛃 📃	$\Theta$ $\Theta$	Q. 🎹			
tcp	o.stream eq 516 and tcp.len>=14	4				×	Expression
	Time Source	Destination	Protocol	Length	Info		
	3011 1693.628 172.30	0.144.180 172.104.100.53			NBSS Continuation Message		
	3012 1693.831 172.10	04.100.53 1 Encrypted 1	25 com	mand	packets nuation Message		
	3013 1693.837 172.30	0.144.180 1.2.107.100.55	23 COIII		noos concinuation Message		
	3014 1694.048 1/2.10	04.100.53 1/2.30.144.180	SMB	191	Negotiate Protocol Request		
	3015 1694.050 172.30	0.144.180 172.104.100.53	SMB	181	Negotiate Protocol Response		
	3016 1694.254 172.10	04.100.53 172.30.144.180	SMB	194	Session Setup AndX Request,	User: anonymous	
	3017 1694.257 172.30	0.144.180 172.104.100.53	SMB	263	Session Setup AndX Response		
	3018 1694.460 172.10	04.100.53 172.30.144.180	SMB	150	Tree Connect AndX Request, P	ath: \\192.168.1.111	\IPC\$
	3019 1694.463 172.30	0.144.180 172.104.100.5	<b>Fransfe</b> i	rred p	ackets nect AndX Response		
	3020 1694.664 172.10	04.100.53 172.30.144.180	SMB	136	Trans2 Request, SESSION_SETU	IP	
	3021 1694.676 172.30	0.144.180 172.104.100.53	SMB	93	Trans2 Response, SESSION_SET	'UP, Error: STATUS_NO	T_IMPLEMENTED
	3022 1694.892 172.10	04.100.53 172.30.144.180	SMB	1138	NT Trans Request, <unknown></unknown>		
	3023 1695.105 172.30	0.144.180 172.104.100.53	SMB	93	NT Trans Response, <unknown< td=""><td>(0)&gt;</td><td></td></unknown<>	(0)>	
	3024 1695.308 172.10	04.100.53 172.30.144.180	TCP	1414	[TCP segment of a reassemble	ed PDU]	
	3025 1695.308 172.10	04.100.53 172.30.144.180	TCP	742	[TCP segment of a reassemble	ed PDU]	
	3026 1695.311 172.10	04.100.53 172.30.144.180	TCP	1414	[TCP segment of a reassemble	ed PDU]	
	3027 1695.312 172.10	04.100.53 172.30.144.180	NBSS	70	[TCP Previous segment not ca	aptured] NBSS Continu	ation Message
	3028 1695.312 172.10	04.100.53 172.30.144.180	NBSS	1414	NBSS Continuation Message		
	3029 1695.312 172.10	04.100.53 172.30.144.180	NBSS	1414	NBSS Continuation Message		
	3030 1695.312 172.10	04.100.53 172.30.144.180	NBSS	1414	NBSS Continuation Message		
	3031 1695.312 172.10	04.100.53 172.30.144.180	NBSS	70	NBSS Continuation Message		
	3032 1695.509 172.10	04.100.53 172.30.144.180	NBSS	1414	NBSS Continuation Message		
	3033 1695.509 172.10	04.100.53 172.30.144.180	NBSS	1414	NBSS Continuation Message		
	3034 1695.513 172.10	04.100.53 172.30.144.180	NBSS	1414	NBSS Continuation Message		

Figure 6. EternalBlue attack packets

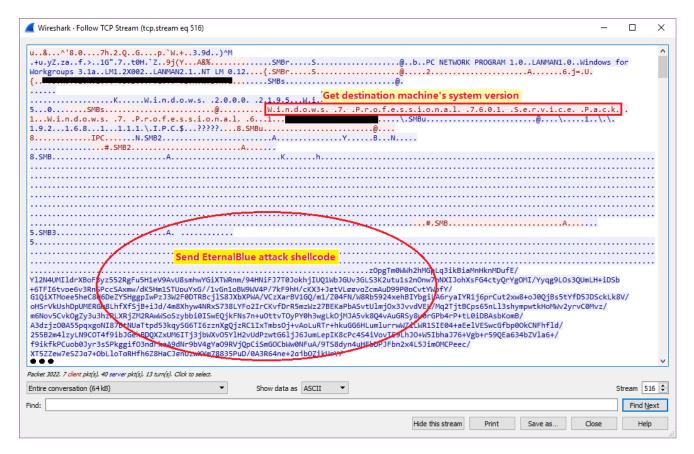


Figure 7. EternalBlue attack packet payload

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.

		-		
mov	[rsp+468h+var_440], rax			
and	[rsp+468h+var 448], 0			
lea	r9, [rsp+468h+var_428] ; ;;thread parameter.	100100700		
lea	r8, Thread fun	00193728		
xor	edx, edx Thread1	00193728	mov	[rsp+arg_8], rbx
xor	ecx, ecx	0019372D	push	rdi Thread Function
call	cs:call CreateThread	0019372E	sub	rsp, 30h
iea	ry, [rsp+408n+var 418r; ;tnread parameter.	00193732	mov	rdi, rcx
lea	r8, Thread fun	00193735	mov	ecx, 400h
mov	[rsp+468h+var 438], rax	0019373A	call	<pre>sub_191BF4 ; ;RtlAllocateHeap(8)</pre>
lea	rax, [rbp+370h]	0019373F	mov	rbx, rax
xor	edx, edx Thread2	00193742		
mov	[rsp+468h+var 440], rax	00193742 loc_19374	2:	; CODE XREF: Thread_fun+5Fij
and	[rsp+468h+var 448], 0	00193742	MOV	rcx, [rdi]
xor	ecx, ecx	00193745	lea	r9, [rsp+38h+arg_0]
call	cs:call CreateThread	0019374A	mov	r8d, 400h
lea	rdx, aSotransferPP ;\"SoTransfer(%p<=>%p)\r\n	00193750	mov	rdx, [rcx]
lea	rcx, [rsp+468h+var 408]	00193753	mov	rcx, [rcx+8]
mov	r9, rbx	00193757	mov	[rsp+38h+var_18], 7530h
mov	r8, rdi	0019375F	mov	<pre>r10, [rdx+30h] ; rdx points to module's node of module5.</pre>
mov	[rsp+468h+var_430], rax	00193763	mov	rdx, rbx
call	cs:call wsprintfA	00193766	call	qword ptr [r10+18h] ; module5.4e1f20, call recv function
lea	rcx, [rsp+468h+var 408]	0019376A	test	eax, eax
call	cs:call OutputDebugStringA	0019376C	jnz	short loc_193789
xor	r8d. r8d	0019376E	mov	r8d, [rsp+38h+arg_0]
lea	rdx, [rsp+468h+var 438]	00193773	mov	rcx, [rdi+8]
lea	ecx, [r8+2]	00193777	mov	r9d, 7530h
or	r9d, OFFFFFFFh	0019377D	mov	rdx, rbx
call	cs:call WaitForMultipleObjects	00193780	call	<pre>sub_193304 ; module5.4e1ef0, call send function</pre>
		00193785	test	eax, eax
mov call	ecx, 3E8h	00193787	jz	short loc 193742
Call	cs:call_Sleep	00193789	-	
		00193789 loc 19378	9:	; CODE XREF: Thread fun+44îj
		00193789	mov	rcx, rbx
		00193780	call	sub_191C1C ; ;HeapFree
		00193791	mov	rbx, [rsp+38h+arg 8]
	$\sim$	00193796	xor	eax, eax
		00193798	add	rsp, 30h
		00193790	рор	rdi
	N	0019379D	retn	
		1		

#### In module3's sub\_1935A8 function

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.

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

#### **Related Posts**

Copyright © 2022 Fortinet, Inc. All Rights Reserved

Terms of ServicesPrivacy Policy | Cookie Settings