EternalBlue Exploit

By: Lucas Tran

Introduction

EternalBlue is an exploit that targets the SMBv01 protocol used by Windows computers. The attack makes use of several techniques such as heap spraying and buffer overrun to get into Windows computers. EternalBlue is best known for it's use in 2017's WannaCry, a ransomware which would encrypt a victim's files and ask for a ransom for the files to be decrypted. Using the TCP through port 45 it would spread to other computers, sending malformed data which would be handled by the SMB protocol, wherein the exploit can be found.

How EternalBlue was used by Attacks

The attacks listed above all use EternalBlue to get into Windows computers by using it to spread to other computers in the local network. Initially these malware, would get onto a victim's computer by having the victim download the file, but from there it would spread to other computers by generating random IPs at a rate of 25 IP/second and seeing that IP was vulnerable to the attack by checking if port 45 is open then if it is, the attack would run the exploit and spread the attack over the next vulnerable computer. Through this self prorogation, all the computers on network can be infected by any of these attacks in a matter of minutes.

Steps Behind the Exploit

Step 1: Initial Connection

There is a handshake between two computers to establish the connection

Step 2: Initial Sending of FEA data

In this step there is large amount of data being sent to the Windows machine to fill up the heap making it easier to run the overrun by causing a bug in a later step when the final part of this FEA data is sent.

Background info, FEA: What is FEA (File Extended Attributes)
A struct like object that describes the characteristics of a file. Which store key value pairs where the key is the attribute name and the value is the attribute value, which are stored in a list. In Os2 format the Os2FeaList stores these objects and a has a variable keeping track of the list size (stored as a double word ~ 2^{32} bits). In Nt format it is more like a linked list and requires a function to get the list size (stored as a word ~ 322^{16} bits) upon conversion from Os2 format. → this conversion part of the bug

Background info, Bug 1: What is the wrong casting bug
When the conversion function to get the NTFeaList size is correct, it should shrink the list size to match the size of the struct. However, the bug occurs when the size is as larger or larger than a double word. In this scenario the function treats the OS2FeaList size as a word only updating two bits leaving the rest the same. Which will then cause the NTFeaList to be larger than it is supposed to be. Then when reading the Os2FeaList with this size, SMB ends up reading more data than just the SMB structure where the overflow happens.

Background info, Bug 2: What is the wrong parsing function bug
The reason bug 1 parses incorrectly are because when sending the Os2 data it can be parsed as a dword through the SMB_COM_NT_TRANSACT then SMB_COM_TRANSACTION2_SECONDARY which parses a word.

Step 3: Heap spraying the heap

To send and increase the chances of the getting the shellcode to execute, the exploit heap sprays the heap by filling out the heap with srvnet structs and the DoublePulsar shellcode. The svrnet struct will have a handler function that executes on disconnection, we will over flow this later to point to the shellcode which the exploit knows the relative address because it created the structs.

Background info: What is heap spraying
A heap spray is a technique when you fill the heap with structs which also contain the shellcode, giving a better chance at a overflow. You can learn more about it from this video https://www.youtube.com/watch?v=Ec4UEtO7dPI

Step 4: Creating/using the buggy chunk

When we create this chunk using bug 3, it takes up space in the heap, because the attacker wants to make room for the NTFealist to be stored into. So that when we free up the chunk and send the last part of the FEA data the list ends up allocating the space where this chunk is. Before freeing step 3 is repeated to increase the chance that there are srvnet structs found after the chunk to overflow into.

Background info, Bug 3: What is the non-paged pool allocation bug
It is a bug in that lets an attacker allocate a specific amount of memory in the non-paged pool which is the heap where the FEA data is stored. By allocating the exact amount of space that the FEA structures take up, the extra data from bug 1 and 2 will not be a part of the chunk.

Step 6: Sending the last part of the FEA data

When sending this last bit of the FEA data, bug 1 and 2 occurs and this will overflow the return address of the handler function of on the svrnet structs if one of them was created in a location found next to the bug 3 chunk.

Step 7: Disconnection

As all the connections are closed, the handler function for each struct is executed, one of which is overflowed to contain a pointer to the shellcode thus executing that.

Mitigation

There are several ways to defend against this exploit.

1. Update Windows, this exploit has been patched and for those who use Windows this is the easiest and best way to defend against the exploit
2. Close port 45
3. Don’t use Windows