CYBERCONTAINMENT 4

- ETERNAL_HELL -

THE WILD SPRING OF 2017

PART I: ETERNALBLUE

Arch0nte – 04/2020

TABLE OF CONTENTS

INTRODUCTION AND CONTEXT

A LITTLE BIT OF HISTORY : ETERNALBLUE

- EternalBlue is an exploit for Microsoft Windows, abusing a vulnerability in the Server Message Block protocol (CVE-2017-0144). By sending specifically crafted packets via this protocol, an attacker can execute code remotely. The issue was patched on the 14th of march 2017 by Microsoft for all supported versions of Windows.

- **The exploit tools were published AFTER the patches so, had everyone updated their Windows computers and servers, only servers and computers on older, unsupported versions of Windows could be infected.**

ETERNALBLUE : HOW ?

- It exploits 3 Bugs (A,B and C) in the SMBv1 protocol that allowed Remote Code Execution. This protocol is mainly used for file sharing over a network.
- Bug A is a problem when the protocol casts the File Extension Attributes from a structure usable by Os2 (an Old Microsoft/IBM OS) to a structure usable by a Windows NT OS (everything from v.3.1 in 1993 up to now).
- Bug B is a bug in the way the protocol parses long messages if the type of the message changes between the packets sent.
- Bug C allows in a specific case, to allocate a free buffer that can be used later.

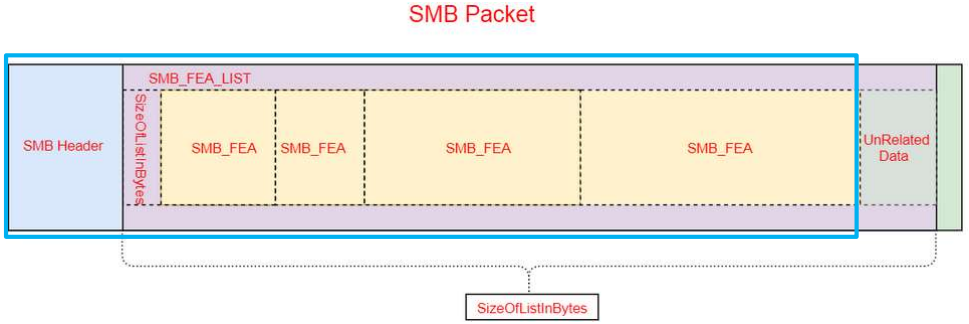
BUG A : THE CASTING COMPLICATION

Figure 1 : Steel being casted into moulds (https://pxhere.com/fr/photo/700619)

THE CASTING COMPLICATION 1/2

- File Extension Attributes describes files characteristics. It is different in Os2 and in NT (Windows) so when sending a file, through SMB from an Os2 to a NT system, you need to cast these FEA into a format NT can understand.
- Inside the code of the functions called to do this change of format, there is a mistake : when dealing with a FEA list too long, the program is supposed to discard all of the out of range data, shrinking the data to a size acceptable for the new packet.
- **But**, when dealing with a FEA list of over 2**16 Bytes, a bug can make the function enlarge the amount of data in its scope, instead of restricting it. Since the buffer size is calculated, without taking that bug into account, this means we have data being written in a buffer that is too small, in short, a **buffer overflow**.

Figure 2 : Sketch showing the data written out of the buffer (the blue square).
Original from : https://research.checkpoint.com/2017/eternalblue-everything-know/



THE CASTING COMPLICATION 2/2

So, why does this *shrinking* function enlarge instead of shrink ?

- The problem comes from a human mistake during the development. The function **mishandles the type** of a variable in the shrinking function. The variable, SizeOfListInBytes is a DoubleWord (or DWord) of 4 Bytes but the shrinking function treats it as a Word, of 2 Bytes.

  Thus, the 2 bytes with the most weight are never modified by the shrinking, causing the possible enlargement !

BUG B : THE PARSING PROBLEM

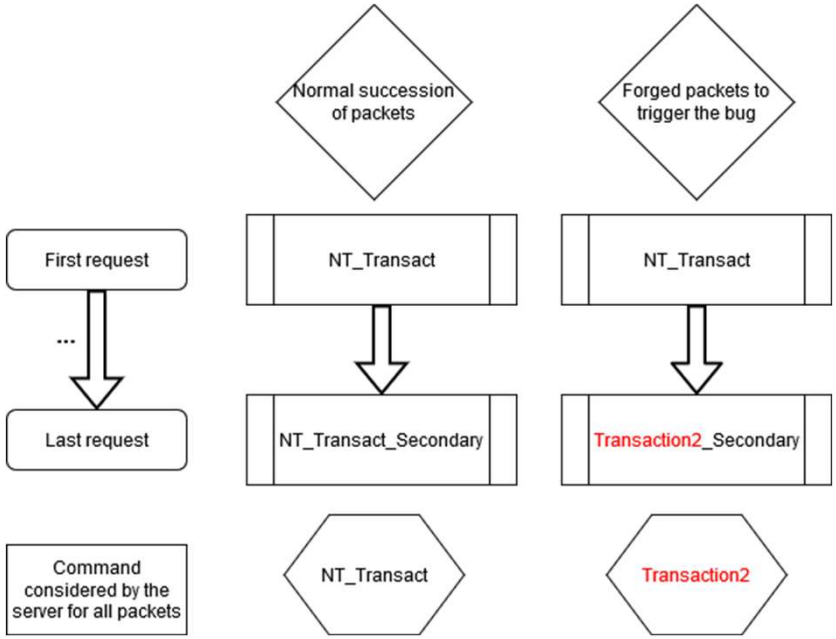
THE PARSING PROBLEM 1/2

- When transmitting large amount of data through the SMB protocol, this data can be sent with different commands attached to them. The ones that we're going to work on are the following : Transaction2 and NT_Transact.
  If the data is too big for a single packet, the program will split it and set the command as XXXXX_Secondary to the packets that follow the first (where XXXXX is the command set on the first packet).
- In Transaction2 packets, the maximum amount of data per packet is set in a parameter of the type Word (2 bytes) in the header. **But** in NT_Transact packet, that same parameter is of the type **DWord** (4 bytes).
- In addition, there is **no check to verify which command sent the first packet** : the parsing of the data is done according to the type of the command of the last packet received.

THE PARSING PROBLEM 2/2

- So if you send a package with it's command set as NT_Transact, which uses a DWord to define the size of the exchange and follow it up with a Transaction2_Secondary, which uses a Word, the protocol will parse all the packets as if they had been sent via Transaction2. Thus, (exactly like in bug A) a **DWord is treated as a Word** which causes parsing issues.

Figure 3 : Diagram showing how to trigger the bug in the protocol



BUG C :THE BOTHER BUFFER

THE BOTHER BUFFER 1/2

- In the requests used by the client to setup a SMB session, there is a call to the BlockingSessionSetupAndX function. This function has a series of if statement to make sure the format is right and to differentiate between Extend Security request and normal NT security.

Figure 4 : Extract of the pseudo-code of the function (Source : https://research.checkpoint.com/2017/eternalblue-everything-know/)

```
// check word count
if (! (request->WordCount == 13 || (request->WordCount == 12 && (request->Capablilities & CAP_EXTENDED_SECURITY))) ) {
    // error and return
}

// ...

if ((request->Capablilities & CAP_EXTENDED_SECURITY) && (smbHeader->Flags2 & FLAGS2_EXTENDED_SECURITY)) {
    // this request is Extend Security request
    GetExtendSecurityParameters(request);  // extract parameters and data to variables (allocation)
    SrvValidateSecurityBuffer(request);  // do authentication
}
else {
    // this request is NT Security request
    GetNtSecurityParameters(request);  // extract parameters and data to variables (allocation)
    SrvValidateUser(request);  // do authentication
}
```

- As we see immediately in the first statement, tests to check the consistency between the values do not check on the Flags2 value. So an Extended_Security request with a WordCount of 12 and a CAP_Extended_Security but without the Flags2_Extended_Security flag will be processed as a NT security request. **Even though the format of the parameters given isn't the same (the WordCount is supposed to be of 13).**

THE BOTHER BUFFER 2/2

- This means that the program will first try to check the parameters for a NT_Security request ("GetNtSecurityParameters(request)") but it won't be able to use the values correctly because of the incorrect format. This leads the program into **allocating a space but not using it**. This is useful because it allows us to create a placeholder there which can be used either to overflow the next chunks or to write something there.
- This could be **easily fixed** by just checking that the entry parameters are correct (with a statement like "elif(request –>WordCount == 13)" instead of the "else" statement) !

THE EXPLOIT

THE EXPLOIT

- There are 3 parts in the exploit and to explain it, I'll start at our final objective !

1) Our end goal is to change the value of the variable HandlerFunction, which contains a pointer to the function called when the connection is closed. This variable is reached through a pointer stored in the struct SrvNetWskStruct, itself called through a pointer variable pSrvNetWskStruct, located in a static position. We want to **change the value of HandlerFunction to an address that we control so we can inject shellcode**.

2) To inject code there and to modify all of this, we first need to be able to control where we can write. Fortunately, our out-of-bond writing (bug A & B) allows us to **reach and overwrite a MDL** (Memory Descriptor List) **structure which maps the data received** by the server to an address. By controlling that value, we can decide where the data we're sending is stored.

3) Thus, we only need to know where the variable pSrvNetWskStruct is stored and then to create, in an unprotected buffer space we control (bug C) in which we write thanks to our modified MDL, a **fake struct SrvNetWskStruct**. In that struct, **in place of a normal pointer to the HandlerFunction**, we will write a **pointer to where we wrote our shellcode**. Then, we just need to modify the value of the pSrvNetWskStruct to point to our new fake struct and we will have a RCE ☺

CONCLUSION

CONCLUSION

- That exploit was one of the main spreading ways of two of the most devastating malwares ever created, NotPetya and WannaCry.
- It was initially developed by the NSA but released to the public by a Hacker group (after Microsoft published patches but before a lot of companies and governments updated their systems).
- The few next posts will explain in greater details everything about these malwares and their impact on the digital culture.
- Thanks again for taking the time to read my blog ☺ Hope you learned something and/or enjoyed it ^^

BIBLIOGRAPHY

- https://en.wikipedia.org/wiki/EternalBlue
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144
- https://research.checkpoint.com/2017/eternalblue-everything-know/
- Link to the original publication of the Shadow brokers : https://steemit.com/shadowbrokers/@theshadowbrokers/lost-in-translation

- If you have any questions : archonte@hackademint.org