UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CASE STUDY ON ETERNALBLUE

Stefano Leggio

2052367

Contents

List of Figures

1 Introduction

1.1 History

EternalBlue is an exploit developed by the National Security Agency of United States targeting Windows systems[1]. In 2017 it was leaked by the Shadow Brockers hacker group along with other 35 exploits and hacking tools, the most relevant are:

- Fuzzbunch: An exploitation framework like Metasploit
- DanderSpritz: Command and control solution for the post exploitation
- DoublePulsar: Trojan
- EternalBlue: Service Message Block (SMB) protocol exploit

1.2 Relevance

EternalBlue exploits a vulnerability that has affected almost all the Windows versions, from XP to 10th version, providing full remote code execution[2]. Nowadays it is considered as one of the biggest leak ever happened to a national security agency.
To make the things worse, after the leak on 14th April 2017, EternalBlue was used in Ransomware[3] and Crypto Miner due to the large number of vulnerable devices. The most famouses cases are:

- WannaCry: Ransomware
- Adylkuzz: Cryptominer

The spread of these viruses was incredible, they were able to infect a network with a thousand of devices in a few minutes.

2 SMB protocol

Before entering the details of the exploit, it is important to describe what is the SMB protocol and how it works.
The Server Message Block (SMB) is a procol based on TCP/IP used for file and printer sharing inside a local network.

2.1 Microsoft Windows implementation

The SMB is used in Microsoft Windows systems since 1996 in two different services, making the computer a *workstation* or a *server*. Over the years a lot of versions of SMB suceedes, in this section it will be analyzed only the first version, because this is the one involved in the vulnerability.
The Windows implementation of SMB version 1 is an extension of the already existing Common Internet File System (CIFS) which was the network file-sharing protocol for Windows NT. SMB added some features on security and disk management, but most importantly it substituted the old NetBios service with an entire TCP connection.
In the SMB V1 there are three phases[4]:

- Establishing a TCP session
- Negotiating a dialect
- Establishing an SMB connection
- Accessing resources

It is important to notice that during the establishing of the SMB connection the client and server decide which is the maximum buffer size for each SMB message (also called transaction).
The Windows Server that receives the SMB request has a driver called *srv.sys* that aims to load balance the queue of the SMB commands received.

2.2 Structure

The SMB packets are divided into three parts: Header, Parameter Block and the Data block. The Header block has the following parts[4]:

- Command: SMB query to run into the server
- Flags: it indicates if it is a response or a request
- Errno: error number
- Signature
- Identifiers (PID, ID, RID, UID)

The Parameter block is used to specify the parameters values of the commands in the header. The last block instead regards all the data of the packet.

Figure 1: SMB packet structure

2.3 Transactions

The accessing of resources is done with Transcations, which are SMB messages that enables atomic read and write between client and server. From a packet view, they are just an SMB messages that embeds another SMB message in the data block.

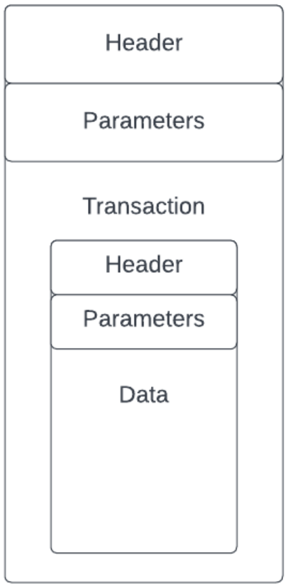


Figure 2: SMB transaction packet structure

In each Transaction there are the following parameters[5]:

- Offset: It indicates when the data block begins
- Count
- TotalCount
- Displacement: It indicates where start writing in the srv buffer

When a Transaction has a data block bigger then the *maxBufSize* specified in the connection establishment, the server requests some Transaction Secondary in order to complete the data transfer[5].

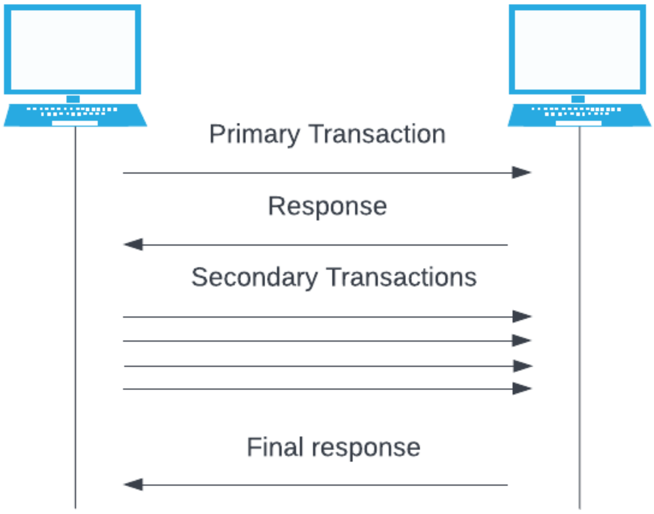


Figure 3: Transactions exchange scheme

There are different kinds of Transaction Secondary due to the different versions of the SMB implementation, some examples are: Transaction2, TransactionSecondary or NTTransaction2.

3 Vulnerabilities

In the following sections will be explained the main vulnerabilities that EternalBlue uses to complete the remote code execution. Then, in the section Exploit (6.0) they will be liked together for the final overview of the attack.

3.1 Integer cast error

This vulnerability is due to an error in the computation of the allocation size of a buffer, it is considered the main bug because it is the one that creates the buffer overflow.

SMB gives the possibility to include in the headers of the message the File Extended Attributes (FEA) which is a data structure that associates files with metadata. Because there is the possibility to include more than one file, the FEAs are grouped up i na list structure called FEAList. The following image shows how these two struct are composed.

```
struct FEA
{
    BYTE fEA;
    BYTE cbName;
    WORD CBvALUE;
}

struct FEALIST
{
    ULONG cbList;
    FEA list[],
}
```

Figure 4: FEA and FEAList structs

The FEAList struct has a variable called cbList used to memorize how many bytes is long the list of FEA.

Due to compatibility problems the FEAList that arrives to the Windows server in the header of the SMB message has to be casted to a custom list called NTFEAList, the following image shows how it is composed.

```
struct FILE_FULL_EA_INFORMATION
{
    ULONG NextEntryOffset;
    UCHAR Flags;
    UCHAR EaNameLength;
    USHORT EaValueLength;
}
```

Figure 5: Windows NTFEAList

The two struct are quite different, so the server has to manage a cast function.

One of the first things that the cast function has to do is computing the size of the buffer that has to be allocated for the NTFEAList. To do that it looks to the value of cbList in the FEAList. This function also checks that the value of cbList corresponds to the actual length of the FEAList, this is done by iterating over all the FEAs in the while cycle. In case the value of cbList is not correct and it is more that what it is expected it changes the value of cbList with the real list size.

The following image shows the NTFEAList buffer size computation function.

```
ULONG SrvOs2FeaListSizeToNt(FEALIST *FeaList)
{
    lastValidLocation = FeaList +  FeaList->cbList;
    fea = FeaList->list;
    ntBufferSize = 0;

    while (fea < lastValidLocation) {
        feaSize = fea->cbName + 1 + fea->cbValue;
        if(fea + feaSize > lastValidLocation) {
            SmbPutUshort(&FeaList->cbList, PTR_DIFF_SHORT(fea, FeaList));
            break;
        }
        ntBufferSize += FEA_SIZE(fea);
        fea = NEXT_REA(FEA);
    }

    return ntBufferSize;
}
```

Figure 6: FEA to NT casting function

This size fixing operation is the vulnerability because it considers the cbList as a UShort type (16 bit), instead it is a ULong type (32bit). Because of that it allocates less space then what the list needs causing an overflow of the NT buffer in the heap[6]. The following image shows clearly the incompatible types of cbList and the method SmbPutUshort().

```
ULONG cbList;
SmbPutUshort(&FeaList->cbList, PTR_DIFF_SHORT(fea, FeaList));
```

Figure 7: Integer cast error code

3.2 Mixing transaction types

Let's consider two different kinds of transaction in the SMB implementation:

- SMB_COM_TRANSACTION2 and SMB_COM_TRANSACTION2_SECONDARY: use WORD size parameters
- SMB_COM_NT_TRANSACT and SMB_COM_NT_TRANSACT_SECONDARY: use DWORD size parameters

They are very similar but used for different scopes: the trans2 are used for managing FEAs, instead the NTtrans are used for the transfer of large blocks of data. During the transmission of the FEAs we have to use the trans2 type, that is a problem for the attacker because this kind of transactions limit the parameters size as a WORD, so he will not be able to trigger the main bug. However there is a small bug that can help the attacker to work around this problem. The system doesn't check if the transactions types are consistent across the communication, so he can initialize the connection with an SMB_COM_NT_TRANSACT which use DWORD parameters, and then continue with SMB_COM_TRANSACTION2_SECONDARY.

No.	Time	Source	Destination	Protocol	Length	Info
9	0.002639	192.168.198.203	192.168.198.204	SMB	251	Session Setup AndX Response
10	0.002651	192.168.198.204	192.168.198.203	SMB	154	Tree Connect AndX Request, Path: \\192.168.198.203\IPC$
11	0.002652	192.168.198.203	192.168.198.204	SMB	114	Tree Connect AndX Response
12	0.002653	192.168.198.204	192.168.198.203	SMB	136	Trans2 Request, SESSION_SETUP
13	0.002654	192.168.198.203	192.168.198.204	SMB	93	Trans2 Response, SESSION_SETUP, Error: STATUS_NOT_IMPLEMENTED
14	0.004962	192.168.198.204	192.168.198.203	SMB	1138	NT Trans Request, <unknown>
15	0.005044	192.168.198.203	192.168.198.204	SMB	93	NT Trans Response, <unknown (0)>
16	0.005204	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
21	0.005578	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
23	0.005818	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
25	0.005971	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
27	0.006130	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
29	0.006231	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
31	0.006356	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
33	0.006467	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
35	0.006638	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
37	0.006762	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
39	0.006936	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
41	0.007078	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
43	0.007121	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
45	0.007244	192.168.198.204	192.168.198.203	SMB	4207	Trans2 Secondary Request, FID: 0x0000
47	0.007405	192.168.198.204	192.168.198.203	SMB	107	Echo Request
48	0.007461	192.168.198.203	192.168.198.204	SMB	107	Echo Response
52	0.007938	192.168.198.204	192.168.198.203	SMB	191	Negotiate Protocol Request
53	0.013157	192.168.198.203	192.168.198.204	SMB	185	Negotiate Protocol Response
54	0.013354	192.168.198.204	192.168.198.203	SMB	139	Session Setup AndX Request
55	0.013428	192.168.198.203	192.168.198.204	SMB	251	Session Setup AndX Response
111	0.017937	192.168.198.204	192.168.198.203	SMB	191	Negotiate Protocol Request
112	0.018167	192.168.198.203	192.168.198.204	SMB	185	Negotiate Protocol Response
113	0.019419	192.168.198.204	192.168.198.203	SMB	139	Session Setup AndX Request

Figure 8: Mixing transaction types

3.3 Session setup allocation error

In the implementation of SMB V1 there are two kinds of authentication:

- LM/NTLM
- NTLMv2

The choice between is done at the beginning of the session setup with the command SMB_COM_SESSION_S containing the parameters of LM/NTM or NTMv2. The following images shows how these parameters and values are composed.

```
//LM/NTLM

SMB_Parameters
{
    UCHAR WordCount;
    Words
    {
        UCHAR AndXCommand;
        UCHAR AndXReserved;
        USHORT AndXOffset;
        USHORT MaxBufferSize;
        USHORT MaxMpxCount;
        USHORT VcNumber;
        ULONG SessionKey;
        USHORT OEMPasswordLen;
        USHORT UnicodePasswordLen;
        ULONG Reserved;
        ULONG Capabilities;
    }
};
```

```
//LM/NTLM

SMB_Data
{
    USHORT ByteCount;
    Bytes
    {
        UCHAR OEMPassword[];
        UCHAR UnicodePassword[]];
        USHORT Pad[];
        SMB_STRING AcccountName[];
        SMB_STRING PrimaryName[];
        SMB_STRING NativeOs[];
        SMB_STRING NativeLanMan[];
    }
};
```

Figure 9: LM/NTLM parameters and data

```
//NTLMv2

SMB_Parameters
{
    UCHAR WordCount;
    Words
    {
        UCHAR AndXCommand;
        UCHAR AndXReserved;
        USHORT AndXOffset;
        USHORT MaxBufferSize;
        USHORT MaxMpxCount;
        USHORT VcNumber;
        ULONG SessionKey;
        USHORT SecurityBlobLength;
        ULONG Reserved;
        ULONG Capabilities;
    }
};
```

```
//NTLMv2

SMB_Data
{
    USHORT ByteCount;
    Bytes
    {
        UCHAR SecurityBlob[SecurityBlobLength];
        SMB_STRING NativeOs[];
        SMB_STRING NativeLanMan[];
    }
};
```

Figure 10: NTLMv2 parameters and data

Notice that the ByteCount value indicates how much memory the server has to allocate for the session.

The choice between these two is done by checking if the wordCount is 13 (LM/NTLM) or 12 (NTLMv2), in this second case it looks also if it is defined the CAP_EXTENDED_SECURITY. If one of these conditions is not satisfied the request is rejected with an error. The server after

the first check controls again that it has CAP_EXTENDED_SECURITY and the FLAGS_EXTENDED_SEC
in the header, if this is true this is considered as NTLMv2 request.

Here there is the bug: the attacker can send a structure for NTLMv2 without setting the flag FLAGS_EXTENDED_SECURITY in the header. The server will interpret it as a LM/NTLM authentication when instead it is NTLMv2.

The following image shows the server side function for the detection of the authentication method.

```
BlockingSessionSetupAndX(request, smbHeader)
{
    if(!(request->WordCount == 13 ||
        (request->WordCount == 12 &&
        request->Capabilities & CAP_EXTENDED_SECURITY))) {
        return ERROR;
    }

    if((request->Capabilities & CAP_EXTENDED_SECURITY) &&
       (smbHeader->Flags & FLAGS_EXTENDED_SECURITY)) {
        //NTLMv2
        getExtendedSecurityParameters(request);
    } else {
        // LM/NTLM
        getNTSecurityParameters(request);
    }
}
```

Figure 11: Session setup allocation error code

Now the server thinks that the words in the structure are 13 instead of 12. Because of that the server reads the ByteCount value inside the SMB_STRING buffers, so the attacker is able to allocate how much memory he wants.

This is called remote heap allocation and it will be very useful to control in which part of the memory the NT buffer overflows.

4 Memory analysis

In this section it will be explained how the SMB buffers are stored in the Windows systems. This is very important in order to know what the attacker is overflowing and how he can be able to execute the code.
At the end of the section there will be a short overview about the grooming technique, a procedure used to predict where is the next memory allocation.

4.1 Srvnet buffers

A srvnet buffer is a struct used to store the content of smb transactions, it is composed by three components:

- srvnet_wsk_struct: It contains a list of functions pointers, they are executed by the server machine in some specific occasion. For example when the connection is stopped it executes the function SrvNetWskReceiveComplete() to process the data in the buffer.
- Memory Description List (MDL): A kernel structure that maps the data buffer to physical memory fragments.
- The data buffer itself

Figure 12: Srvnet buffer

4.2 Hal's Heap

The Windows Hardware Abstraction Layer (HAL) is a module that runs in kernel mode and it is loaded on Windows boot. It consists in a set of routines to access hardware resources through programming interface. This is used to allow programmers to write software which is hardware-indipendent.

This is a particular heap in Windows because the Address Space Layout Randomization (ASLR), which consists in the randomization of the memory addresses, doesn't regard the ones in the HAL's heap which are always costant. Fortunately, in the last versions of Windows 10 also the HAL's heap is randomized, but not at the time of the exploit. Another important feature of the HAL's heap is that it has execution permissions is some Windows version.

It is clear now that this part of memory is perfect for the exploitation, indeed here is where the attacker wants to execute the payload.

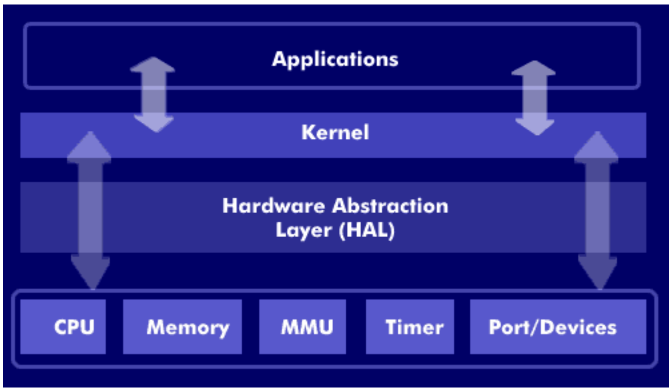


Figure 13: Hardware description layer

4.3 Grooming technique

If the attacker wants to use a buffer overflow as a remote code execution, he has to control the memory that he is overwriting. There are many ways to do it, in the case of EternalBlue this is done with the grooming technique. This method is based on the concept that the memory allocations tend to reuse the same chunks of memory. So if the attacker allocates a buffer of X bytes and then free it, it is very probable that if he does another malloc() of X bytes it will be stored in the same location.

Thanks to that, the attacker can predict the location where the buffer will be stored by creating a custom-sized memory and free it just before he needs to store the new buffer. In the case of EternalBlue this technique will be used by doing a small heap allocation with secondary transaction and a custom-sized heap allocation with the vulnerability explained in 3.3. The following images show the passages used for this technique.

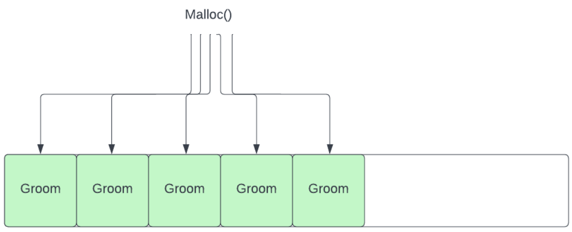


Figure 14: Hardware description layer

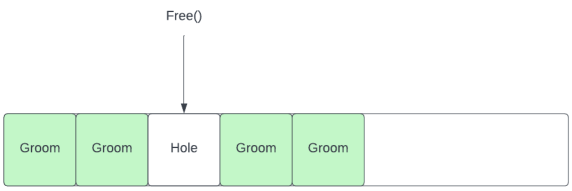


Figure 15: Hardware description layer

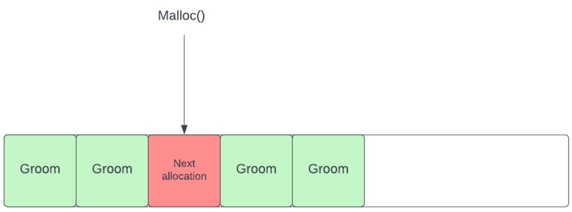


Figure 16: Hardware description layer

5 Doublepulsar

Before talking about the complete working exploit is it important to spend a few words regarding the payload used by EternalBlue.
As anticipated in the introduction, this was also created by the NSA for short term monitoring in Windows systems. It is an advanced payload which is installed in the RAM memory, for that is for ”short term monitoring”, and it runs in a kernel mode. It opens a backdoor using the SMB port and service, this increments the difficulty of detection.
It will be installed and executed in the HAL’s heap with full privileges, this will gain the full control of the system to the attacker.

6 Exploit

In the following sections there are all the passages for the EternalBlue remote code execution.

6.1 Small heap grooming

The first step is making a small heap grooming with some secondary transactions.

The attacker sends the first transaction and some secondary transactions without the last one.

In this way the server will allocate a srvnet buffer for each transaction in the memory.

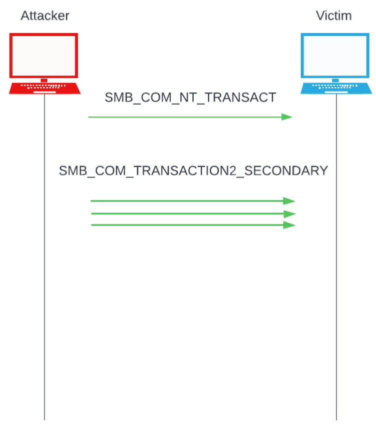


Figure 17: Small heap grooming - communications

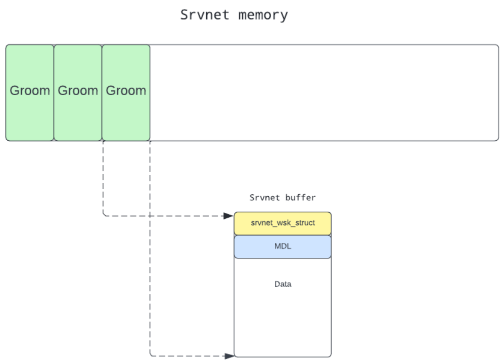


Figure 18: Small heap grooming - memory buffer

6.2 Session setup allocation

In this step we use the session setup bug explained in 3.3 to create a memory allocation with a specific size.

The size of the memory allocated will be the same of the future NT Buffer in order to do the remote heap allocation.

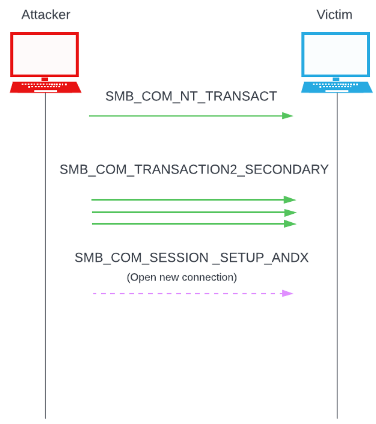


Figure 19: Session setup allocation - communications

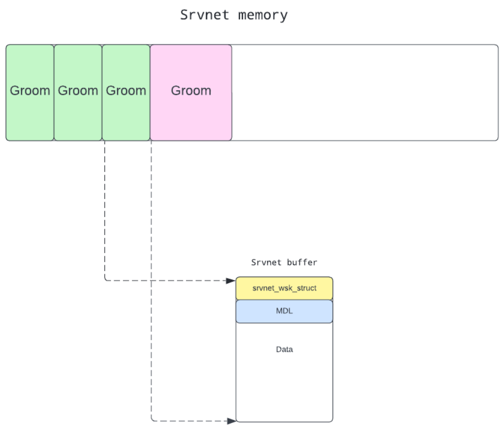


Figure 20: Session setup allocation - memory buffer

6.3 Keep grooming

In this step the attacker sends some secondary transactions to make the custom-sized memory contiguous to another srvnet buffer.

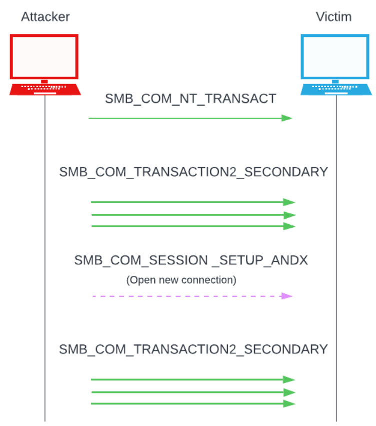


Figure 21: Keep grooming - communications

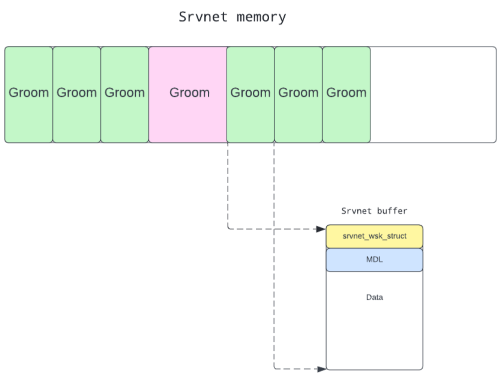


Figure 22: Keep grooming - memory buffer

6.4 Creating the hole

Now the attacker has to close the session opened in 6.2 in order to create a custom-sized hole in the heap memory.

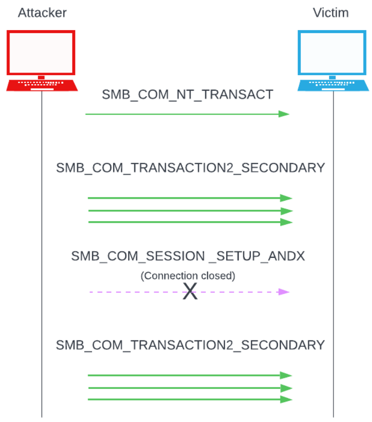


Figure 23: Creating the hole - communications

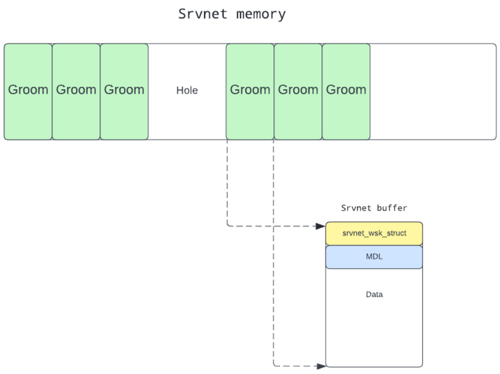


Figure 24: Creating the hole - memory buffer

6.5 Buffer overflow

Because of the grooming technique is very probable that the next transaction's buffer will be memorized in the hole that we have created in the previous step because it has the same size of the hole.

Now, the attacker can use the main bug explained in 3.1 to overflow the FEAList and overwrite the headers of the contiguous srvnet buffer.

In particular, he overwrites the pointers of the svrnet struct and the MDL in order that they point to the HAL's heap addresses.

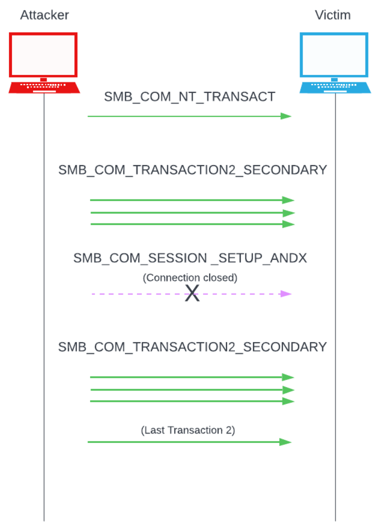


Figure 25: Buffer overflow - communications

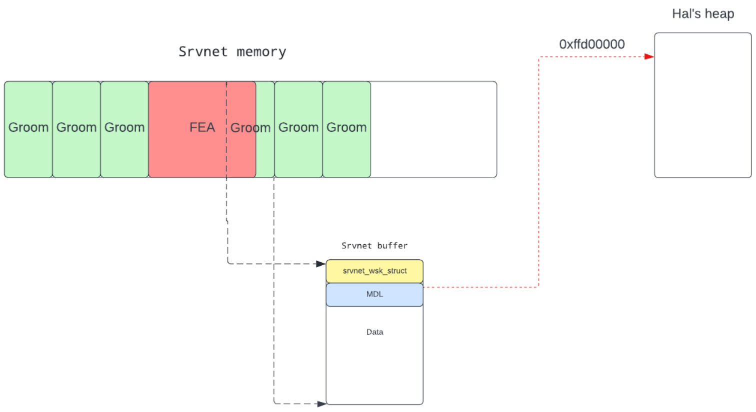


Figure 26: Buffer overflow - memory buffer

6.6 Payload injection

At this point, the data of the next transaction will not be stored in the srvnet buffer because the MDL points to the HAL's heap.

So, the attacker sends the payload, which is the DoublePulsar shellcode and a function handler, in the next transaction data. Now the attacker has memorized his payload in the HAL's heap.

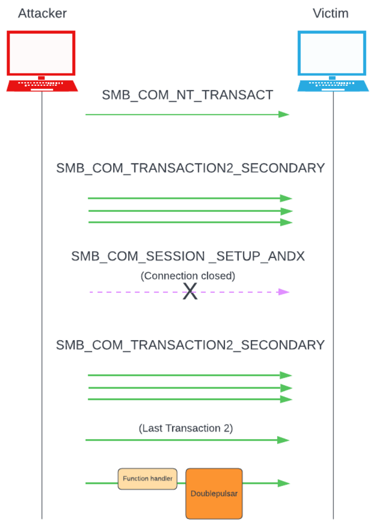


Figure 27: Payload injection - communications

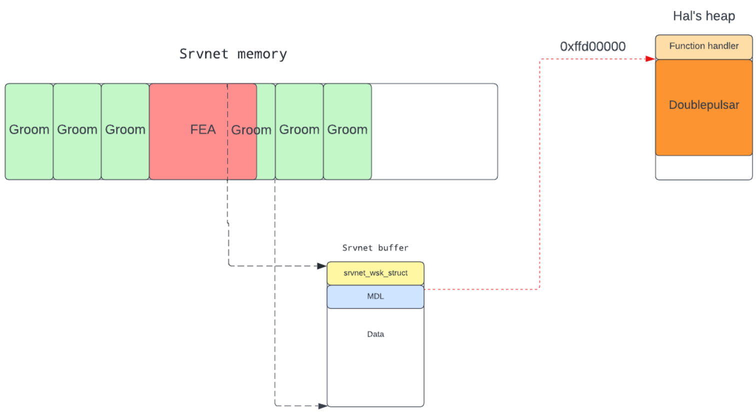


Figure 28: Payload injection - memory buffer

6.7 Remote code execution

In this last point the attacker wants to execute the DoublePulsar shellcode in the HAL's heap. To do that the attacker closes all the auxiliary connection, this will trigger the function SrvNetWskReceiveComplete() for each memory allocation.
For the srvnet buffer that he has overflowed the pointer of the SrvNetWskReceiveComplete() points to the function handler in the HAL's heap. When the function handler is triggered it executes the DoublePulsar shellcode with root permissions.
The attacker has successfully executed the payload on the Windows server and he has the complete control of it.

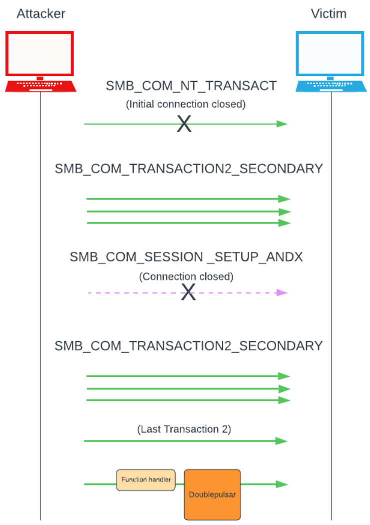


Figure 29: Remote code execution - communications

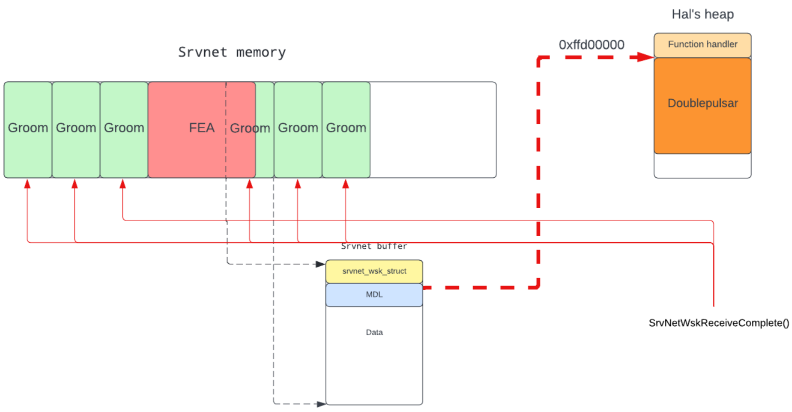


Figure 30: Remote code execution - memory buffer

7 Conclusions

Due to his potential, EternalBlue was implemented in a lot of ransomware like WannaCry. Between 2017 and 2018 the spreading was incredible. Just in 2017 were infected more than 200.000 computers in 150 countries. Also in Italy there where many cases, in particular part of the network of Milano Bicocca University was compromised.

Il virus Wannacry arrivato a Milano: colpiti computer dell'università Bicocca

L'ateneo esclude la violazione della rete interna: contagiati dagli hacker solo pc usati dagli studenti

Figure 31: Milano Bicocca infected by WannaCry

The National Health Service hospitals in England and Scotland was one of the largest entity compromised by this ransomware. It was particullary easy for the ransomware to spread the infection because the file sharing between hospitals was based on the SMB protocol over internet.

BBC Sign in Home News Sport Reel Worklife

NEWS

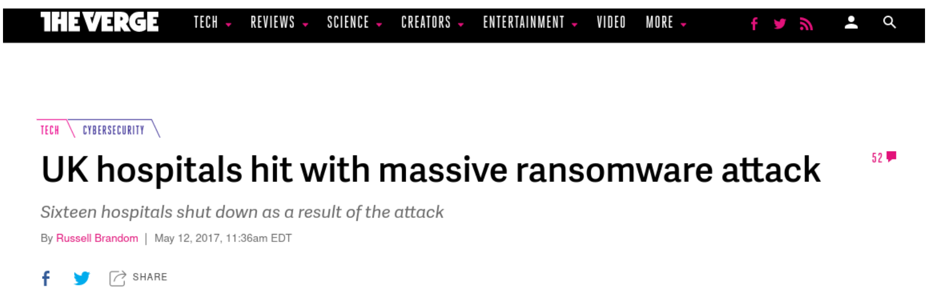
Home | War in Ukraine | Coronavirus | Climate | Video | World | UK | Business | Tech | Science | Stories

Tech

Massive ransomware infection hits computers in 99 countries

13 May 2017

Figure 32: BBC article about WannaCry



THE VERGE TECH REVIEWS SCIENCE CREATORS ENTERTAINMENT VIDEO MORE

TECH \ CYBERSECURITY

UK hospitals hit with massive ransomware attack

52

Sixteen hospitals shut down as a result of the attack

By Russell Brandom | May 12, 2017, 11:36am EDT

SHARE

Figure 33: The Verge article about WannaCry

Cyence, a cyber risk modeling firm, has estimated the total losses due to WannaCry at $4 billion, making it one of the most damaging cyber attack.

Bibliografia

[1] *NSA hacked*. URL: https://arstechnica.com/information-technology/2017/04/nsa-leaking-shadow-brokers-just-dumped-its-most-damaging-release-yet/.

[2] *Microsoft Security Bulletin MS17-010*. URL: https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010.

[3] *Wannacry ransomware*. URL: https://support.eset.com/en/ca6443-vulnerability-cve-2017-0144-in-smb-exploited-by-wannacryptor-ransomware-to-spread-over-lan.

[4] *Service Message Block (SMB)*. URL: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-smb/f210069c-7086-4dc2-885e-861d837df688.

[5] *SMB Transactions*. URL: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/9d877a4a-c60c-40e9-8520-849c5a94fc1d.

[6] *Exploit database Eternalblue*. URL: https://www.exploit-db.com/exploits/41987.