Research into the EternalBlue Exploit

Rohan Gorajia
Rutgers University
New Jersey, United States of America
rgorajia01@gmail.com

This literature research paper regards the EternalBlue exploit that occurred on Windows Devices prior to Windows 8. It is suspected to have originated in the NSA and then leaked. This exploit was utilized with the WannaCry ransomware to severely impact and cripple multiple services, including the NHS and telecom companies. It uses an exploit in the Windows implementation of the Server Message Block (SMB) and a set of specific bugs to achieve entry into any other system. The Windows versions earlier than Windows 8 contain an interprocess communication share (IPS$) that allows a null session, enabling the client to send different commands to the server. The main three bugs that EternalBlue exploits are a Wrong Casting Bug, a Wrong Parsing Function Bug, and a Non-paged Pool Allocation Bug. The Microsoft MS17-010 Security Bulletin is what was released with a patch with Windows 8 to mitigate and stop the exploit from existing. Nevertheless, this exploit showed the power of specific exploits and was a significant blemish against the NSA, as well as crippling quite many companies and services worldwide.

I. What is WannaCry and EternalBlue (Problem Statement)

In March of 2017, a new strain of ransomware started appearing and spread rapidly across the world. This ransomware was known as WannaCry (also known as WanaCrypt0r and WCry). It infected tens of thousands of systems in over 74 countries (see Figure 1 [5]). One of the biggest institutions that was hit by the WannaCry ransomware is the National Health Service (NHS) in England. Non-urgent services were cancelled and reverted to backup procedures. Other companies, such as Telefonica in Spain, were also heavily affected. This ransomware could also spread over a network, from one computer to another, to be able to maximize its damage. The spread was on a scale that had not been seen before and ended up netting an estimated $4 billion globally.

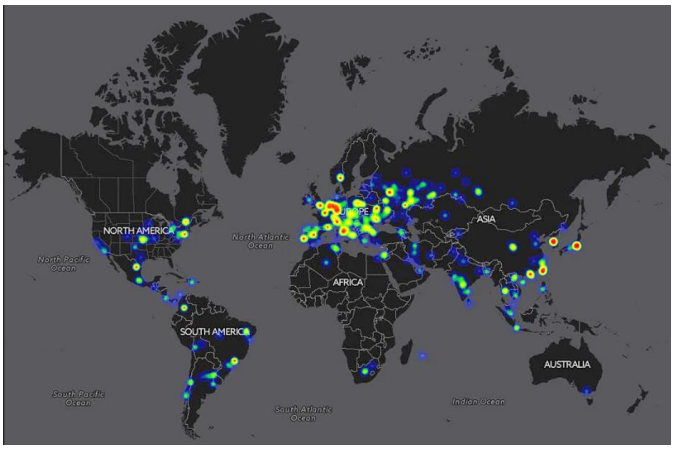

Fig 1. Map of affected countries, provided by MalwareHunterTeam [5]

The WannaCry ransomware utilizes an exploit in all Windows systems prior to Windows 8. This exploit is entitled EternalBlue, and allegedly originated with the NSA. It was released with a large amount of other NSA tools by a hacking group entitled the Shadow Brokers. Microsoft was able to release a patch for the exploit, detailed in security bulletin MS17-010 [4] which was an update sent to all systems running Windows OS. However, due to the scale of some systems, it would take months for them to be able to put the patch through and fully upgrade. [5]

The EternalBlue exploit is part of a family of exploits which include EternalBlue, EternalChampion, EternalRomance, and EternalSynergy. EternalBlue only requires access to the interprocess communication share (IPC$) while the other exploits also must access a certain named pipe as well. [2]

Some of the challenges in creating a solution to the EternalBlue exploit included time sensitivity, reverse engineering, creation of a solution, and distribution. As this exploit was not known by Microsoft before it was released and the WannaCry ransomware started to get installed, Microsoft developers needed to discover how it worked, create a solution, and distribute that solution as fast as possible. This needed to be distributed as fast as possible to attempt to mitigate any further damage and prevent any more attacks. The challenge in creating a solution was in how the WannaCry ransomware worked, as it was very lightweight and able to avoid certain detections. Being able to isolate, reverse engineer, discover a solution, test the solution, implement the solution, and push the patch out to all Windows machines was incredibly time sensitive and yet was able to be completed as soon as possible, and was done so quite well. The papers reviewed determine and specifically focus on the aspect of discovering how the EternalBlue exploit worked and a deep dive into the technical components that allowed the exploit to work.

II. Technical Background

A. *Server Message Block*

The Server Message Block (SMB) is a version of what is known as a Common Internet File System (CIFS). It operates as

a network protocol on the application-layer and is used to provide shared access and communication between different nodes on a network. [1]

The SMB Version 1.0 Protocol (SMBv1) extends CIFS and adds security, disk management, and file support. This does not alter the basics of CIFS Protocol, but rather introduces new requests and responses, information levels, and flags. [1]

There are 6 main SMB commands for any transaction subprotocols. MaxBufferSize is used in a session parameter to determine the size of an SMB message. If a transaction message is larger than the SMB message size, as set by MaxBufferSize, then the client must use at least one of the SMB_COM_*TRANSACT_SECONDARY commands. These would send the transaction message that did not fit in the initial message. Each SMB transaction command has different subcommand codes, with 3 groups of subcommands. [2]

Windows SMB transaction has interesting qualities in how it is set up. In one buffer, a TRANSACTION struct, and a transaction data buffer are allocated. The TRANSACTION struct is followed by the data buffer in memory. [2]

A transaction buffer is, what is referred to as, a paged pool buffer. TRANSACTION is also an incredibly important struct member, with quite a few components inside to ensure that the struct works as intended. The components are the InSetup, InParameters, InData, OutParameters, and OutData. There are three main memory layouts for the transaction data buffer, two of which have certain parameters (Case 1 has In and Out buffers overlapped, Case 2 has no overlap, and Case 3 has InParameters overlapping with OutParameters and InData overlapping with OutData.) [2]

Transaction is only executed when the ParameterCount is equal to the TotalParameterCount and the DataCount is equal to the TotalDataCount. After the transaction is executed, the ParameterCount and the DataCount are used in determining the size of the OutParameters and OutData, respectively. [2]

A SMB_COM_*_SECONDARY request might be used to overwrite sent transaction data and parameters but adding in displacement, and ParameterCount and Datacount are added no matter what displacement value, as long as the displacement value is valid. For example, if the TotalParameterCount is 0 and Total DataCount is 16, the first transaction request has 8 bytes of data, the second can have 8 bytes with displacement 0. This would mean that the 8 bytes in the first transaction request get overwritten and thus the next 8 bytes of data are never written. [2]

B. Memory Descriptor List (MDL)

A Memory Descriptor List (MDL) is a kernel structure that describes buffers by a set of physical addresses. Data is written through the MDL by the driver after performing direct input/output to receive a pointer from the I/O manager. [1]

The Operating System (OS) uses an MDL to describe the physical page layout in a virtual memory buffer, even if these pages are discontinuous. Being able to control the MDL allows one to "write-what-where" the primitive. However, when the MDL is "locked", the specific physical pages are locked in mapped to certain specific virtual address spaces. [1]

C. HAL's Heap

HAL's Heap is a static area, located at 0xffffffff'ffd00000 or 0xffd00000 in the 64-bit and 32 bit versions of Windows, respectively. Prior to windows 8, it was executable. [1]

III. The Main Bugs

The Eternal family used a set of 9 different bugs that interacted with each other to be able to breach into different system. [2] However, the EternalBlue used 3 of these bugs, and a dive into those three is taken in the following section. These three are a Wrong Casting Bug, a Wrong Parsing Function Bug, and a Non-paged Pool Allocation Bug. [1]

A. Bug A: Wrong Casting Bug

This bug occurs when converting the File Extended Attributes (FEA) from the Os2 structure into the NT structure by the Windows SMB Implementation. NT Format is also known as the Windows format. This would lead to buffer overflow in the non-paged kernel pool. The following paragraphs will be delving into what that means. [1]

The structure of File Extended Attributes (FEA) is importantly used to describe the file characteristics. It can be thought about as pairs of a key and a value. [1]

There are a couple important functions that are related to the bug and are part of the srv.sys driver. The first of these are the SrvOs2FeaListToNt which converts the Os2 FEA list to an NT FEA list. This works by getting the Os2FeaList, calling SrvOs2FeaListSizeToNt, which gets the proper size for the NtFea List, allocating a buffer according to this size, and iterating over the Os2FeaList until it reaches the proper size, converting into NT format and adding into NtFeaList. The next important function, as mentioned in SrvOs2FeaListToNt, is the SrvOs2FeaListSizeToNt which calculates the needed size to convert Os2FeaList structures into the appropriate NtFeaList structures. This completes this by calculating the needed buffer size for NtFeaList, calculating how many records of Os2Fea from the Os2FeaList need to be converted into NtFea and storing the calculation, overwriting the previous value. The records are stored in bytes. The last important function for this bug is the SrvOs2FeaToNt, which converts the Os2Fea record into an NtFea record. [1]

The main bug in here is inside of the SrvOs2FeaListSizeToNt and with regards to shrinking. Before shrinking in the SMB Packet, the SMB_FEA is out of range of the SizeOfListInBytes. If the size of SizeOfListInBytes is below 2^{16}, then the SMB_FEA gets partitioned and the SizeOfListInBytes is adjusted to fit accordingly. However, the bug arises if the SizeOfListInBytes is above 2^{16}. The SizeOfListInBytes is made significantly larger to encompass the full SMB_FEA and incorporating some Unrelated Data. [1] If feaList.SizeOfListInBytes is larger than it the output buffer gets overflowed. [2]

There is an incorrect casting in the SrvOs2FeaListSizeToNt function which incorrectly enlarges the SizeOfListInBytes. The

remainder SizeOfListInBytes stores how many records (in bytes) of the Os2Fea that need to be converted to NtFea. However, the size of the returned NtFeaList buffer is correctly calculated for the appropriate size in converting a shrunk Os2FeaList to NtFeaList. This then leads to a situation where the size of the bytes that need to be copied is much bigger than the buffer size, which cases an Out of Bound write. Again, this only happens if the SizeOfListInBytes is above the range of 2^{16}. [1]

B. Bug B: Wrong Parsing Function Bug

This is the bug that enables and allows Bug A to function, by treating the Dword command as Word, which mean two different things, as they are two different sizes and thus update two different amounts of data. Dword is a double word (32 bits) and Word is 16 bits.

When transmitting a file over the SMB protocol, there are quite a few occurring functions, all data-related. [1]

Normally, any of the SMB_COM_TRANSACTION commands must be followed by a secondary command, for example SMB_COM_NT_TRANSACT must be followed with SMB_COM_NT_TRANSACT_SECONDARY. Another example is SMB_COM_TRANSACTION2 must be followed by SMB_COM_TRANSACTION2_SECONDARY. The difference between these two is the size of data that can be sent. In SMB_COM_TRANSACTION2, the size of the maximum data length is a Word, whereas with SMB_COM_NT_TRANSACT the maximum is a Dword. [1,2]

This bug lies in the secondary command sent, where there is no check for which secondary command is sent. This could cause SMB_COM_NT_TRANSACT to be followed by SMB_COM_TRANSACTION2_SECONDARY, or any other combination. This can cause a large amount of wrong information and causes incorrect data parsing. [1, 2]

This in turn enables Bug A: Wrong Casting Bug, and the data is parsed incorrectly. In the example, with SMB_COM_NT_TRANSACT followed by SMB_COM_TRANSACTION2_SECONDARY, it is a Dword followed by a Word. This tricks the data being parsed as if the command originally came from SMB_COM_TRANSACTION2, treating Dword as Word, which causes issues. [1,2]

C. Bug 3: Non-paged Pool Allocation Bug

This third bug lets a chunk with a specific size be allocated in the kernel non-paged pool with the specified size. This is used in the phase when creating a hole to be filled with a certain data size. This certain data size would cause an our of bound write to the next chunk, enabling and utilizing Bug A and Bug B. [1]

In order to establish user authentication on an SMB connection and establish an SMB session, an SMB_COM_SESSION_SETUP_ANDX request must be sent by the client. There are two different formats for an SMB_COM_SESSION_SETUP_ANDX request. The first one is for LM and NTLM authentication, and the second is for NTLMv2 (NTLM SSP) authentication. The important thing to note is that the two different formats have two different WordCounts (13 and 12, respectively). [1,2]

The bug exists in a function names BlockingSessionSetupAndX. This function handles the SMB_COM_SESSION_SETUP_ANDX request. If this request is sent with a WordCount 12 (Extended Security) but also with certain flags, it will process as a NT Security Request (WordCount 13), and this can occur vice versa. The ByteCount is incorrectly calculated and the allocation of size in the non-paged pool is created, which is bigger than the packet data. This allocation is later freed and reallocated using an NtFea chunk to overflow the next chunk. [1,2]

IV. Exploitation Technique

Before the exploit is completed, there are certain primitives that must be utilized with certain techniques to enable Remote Code Execution (RCE). [1]

A. Primitives and how they are used

The first of the primitives is an MDL (pMdl1) overwrite. As defined earlier, a virtual address in the MDL maps the data incoming to a specific address. This, if it was changed by overwriting, the incoming data is mapped to a place that can be controlled and changed. When writing a primitive to an MDL, an I/O data is mapped to a specific virtual address. In the exploit however, an Out of Bound (OOB) is written in the srv allocation due to Bugs A and B. Therefore, the header of the srvnet chunk can be overwritten. [1]

The other important primitive is the pSrvNetWskStruct Overwrite. The pSrvNetWskStruct is inside the srvnet header struct and it points towards the SrvNetWskScruct, which contains a pointer to HandlerFunction which is called when srvnet connection is closed. However, if the pointer in pSrvNetWskScruct is overwritten with an address to a fake SrvNetWskScruct which is controlled by the exploiters, the value of the pointer to that function is called when the srvnet connection is closed and thus enables Remote Code Execution. [1]

This is where HAL's Heap, as defined earlier, becomes important. The main quality of it that is important is that it is executable. If the MDL is overwritten, the data under the exploiters control has its location controlled as well. If the exploiters overwrite in the virtual address to a static address, such as in HAL's Heap, then the data that the client sends is mapped there. This can then be executed with changing the pointer to SrvNetWskStruct to the same address that was overwritten by the exploiters in the MDL, then a fake struct is created, which is preceded by shellcode. The shellcode is called upon when closing the connection with the srvnet, and Remote Code Execution (RCE) is achieved. [1]

B. Exploitation Flow

0) Take the initial state of the kernel pool, when non-paged, and HAL's Heap.

1) Using Bug A and Bug B, complete a srv allocation and only the connection for Os2Fea transmission is opened. In one

of the free chunks, it becomes a SRV chunk with free space for FEA structure.

2) SMB_COM_NT_TRANSACT fills the first part of OS2Fea. This is then filled by SMB_COM_NT_TRANSACT_SECONDARY or SMB_COM_TRANSACTION2_SECNONDARY. This is done as described in Bug A and Bug B, but completed without sending the last SECONDARY package. As the secondary is not sent yet, the Out of Bound (OOB) write is not yet triggered.

3) In the non-paged pool, multiple different srvnet connections are opened. Multiple connections are opened to increase the chance of srvnet overwriting. This is done by preceding the srv allocation of converted Os2Fea to NtFea. This then utilizes Bugs A and B and overflows the next chunk, which is the srvnet Header. The technique employed here is also used as a grooming technique.

4) This is where Bug C comes into play. The attacker/exploiter creates a chunk according to Bug C. This chunk is used as a placeholder for the converted Os2Fea to NtFea, and is the same size as the overflowing NTFeaList. NtFeaList is overflowing due to Bug A. Later the connection is closed and thus the chunk is freed and NtFea (write OOB) fills the hole. However, before it is freed, more srvnet connections are created. It is freed before the final packet of srv allocation. This packet allocates a chunk to store the NtFea data, done by (SMB_COM_TRANSACTION2_SECONDARY). This allocation is stored in the free chunk, most likely. Again, this is part of the grooming technique.

5) A new srvnet allocation and connection is created, located after the chunk that Bug C created. If the NtFea is located, as most likely it will be, this will overflow the srvnet chunk.

6) The chunk created by Bug C is finally freed.

7) Three main things occur in this important step. First, using SMB_COM_TRANSACTION2_SECONDARY, the last of the Os2Fea data is sent and the allocation converts Os2Fea to NtFea. The NtFea is allocated at the free hole, and then NtFea overwrites the next srvnet chunk. Thus, the srvnet header is overwritten and modified, and the two header properties point to the exact same address in HAL's Heap. The two header properties that are edited are the pSrvNetWskStruct and the MDL that is used to map, as talked about in the primitives section.

8) In this step, the first part that occurs is the incoming data arrives, from the overflowed srvnet connection. Due to the overwriting of the MDL, this is sent to HAL's Heap and written there. The user data written to HAL's Heap contains a fake struct, and thus pSrvNetWskStruct points to the fake struct. The connection then closes and due to HandlerFunction, the fake struct is called. Finally, the data from user after the creation of the fake struct contains certain shellcode, entitled DoublePulsar. This is written after the fake struct in HAL's Heap.

9) The srvnet connections are all closed, and the HandlerFunction pointed at the fake struct is executed. The fake HandlerFunction is executed with the shellcode function. [1]

Thus, the exploit is able to be completed and files are able to be read, downloaded, or edited. Files and programs are also able to be uploaded, and this is how the WannaCry ransomware became uploaded onto systems. [3]

V. Discussion and Conclusion

In this report, the exploit known as EternalBlue was explored along with its functionality. The exploit was used to deal significant damage to over thousands of systems and thus was patched by Windows as soon as possible. This exploit helped usher in a new age of exploitation research and security. Adam Kujwa, who is the director of Malwarebytes, said that he had "never seen anything before like this." Malwarebytes discovered the original version of the WannaCry ransomware. [5]

The papers that were read and used in the research all worked very well with each other, in being able to paint an incredible and accurate picture of how the exploit worked, what damage it did, and a simulation of the exploit itself. Citations 1 and 2 were the strongest citations with regards to the technical aspect of the exploit itself, as they went in depth into the Bugs and Exploit itself. Citation 3 was helpful as it painted a picture of how the exploit could be done, but was the weakest of the citations, given the available time and resources. Citations 4 and 5 helped paint a picture of the damage and intensity of the exploit and helped show its importance from a non-technical perspective. I think that they were all effective in their purpose and in doing so, they helped write an effective literature review. I personally do not have any other ideas or methods to be able to improve the solution that was created and implemented, as the current one fixed everything that needed to be fixed with a single patch update.

Overall, this was quite an interesting and unique research topic and paper, and I quite thoroughly enjoyed it. The papers and topic were all very interesting and I will be continuing to conduct research on my own personal time, to be able to further my knowledge in exploit, exploit prevention, and system/computer architecture.

References

[1] "Eternalblue - Everything there is to know." Check Point Research, 4 Feb. 2019, https://research.checkpoint.com/2017/eternalblue-everything-know/.

[2] Worawit. "Worawit/MS17-010: MS17-010." GitHub, https://github.com/worawit/MS17-010.

[3] "Simulating EternalBlue." Simulating EternalBlue Exploit Used by WannaCry Attack - SYANG.IO, https://syang.io/2017/05/17/Simulate-EternalBlue.html.

[4] "Microsoft Security Bulletin MS17-010 - Critical." Microsoft Docs, Microsoft, 14 Mar. 2017, https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2017/ms17-010.

[5] Newman, Lily Hay. "The ransomware meltdown experts warned about is here." Wired, Conde Nast, 12 May 2017, https://www.wired.com/2017/05/ransomware-meltdown-experts-warned/.