Source: Bing Chat AI

LABORATORY 7. NETWORK SECURITY

COMPUTER SCIENCE DEPARTMENT. UNIVERSITY OF OVIEDO

Computer Security | 2023 – 2024 (V4.0 *"S-81 Isaac Peral"*)

Content

Infrastructure of this laboratory

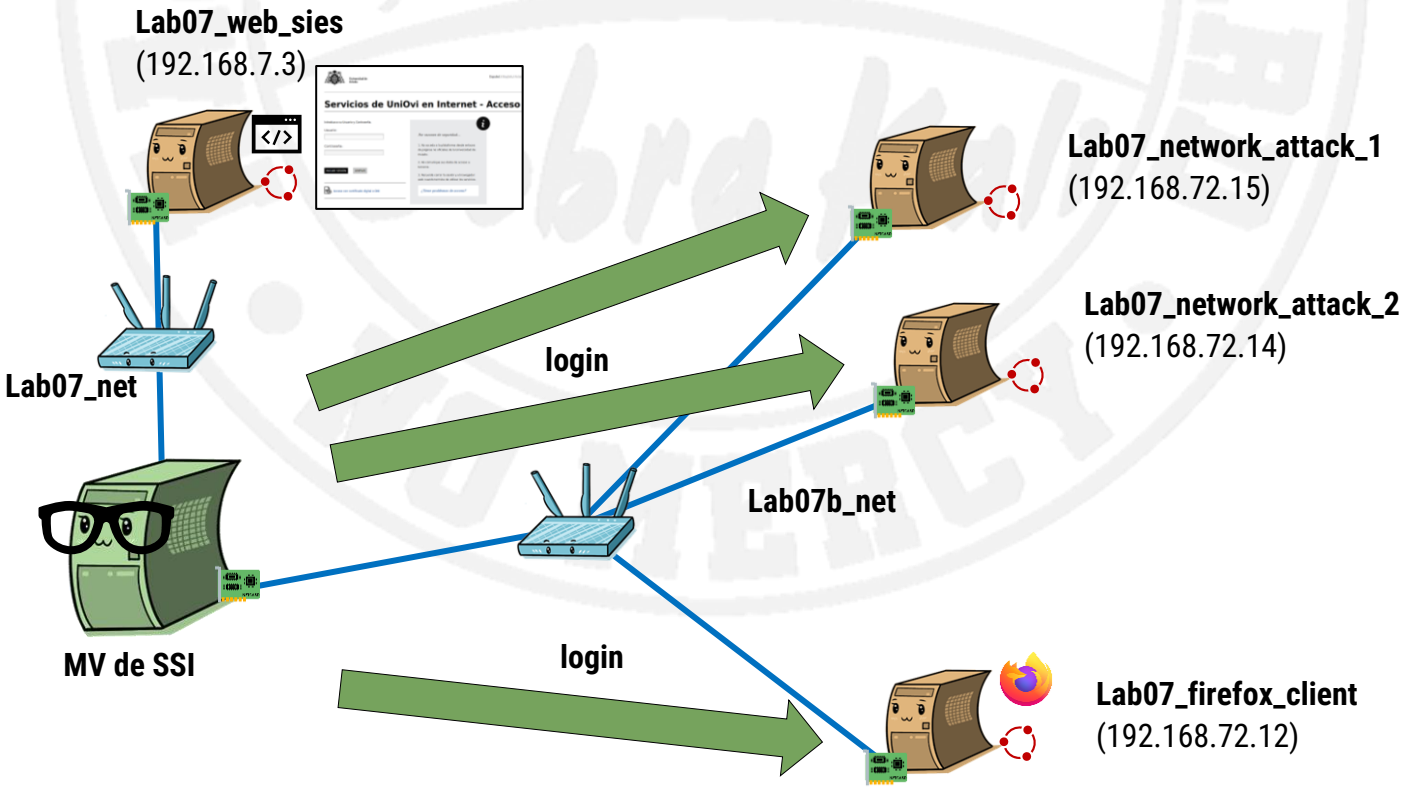
*(**NOTE:** To avoid issues, we strongly recommend that you run this lab infrastructure **on the base VM instead of its hardened versions** that you may have created in previous labs. In real life, you should do the opposite, but real implementations also have a validation time to ensure that everything works as expected after hardening.*

***NOTE**: Some of you may have had trouble running Firefox from a container, which we will be doing in this lab. To avoid them if they happen to you, follow these steps:*

- *Open the* `build_lab07.sh` *file*
- *Add the following line:* `xhost + local:root`
- *Build the lab like other times*

The infrastructure of this laboratory is made up of containers communicated through internal networks with the MV. Most will be used interactively.

- Two containers are "network attack" containers and have the tools to test the *firewall* and other protection software that we are going to look at in this lab. Information that is written to the `/etc/sysctl.d` directory will also be written to the VM directory `volume_data/network_data_<N>/sysctl.d/`
- Another container is enabled to run GUI applications and has *Firefox* installed.
- Finally, the last container is a web server with the *front-end page* of the Uniovi intranet communicated with the VM with a special private network (`192.168.7.0/24`) and with fixed IP.



🔍 ***NOTE:*** ***Some of the operations in this lab require you to know the IP address of the containers and networks they are using. Please remember to use*** `ifconfig` ***inside the containers to know their current IP addresses***

These containers have the necessary *software* installed to carry out the laboratory activities. However, several of the products we are about to use are incompatible with containers, and **therefore need to be installed on the *Ubuntu virtual machine*.** In this case, the containers will act as "attacker clients" of the VM, so you can test that they work without using more external elements. Which software is targeted at containers, and which is to be installed on VMs is clearly indicated in each activity.

On the other hand, **you will also need to use the "*Wazuh machine*"** from the previous lab to do an exercise. Remember that if you do not have a machine at home capable of supporting the creation of a VM with these characteristics, we recommend **that you prioritize *Wazuh*'s activity to do it in the lab**, where you do have hardware that supports that VM. You can even start with it without any problems.

🔍 ***NOTE: It has been detected that sometimes the lab build fails because apparently the firewall boots before the Docker and a conflict occurs. If this is your case, do this to fix it:***

- ***Stop the firewall***
- ***Stop the Docker service***
- ***Start the Docker service***
- ***Run the firewall as the last operation***

On the other hand, sometimes the firewall GUI does not correctly display the most current information about the machine's network status. This information does appear if the firewall is handled from the command line, with the command `firewall-cmd –list-all-zones`

Block 1: Securing a host's networks. *PF* firewalls

Exercise L7B1_FWZONES: Managing Firewalld zones

Description of the activity / Practical application

You need to **separate your network interfaces into** `firewalld` zones so you can assign different configurations to them

Expected results

This activity will end when you can **assign zones to the network cards in the *firewall and*** activate and test that a service can be accessed in the "`work`" zone.

Additional information required to do it

This activity will teach you how to use `firewalld` to perform typical network security operations that illustrate parts of the SNI (*Secure Network Infrastructure)* of the theory.

> 🔍 `firewalld` ***is a more powerful product than the traditional*** `ufw` ***that comes with every Ubuntu distribution. It works for both Debian and RedHat distributions and you also use it in ASR.***

`Firewalld` complies with the same functionalities (and more) as `ufw`. However, having both enabled at the same time causes problems, so first **we need to make sure that** `ufw` is disabled: `sudo ufw disable`.

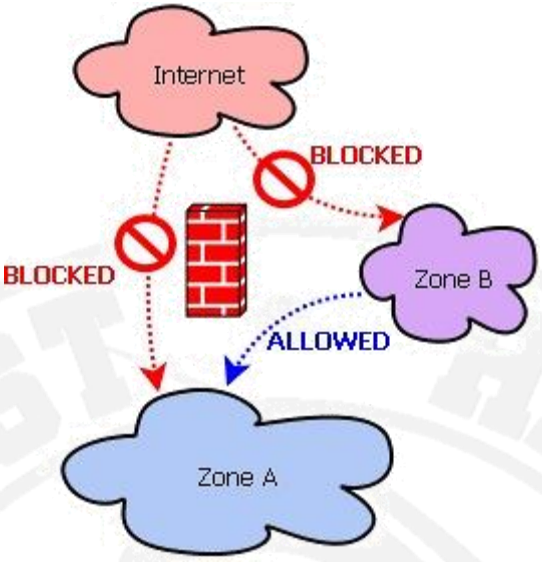
Once this is done, we need to install the appropriate *software* to use and manage `firewalld`, consisting of the firewall service itself *and its GUI* (`firewall-config`). You can install both this way: `sudo apt install firewalld firewall-config`.

> 🔍 ***Once this is done, it is very important to restart the virtual machine.***

Once the VM has rebooted, we can work with `firewalld`. The first thing we need to understand about this *firewall* is that it distributes the different network cards that we have in the VM into **zones.** Zones are predefined sets of rules that specify what traffic should be allowed based on the level of trust in the networks to which the computer is connected.

You can assign any of the network interfaces from the machine that has `firewalld` to a zone. From then on, all machines connected to the same network as each of those network cards (assuming that they send all their traffic to other networks through that card, i.e., that the firewalled machine **acts as a gateway between networks**), will belong to that zone. You will then be able to distribute the machines by zones, forming something similar to what we saw in theory with the SNI or this diagram, which is a simpler version.

🔍 ***In both cases, a principle is fulfilled: the firewall acts as a "policeman" and "customs inspector" of all connections, always putting itself "in the middle"***



Source: https://akm111.wordpress.com/2017/04/29/linux-firewalld-zones-and-services

Below are the zones that incorporate by default `firewalld`:

- **docker**: A special zone that initially contains all the networks created by our *Docker containers*. We have it because it is created when *Docker is installed* on the system.
- **Zones where inbound connections are prohibited, and only outbound connections are allowed**
  - **drop**: All incoming connections are deleted without any notification
  - **block**: All incoming connections are rejected (client is notified)
- **Zones where incoming connections are allowed that are previously authorized by us.** As you can see, they are designed for different situations and networks in which the machine that has `firewalld` installed can be found
  - **public:** For use in **untrusted public areas**. You do not trust other computers on the network.
  - **external:** For use on **external networks** with NAT masking enabled when the system acts as a gateway or router.
  - **Internal:** For use in **internal networks**, when the system acts as a gateway or router. The other systems in the network are generally trusted.
  - **dmz**: Used for teams located in a **demilitarized zone or DMZ** (**Theory Topic 5**), which will have limited access to the rest of your network.
  - **work**: Used for **work machines**. Other computers on the network are generally trusted.
  - **home**: Used for **household machines**. Other computers on the network are generally trusted.
- **Areas where no connection is restricted**
  - **trusted**: All network connections are accepted. Trust every computer on the network.

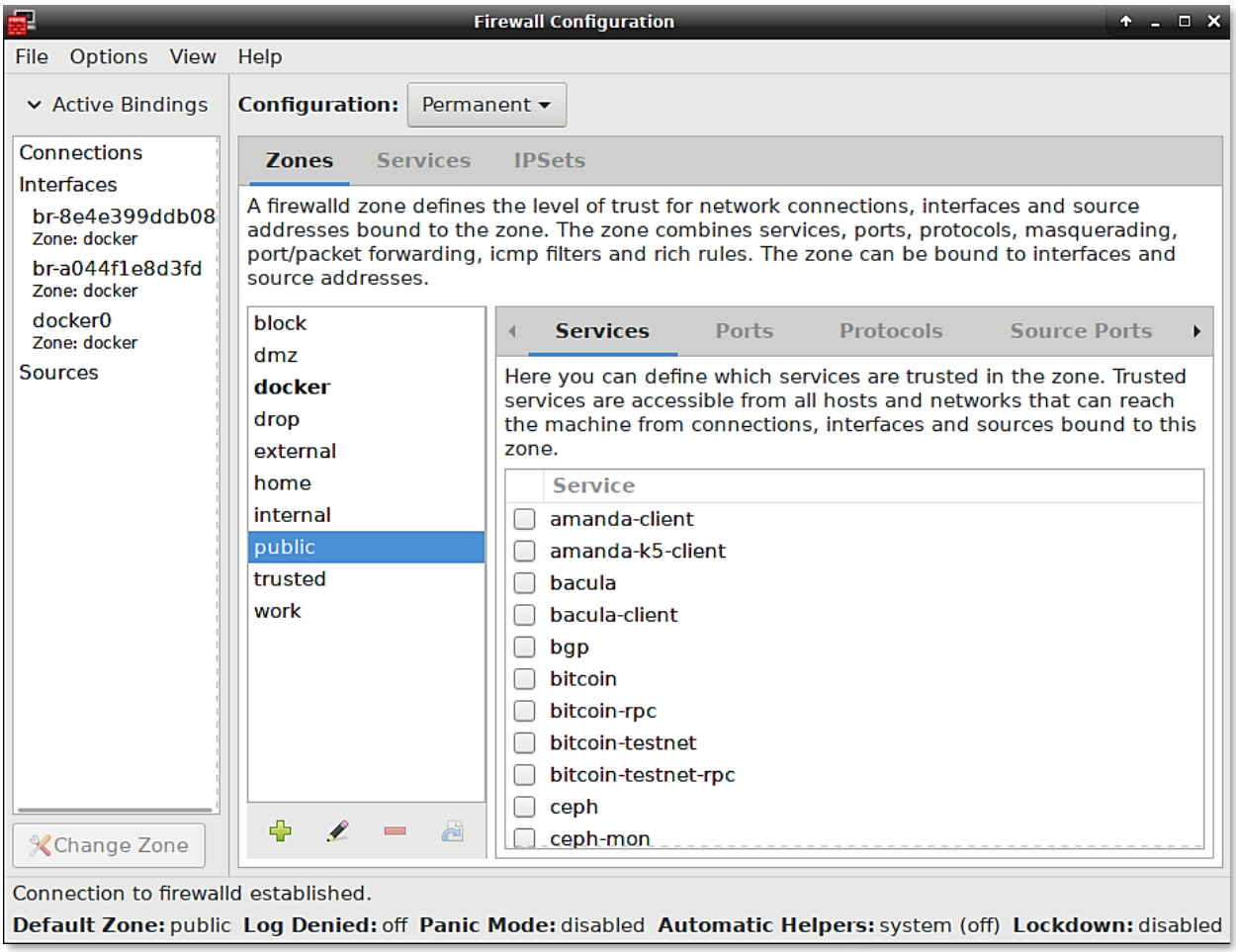
You do not have to understand all of these areas, but you need to know how to change the interfaces between them.

🔍 ***The good part about using zones is that once you enable a service or port for a zone, you enable it for all the interfaces that belong to that zone at the same time!***

We are going to use the *Firewall Config* GUI to manage most *firewall* operations, but you can also do the same things via the command line:

- https://computingforgeeks.com/install-and-use-firewalld-on-ubuntu/
- https://www.supportsages.com/everything-you-need-to-know-about-firewalld/

To do that, first run sudo `firewall-config` (remember to use `sudo`, please, or it will give you an error) and change the *ComboBox* from ***Configuration*** to ***Permanent***, as shown in this image:



Unfortunately, most of you will see that the `eth0` and `eth1` network cards are not assigned to any zone and do not appear in the GUI. We can fix this by using the command line, making these network cards appear, and being able to manipulate them using the GUI from now on. Do the following operations for that:

- Add `eth0` (the card that allows the VM to connect to the Internet) to the "*public*" zone: `sudo firewall-cmd --zone=public --change-interface=eth0`
- Add `eth1` (the card that allows the virtual machine to connect to your PC) to the "*work*" zone: `sudo firewall-cmd --zone=work --change-interface=eth1`

Once this is done, make sure that the `work` zone has the `ssh` service enabled and test that `ssh` is working from your PC.

Exercise L7B1_FWBLOCK: Blocking IP addresses and networks with firewalls

Description of the activity / Practical application

You need **to block individual IPs or entire networks** from accessing your machine

Expected results

This activity will end when you can **block an IP or network** from accessing a service in a *firewall zone*.

Additional information required to do it

One of the most common uses of a *firewall* is to allow/block access from certain IPs or networks to other IPs and networks. To practice how to do this, follow this procedure:

- Make sure you can access the virtual machine from your PC via `ssh`
- Go to the "`work`" area in the *firewall* GUI and look under the ***"Rich Rules"*** *tab.*
- **Create a new rich rule** that **drops** all connections to the `ssh` service from `IP 192.168.56.1` (usually your PC's IP on the host-only network) to IP `192.168.56.10` (your virtual machine's IP on the `host-only` network). Also, check all the options that rule creation has and think about its possibilities, especially the *limit, log,* and *audit options*.
- Prove that you can no longer connect via `ssh`
- Modify the previous rule to block the entire network `192.168.56.0/24`
- Prove that you still cannot connect via `ssh`
- Remove the rule and prove that you can now connect again.

Exercise L7B1_FWFORWARD: *Port forwarding in firewalld*

Description of the activity / Practical application

You need **to "hide" where a service really is** (the IP and port of its actual server machine) by redirecting its access through a *firewall*

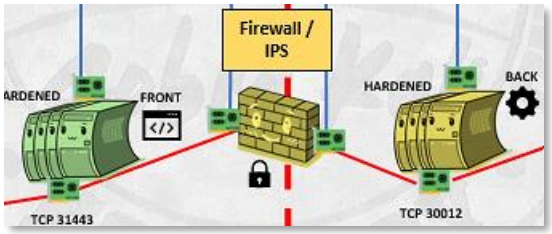
Expected results

This activity will end when you **can communicate a container in the public zone with a container in the DMZ zone so that a** web front-end placed in the DMZ can be accessed from the public zone using port forwarding. You can answer these questions:

- *Do you now understand how network isolation works in practice and the role the firewall plays in it?*
- *What would I have to do to be able to directly contact the machine that actually offers the service if I am an attacker?*

Additional information required to do it

The aim of this exercise is to emulate the following part of the SNI:



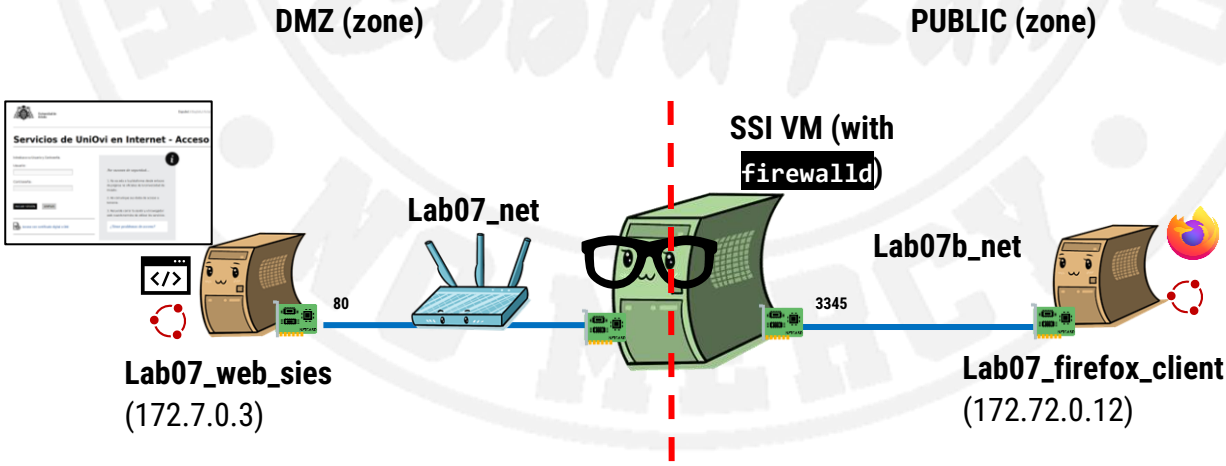
We have two machines located in different network zones and, therefore, cannot communicate directly. Between the two zones there is a firewall that can connect to both, and we can instruct the firewall to redirect a connection made to one of **its ports in one of the zones to another machine that is in a different zone: this is called** port **forwarding**.

🔍 ***The machine in one of the zones will not know who is the machine that serves its request in the other zone, and the machine that provides the service will never have direct contact with its customers: the firewall isolates both.***

To emulate this, we are going to use the following elements of our **Lab 7 infrastructure** (see diagram):

- Log in to the `lab07_firefox_client` container and write down its IP (`ifconfig`)
- Go to the VM and locate the card that is connected to the network that uses the *Firefox* container by looking for the one with a compatible IP that you wrote down in the first step. Its name should be something like `br-5c8ea19b05fb`
- Using the *firewall GUI*, place this network card in the "public" zone.
- Now, locate the network card that serves container `lab07_web_sies` (the one with an IP on the network `172.7.0.0/16)`, which should be `172.7.0.3`.
- Place this network card in the " *dmz*" zone.

This diagram represents the structure for this exercise:



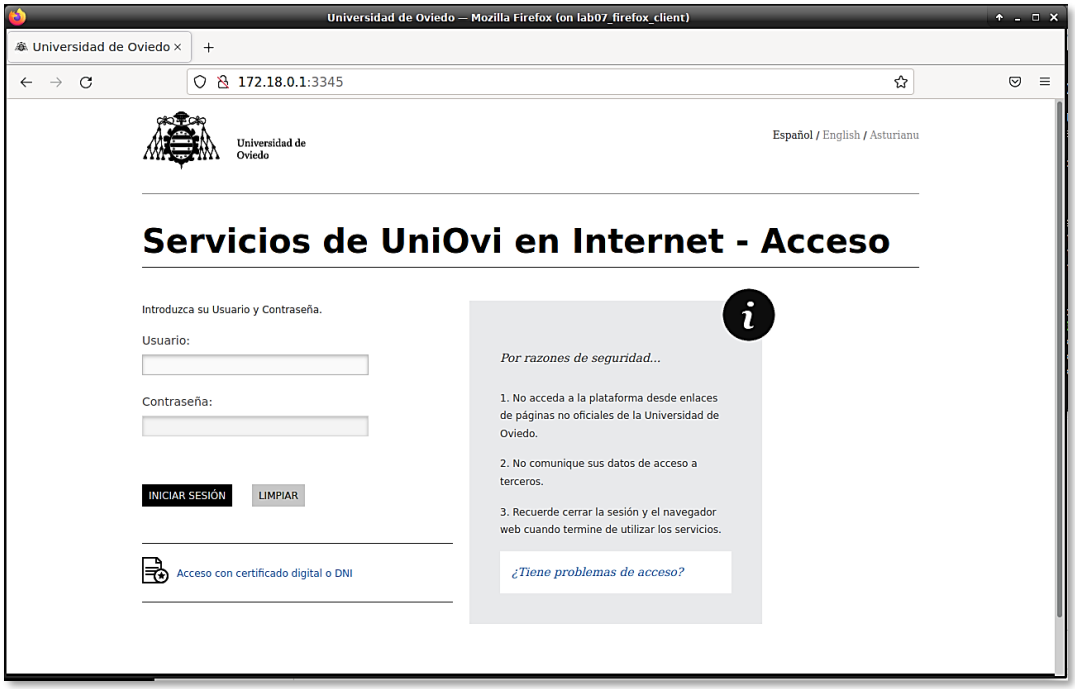
Now you have set up two containers in different zones and the *firewall* (your virtual machine) in the middle. To be able to communicate them using port forwarding, do the following:

- Go to the "*public*" area in the GUI and look for the "***Port Forwarding***" tab (click on the right arrow to see more tabs)

- Create a new port forwarding rule that redirects all connections from port `3345` to the port 80 of the IP `172.7.0.3` (the `lab07_web_sies` container). If the firewall asks you to enable masking, answer *Yes*.
- **Reload the *firewall* (*Options – Reload firewall*).**

To test this, you can run *Firefox* on the bash command line of the `lab07_firefox_client` container. The first time it will probably give an error, **but if you restart it, it will work.** You now have a *Firefox* isolated container in the "public" area. Browse to the IP of the VM in this zone (if the IP of the container is `172.22.0.2`, the IP of the VM on this network uses the same network but ends in `1`) but to port `3345` (Example: `172.22.0.1:3345`).

🔍 ***If all is correct, you should see this web page, which is just the one with the machine "on the other side" of the firewall hosted, but...you do not know its IP or the actual port you are accessing!***



With this, you have connected a machine on one network with a machine to another network through a *firewall*. Think about what this means related to security (who can access the second machine, what services are being exposed...) and how you could replicate this to connect a *front-end* of an application with its *backend* if both are placed in different network zones.

Block 2: Network protection on outbound and inbound connections

Exercise L7B2_DNSPI: Securing outbound connections: DNS filtering with *PiHole*

Description of the activity / Practical application

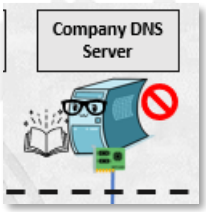
You need **to block connections to known malicious domains** using a local DNS that filters the requests your machine makes to external domains

Expected results

This activity will end when you can configure the ***PiHole DNS filtering server*** to test how your browsing changes once you use its features. You can answer the questions that are asked throughout the exercise and this one. *Which is the main difference between PiHole and the NextDNS you saw in **Lab 1**?*

Additional information required to do it

We strongly recommend that you create a snapshot of the VM before performing this exercise. The goal of this exercise is to show you the importance of controlling your own DNS server to filter out potentially malicious outbound traffic. This corresponds to this part of the *Secure Network Infrastructure* (SNI) theory, **but you can also use it at home** .



We will do this by setting up a professional product, *PiHole* (https://pi-hole.net/), which allows you to have a DNS server that also filters any request it receives using a ***blocklist*** approach: any connection made to an IP present in the block list will be aborted.

PiHole allows installation on many systems, but for this lab we have packaged it in a container. Go to the `pihole` folder of the lab files and run the `run_pihole.sh` script (please make sure that *Apache 2* is not installed on the virtual machine and do `sudo apt-get remove apache2` if it is). Once the command finishes running, navigate to the URL `127.0.0.1/admin` in the VM's browser to view the *PiHole* welcome stats screen. The admin password is `test123...`

We recommend that you now browse to any ad-supported website (such as www.elmundo.es) and leave that website open. Now *that PiHole* is up and running, to start filtering connections we just need to change the DNS of our `eth0` network card (the one that connects to the Internet) to `127.0.0.1` (PiHole runs locally). Do not worry, you will not lose your internet connection, *PiHole* knows what to do 😊. The next image shows the changes you have make for this, **although you will also need to change it in the /`etc/resolv.conf` file:**

```
 GNU nano 2.9.3            /etc/netplan/01-netcfg.yaml            Modified

network:
  version: 2
  ethernets:
    eth0:
      dhcp4: true
      nameservers:
                addresses: [127.0.0.1]
```

Reload the old website now. *Do you see any difference? What will happen now to any device that uses this DNS server?*

Exercise L7B2_DNSPIPLUS: Advanced filtering of outbound connections with *PiHole*

Description of the activity / Practical application

You need to block connections to even more malicious domains, **boosting *PiHole's* capabilities**

Expected results

This activity will end when you can upgrade ***PiHole's blocklists*** to block connections to more types of malicious websites.

Additional information required to do it

Although *PiHole* filters a large number of malicious websites out of the box, we can give it more "power" by using the malicious URLs collected by ***The Block List Project*** (https://blocklistproject.github.io/Lists/). Follow these instructions to add this list of IPs to *PiHole*:

- Copy the link from the most complete list (***Everything***) in *PiHole*-compatible format (https://blocklistproject.github.io/Lists/everything.txt).
- Add the URL to your *PiHole's block lists* (***Login - Group Management-Adlists -Paste the URL of the list into the "Address" field, add a comment - Click "Add")***
- Update ***Gravity*** (***Tools-Update Gravity-Click "Update")*** to add the new URLs to the block list. The process takes a while, **please wait, and do not navigate away from this page!**

Navigate back to any website with *PiHole's* stats webpage open and see how much larger blocklists are now up and running.

Exercise L7B2_FAIL2BAN: Protecting from intrusion attempts: *Fail2Ban*

Description of the activity / Practical application

You need **to block the connections of machines** that try to access your SSH service unsuccessfully multiple times

Expected results

This activity will end when you are able to protect `ssh` using `fail2ban`'s aggressive protection mode. You can also verify that the protection is working and revoke the IP blocks created by this program. You can answer this question: *What kind of attack on SSH do you think this tool particularly blocks?*

Additional information required to do it

Fail2Ban is an application that **processes *system* logs** to look for signs of attacks. When detected, according to the rules that are defined for it, `fail2ban` adds a new rule to the *firewall* that blocks the IP from which the identified attack attempt was made. This rule blocks the attacker's IP temporarily or permanently, depending on the configuration.

🔍 ***You can also email notify a user that an attack is underway (by installing `sendmail`, but we will not use it in this lab).***

`fail2ban` is typically used to **protect** `ssh` but has several default predefined rules (called ***jails*) that also work with other services, such as *Apache 2 or Nginx* (web servers).**

🔍 ***Technically, any network service that generates a log can be protected by `fail2ban` as long as it has the appropriate rules configured. To do this you can create your own rules, but that is beyond the purpose of this lab.***

This program does not work inside containers, so you need **to install it on your virtual machine.**

SSH protection using *Fail2Ban*

Using `fail2ban` with SSH requires you to follow these steps:

- Install it (the service starts automatically once installed): `sudo apt install fail2ban`
- Make sure that the services to be protected by `fail2ban` (SSH, HTTP, ...) are not blocked by the *firewall* (there would be nothing to protect then! 😊).
- Modify the `fail2ban` configuration (`/etc/fail2ban/fail2ban.conf` file). It is **HIGHLY** recommended to copy this file to `fail2ban.local` and edit this new file instead of the original `.conf`. In this file you can configure things like the level of detail of the log (`loglevel`), where to write the log information (`logtarget`), etc. We will use the default settings in this lab.
- Modify the /`etc/fail2ban/jail.local` file to enable or disable *jails*. To do that, do the following:
  - If the `jail.local` file does not exist, create it by copying `jail.conf`
  - **At first, only the `sshd` jail is active by default.** The rest must be manually enabled by adding `enabled = true` after each *jail* name you want to activate (the one between `[]`).
  - There are many other commands and parameters (such as IPs to ignore), but we will not be using them in this lab, as we want a simple (but effective) implementation of `fail2ban`.
  - When a jail is triggered, the `bantime`, `findtime`, and `maxretry` parameters define how an IP is deauthorized. A negative `bantime` means a **permanent ban. A host is blocked by a *jail*** if it successfully triggers its conditions more than `maxretry` times over the course of `findtime` seconds.
  - The available *jails* are defined in a section of this file. As we said, `sshd` is enabled by default, while the others must be enabled manually. Of course, this is only recommended if we really have the service running.

```
[sshd]

# To use more aggressive sshd modes set filter parameter "mode" in jail.local:
# normal (default), ddos, extra or aggressive (combines all).
# See "tests/files/logs/sshd" or "filter.d/sshd.conf" for usage example and det$
#mode   = normal
port    = ssh
logpath = %(sshd_log)s
backend = %(sshd_backend)s


[dropbear]

port     = ssh
logpath  = %(dropbear_log)s
backend  = %(dropbear_backend)s


[selinux-ssh]

port     = ssh
logpath  = %(auditd_log)s


#
# HTTP servers
#

[apache-auth]

port     = http,https
logpath  = %(apache_error_log)s


[apache-badbots]
# Ban hosts which agent identifies spammer robots crawling the web
# for email addresses. The mail outputs are buffered.
port     = http,https
```

- Each *jail* behavior is defined in a different file in the /`etc/fail2ban/filter.d` directory. Each of these files contains **the regular expressions that**, when matched to an entry in the *log*, trigger the rule. They also define modes of operation and other parameters. They do not need to be examined in this lab; this is just so you know how it works 😊
- For this lab **we will enable aggressive mode for `sshd` and ensure maximum SSH protection mode.**

Fail2Ban health check

`fail2ban-client` can be used to manage `fail2ban` through the following options:

- `start`: start `fail2ban`
- `reload`: Reloads `fail2ban` configuration
- `reload JAIL`: Replaces the current configuration of the *specified jail* with the one in the file being passed
- `stop`: Stops `fail2ban`
- `Status`: Shows the current status of `fail2ban` and *active jails (including their names)*
- `status JAIL`: Displays the status of the *specified jail*. It is a very important option as it also shows how many times it has been activated and the IPs currently blocked.

Every time you activate new jails in the previous file, you need to reload `fail2ban` and check how many jails are active.

Fail2Ban *testing*

Testing `fail2ban` with SSH is very easy: you just have to make several login attempts with incorrect passwords within the time you have set up. It is recommended to do this from one of the containers in the **Lab 7 infrastructure.** Once you have been blocked, **use the commands above to check that the IP is actually listed as banned**.

To unblock the IP, you can wait for the time you have specified (if not, it is infinite!) or run: `fail2ban-client set <jail name> unbanip <ip>`

Exercise L7B2_PORTSENTRY: Protecting scan attempts: *PortSentry* as a *honeypot*

Description of the activity / Practical application

You need to block connections from attackers who **are trying to scan you**

Expected results

This activity will end when you are able to **protect a host against** `nmap` scan attempts using `portsentry`, and understand the blocking behavior it performs, all without interfering with the machine's firewall. You should also **verify that the protection is working** and **undo the IP blocks** created by this program.

Additional information required to do it

While we already saw an example in the first lab, port scans with *Nmap* will be covered in depth in the next lab and are one of the most common operations in any *pentesting* activity. In this lab we will look at how to stop them using this tool. ***PortSentry* can detect port scans and block them**, so the machine you are scanning will not get any information.

There are several ways to use `portsentry`, but we are going to use **one of the most interesting**: combining it with the *firewall* we deployed in a previous activity to **emulate the behavior of a *Honeypot***. To do this**, we will purposely leave some ports open in the *firewall***, even though there is no service listening to them. Scan attempts through these "falsely open" ports will be detected and the tool will proceed to ban the machine doing the scanning. Note that `portsentry` cannot listen on ports that have actual services that are using them, as they "occupy" the ports. Follow these steps:

- Install the *PortSentry* service on your *Ubuntu* VM (containers are not supported): `sudo apt install portsentry`
- Verify that it is running: `sudo service portsentry status`
- Understand the tool's two working modes (`TCP_MODE` and `UDP_MODE` parameter values in the `/etc/default/portsentry` file)
    - **Basic mode** (values: "`tcp`", "`udp`"): Listens for a static list of predefined ports in the configuration file. **We will not** use this mode as the presence of the tool can be easily detected.
    - **Advanced *stealth* mode** (values: "`atcp`", "`audp`"): It uses a *raw socket* to detect scans and so the tool cannot be easily detected: the ports appear to have the "normal" service on it, but it is `portsentry` that is listening and detecting suspicious activity.
- Understand the **possible responses** to a scan attempt (`BLOCK_UDP` and `BLOCK_TCP` values in the `/etc/portsentry/portsentry.conf` file)
    - Only log **suspicious traffic in** `/var/log/syslog` (value: "`0`") (**default**)
    - **Block the "offsetting" machine** (value: "`1`")

- **Run a program as a** response (value: `"2"`). This gives us a lot of flexibility and allows for "wild" responses to scan attempts , but we will not use it here.

Log and block analysis attempts

The first thing to do to enable `portsentry` is to enable the following services in the `docker` firewall zone: SSH, FTP, and Telnet.

> 🔍 ***Thus we have a "bait" to attract scanning attempts to our server.***

Once this is done, open the /`etc/default/portsentry` file and set `TCP_MODE` to "`atcp`" and `UDP_MODE` to "`audp`" to enable advanced stealth mode**.** Restart `portsentry` and perform a simple `nmap` scan from any of the **Lab 7 infrastructure** containers to the host machine (`nmap <IP>`). After the scan is complete, check the contents of `/var/log/syslog` to see if scan attempts have been detected.

Successful *logging* of scan attempts is the condition for finishing configuring `portsentry` as expected: Open `/etc/portsentry/portsentry.conf` and change the parameter values `BLOCK_UDP` and `BLOCK_TCP` to `"1"`. Restart `portsentry` and perform the same scan again.

Revert blocks

To undo a block, you need to do this:

- Remove the IP of the blocked machine from `/etc/hosts.deny`. **Being in this file prevents any communication from this IP to our machine**.

```
  GNU nano 2.9.3                    /etc/hosts.deny

# /etc/hosts.deny: list of hosts that are _not_ allowed to access the system.
#                  See the manual pages hosts_access(5) and hosts_options(5).
#
# Example:    ALL: some.host.name, .some.domain
#             ALL EXCEPT in.fingerd: other.host.name, .other.domain
#
# If you're going to protect the portmapper use the name "rpcbind" for the
# daemon name. See rpcbind(8) and rpc.mountd(8) for further information.
#
# The PARANOID wildcard matches any host whose name does not match its
# address.
#
# You may wish to enable this to ensure any programs that don't
# validate looked up hostnames still leave understandable logs. In past
# versions of Debian this has been the default.
# ALL: PARANOID

ALL: 192.168.8.100 : DENY
```

- Remove firewall rule that rejects (`REJECT`) all offending client traffic: `sudo route of the -host <IP of the offending client> reject`
- We can use `netstat -rn` to verify that the rule has been deleted.

Block 3. Intrusion detection

Exercise L7B3_IDSMALTRAIL: Installing and testing the IDS *Maltrail*

Description of the activity / Practical application

This activity teaches you how to **install and use an IDS called *Maltrail***, so you can see how a network IDS or NIDS works.

Expected results

This activity will be considered complete when you have **an operational *Maltrail*** installation capable of detecting issues such as port scans. You can answer this question: *Do you understand how a well-configured IDS can be used to locate suspicious activity coming from other people's machines?*

Additional information required to do it

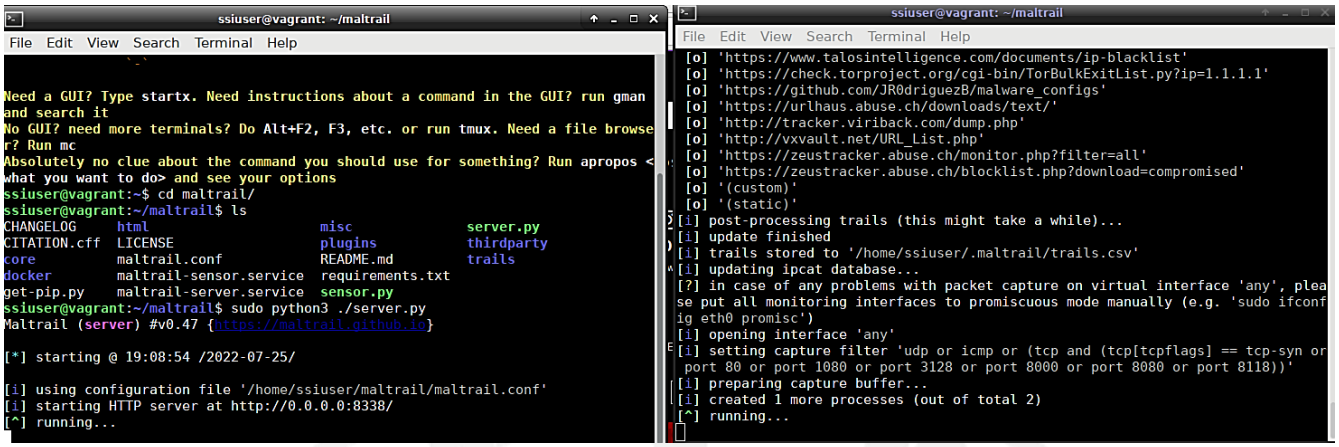
Snort (https://www.snort.org/) is **a reference open-source Intrusion Prevention System (IPS).** The problem is that its installation, commissioning and administration **is complex** (https://snort-org-site.s3.amazonaws.com/production/document_files/files/000/011/074/original/Snort_3_on_Ubuntu_18_and_20.pdf), and therefore it is an investment for large infrastructures that require this level of monitoring. Another tool that can be used for the same purpose is ***Suricata*** or a professional monitoring and security distribution such as ***SecurityOnion***. However, both are resource-intensive and complex to manage, just like *Snort*.

Because of this, in this activity we are going to look at **a much simpler IDS that has a user-friendly interface** and fulfills the same function: *Maltrail*. *Maltrail* is a self-contained malicious traffic detection tool that **runs on a monitoring node** through which all traffic on a network is supposed to pass (remember what kind of machines comply with this in our theory SNI, such as a ***reverse* proxy**).

This IDS uses public block lists of suspicious web addresses and/or *malware*, along with static addresses collected by various antivirus reports to detect potential threats. In case they are detected, the event details are sent to the server and stored in a *log*. However, this occurs if the component called "**sensor**" is running on a different machine than the component called "**server**".

To simplify, both components can reside on the same machine, as we will do in this activity.

Maltrail can be installed on any *Linux* machine, and for this it is best to follow its official installation tutorial: https://github.com/stamparm/maltrail, although one of the prerequisite packages listed in this tutorial is not found in *Ubuntu*, and the final prerequisites are: `sudo apt-get install git python3 python3-dev python3-pip libpcap-dev build-essential procps schedtool`. Once the installation of both components on the same machine is finished, and as indicated in the tutorial, we start the sensor and the server:



And with this, we will be able to access http://localhost:8338 to access the interface. To test the IDS, simply perform a `nmap` scan from *the host* to the virtual machine that has *Maltrail* operating, and after reloading its web interface it should detect a possible port scan as malicious activity.

Exercise L7B3_WAZUHSCAN: *Wazuh* and network attacks

Description of the activity / Practical application

Consist on seeing **how *Wazuh reacts*** to a network scan or a failed SSH access attempt

Expected results

You can see how the *Wazuh* XDR from previous activities reacts when facing the same attack cases covered by ***Fail2Ban*** (repeated access attempt failed via SSH) and ***PortSentry*** or ***Maltrail*** (scanning ports with `nmap`). You can answer these questions:

- *Does Wazuh block these activities, or does it just report that it has detected them?*
- *Does Wazuh detect the above activities in the same way as the rest of the specific applications, or does it do so in a less obvious way?*
- *What difference do you see between the information Maltrail and Wazuh gives you after a `nmap` scan?*
- *Do you understand better now the difference between a NIDS and a HIDS?*

Additional information required to do it

This activity is simple, since you only have to **start the "MV *Wazuh*"** from previous activities and, from your host, make an `nmap` and try to **access several times via SSH** with wrong username / password to the MV of the course. Once this is done, examine the information captured by *Wazuh* in order to answer the questions asked in the exercise.

Block 4. Increasing connection security: VPNs

Exercise L7B4_VPN: Using traditional commercial VPNs: *ProtonVPN*

Description of the activity / Practical application

It consists of using the *ProtonVPN* commercial VPN in its free version to **familiarize yourself with a traditional business VPN service.**

Expected results

You can use *ProtonVPN* and connect to any available free remote server, checking that the internet connection is still available and that **your IP has changed** to another one, geolocated in the country of the remote server. You can answer the questions asked in the exercise text.

Additional information required to do it

The first thing you need to do is test what your current IP is, and your geolocation based on it. You can do this on several websites on the Internet, but one of the most typical is https://www.whatismyip.com/. Save or write down the result for later comparison.

If you have decided to use *ProtonMail* in **lab 4**, the account you created will be useful for this exercise. Otherwise, you can **get a free account on *ProtonVPN*** (which will also work for *ProtonMail* if you want to use it 😊) at this address: https://protonvpn.com/es-es.

🔍 ***Actually, any VPN service would be valid to do this exercise, but we have opted for this one because it has a good reputation and offers 200 or so servers in countries in Europe, America, and Asia for free to test it from a device.***

Another reason to opt for this service is that, although we cannot enjoy these features on the free plan, it does clearly show how a VPN **can be a security system** not only for the connection privacy, but also preventing you from browsing malicious content. *Can you find what exactly does it block?*

Before you get started, it is important to keep one thing in mind. The December 2023 tested version of *ProtonVPN*'s VPN client installs on *Ubuntu Server XFCE* (our VM), but unfortunately **it does not connect to any** servers. A *desktop version* of *Ubuntu* should fix that problem, but we are not going to install it just for this exercise. Therefore, the simplest solution is to use **a linked clone of your Windows VM *from our "sister" course*, *Server and Network Administration***, since the *Windows* client works without any problem.

🔍 ***We ask you to use a linked clone so that you do not affect ASR's work and so that you can create it at minimal cost, but of course you are free to use any other option!***

However, *ProtonVPN* has **clients for a large number of systems** (*Android,* iOS*, Windows*, MacOS, GNU/Linux*, Chromebook*, and *AndroidTV*), so if you like the product, you can use it on anyone that suits you best, as it is the same on all of them.

If you already have experience with VPNs and have a client that supports ***OpenVPN***, it also has configuration files for them. However, in this exercise we assume that you have no experience, and that you will be using *ProtonVPN*'s native client. Once the client is installed, we validate ourselves on it with the *ProtonMail/ProtonVPN account data* that we already have/have created. If everything is correct**, we should see this screen**, where we are shown the countries we can connect to on the map and in the bottom left list. We select one (better the European one) and connect.

Once this is done, answer the following questions:

- *Does the IP on the VPN screen change when you connect?*
- *If you now browse to the address to find out your IP, ISP, and previous geolocation, does it change as well?*
- *Do you understand that you are now that machine and use the ISP of the chosen country for any site you connect to? What do you think would be a way to detect that you are in another country while browsing normally (and a possible usability issue)? What do you think this is for?*
- *Do you understand that all traffic from your VM to the VPN server you have connected to is encrypted using the combination of asymmetric and symmetric encryption we saw in* ***Lab 4****? What does this guarantee you then?*
- *As a result of the above, do you understand why using a VPN is a way to be safe even if you use public Wi-Fi (cafeteria, airport...)?*
- *Can you locate the security features included in the non-free version and understand what they would be used for if you could activate them? Concretely:*
    - *Imagine that you have connected via VPN with the Wi-Fi of a bar. How could the kill switch save you from having your data stolen?*
    - *Imagine you have your own Minecraft server on port 25565 and you want to play with your friends but with the VPN active. Could you use it at the same time as ProtonVPN with any of its options? (Remember the exercise L7B1_FWFORWARD)*

Exercise L7B4_TORBROWSER: Install and test *ToR Browser*

Description of the activity / Practical application

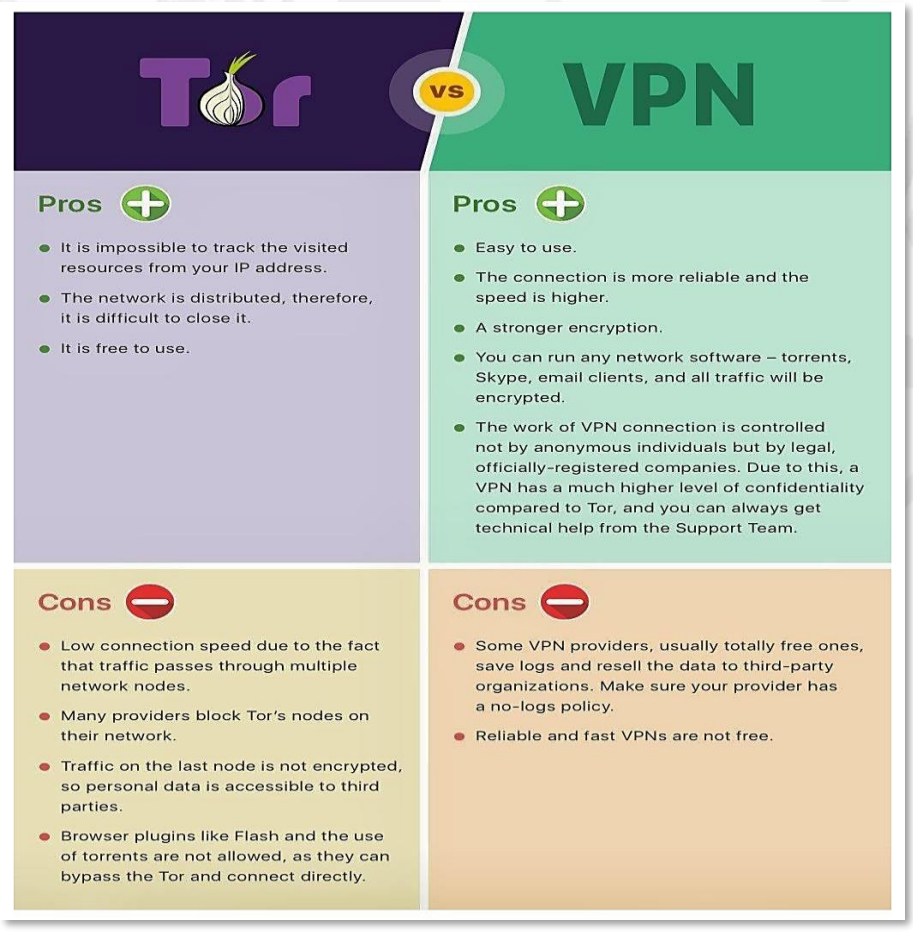
It consists of **installing the *ToR Browser*** and using it to browse different websites to check the advantages and disadvantages of the **anonymity** provided by the ToR network, as well as testing an `.onion address`

Expected results

You can launch **a functional installation of *ToR Browser*** and use it to navigate through the ToR network, understanding the advantages and disadvantages of doing so. You can answer these questions and the ones that are asked during the exercise statement:

- *Do you know the difference between Deep and Dark web and what they both are?*
- *Do you understand what exactly an* `.onion` link is *and why they are used to access the dark web?*
- *Is browsing speed suffering?*
- *Is there any effect of browsing websites hiding your identity and, therefore, the country from which you are actually browsing?*
- *Do you understand what the purpose of an* `.onion` *link is and why it can take you to Deep and Dark Web sites?*

The ultimate goal of the exercise is for you to check a few things in this diagram for yourself (**Source**: https://www.reddit.com/r/Tunisia/comments/xj3k6r/tor_vs_vpn/)



Additional information required to do it

Before doing this exercise you need to familiarize yourself with 3 concepts.

What is the *deep web*?

The "*Deep Web*" is the part of the Internet **not indexed by traditional search engines**. You will not find any of its content on *Google*, *Bing*, etc. It does not matter why they are not; they just do not show up

> 🔍 ***It is not all illegal websites (which obviously there are too), there are also places to consult business or government databases, private websites of groups or organizations (although basing its privacy only on not being indexed is a wrong decision)***

In the literal sense of the definition, *deep web* is also your *Dropbox* or *OneDrive*, for example. Or, in general, content that a search engine cannot access because it requires access credentials. When, in addition to those credentials, **an access method is required that requires anonymity** from the person accessing it, then we speak of the *Dark Web*.

> 🔍 ***Obviously, the Dark Web is part of the Deep Web, as its contents are not indexed either***

It is important to mention that **not all content on the *Deep Web* is illicit**. Many legitimate and useful sites are found on the *Deep Web* because they require login credentials to access them. However, due to the limitations and complete anonymity of the *Dark Web*, it is unfortunately often a magnet for criminal activity.

What is the *dark web*?

The *"Dark* Web" is a part of the *World Wide Web* that can only be accessed through a special browser **to access websites that use the** `.onion` domain ending . It is characterized by (theoretically) absolute anonymity and special access restrictions. This level of privacy and security attracts individuals and criminal groups, which has led to the *dark web* being associated with forbidden content and shady activities.

> 🔍 ***Although the mere use of the Dark Web and related technologies is not punishable or illegal, from the moment you enter markets and forums for weapons trafficking, drugs, or the like, and establish a business relationship with the sellers of these sites, you are committing a crime. Browsing there is by no means a joke, and not only are there all sorts of illegal (and extremely disturbing) content and services, but browser exploits, malware, and other threats***

It is important to mention that despite its reputation, the *Dark Web* **is also used for legitimate purposes**. For example, it can be a haven for people living under oppressive regimes and looking for a safe way to communicate or access information. However, as a general rule, it **is not a place for average internet users to entertain themselves or browse**. Anonymity brings out the worst in humanity, and real atrocities 😟.

> 🔍 ***In this course we prefer to use a transparent approach to this topic. It exists, and accessing it is easy if you have the tools. But do not fool yourself: the worst of humanity is there, and you can find it even by accident...***

What is a `.onion` link?

An `.onion` link is a type of web address that corresponds to an anonymous server accessible through the Tor network mentioned in the theory. **These links are used on the *Dark Web*** and are known for their high degree of anonymity.

> 🔍 ***.onion addresses do not work like a DNS. Web browsers can access .onion sites using proxy programs and sending the request through servers on the Tor network***

These addresses are opaque and not mnemonic. These are manually generated combinations of alphanumeric characters. The name "*onion*" refers to the onion *routing* **technique used by *ToR* to achieve a high degree of anonymity**.

> 🔍 ***It is important to note that when using a proxy to access .onion sites, the proxy operator can spy on the source traffic and IP. If the intention is to be anonymous, proxies should not be used***

But to understand everything related to `.onion` links and the ToR network, you need to familiarize yourself with ***the ToR Browser***, which is the ultimate goal of this exercise.

Testing *ToR Browser*

You can install *ToR Browser* on your *Ubuntu* VM by downloading it from here: https://www.torproject.org/download/ and following the installation instructions at this link: https://tb-manual.torproject.org/installation/. It is advisable that you download the executable and the file with its signature, and follow the signature verification procedure that you have documented here: https://support.torproject.org/tbb/how-to-verify-signature/. *Do you understand the items you need based on what we have seen in **labs 3 and 4**?*

Once you extract and run the browser, you will see that the first thing it allows you to do is connect to ToR to browse, because if you do not then you will browse normally. However**, before doing so, you should adjust the settings as follows just in case**:

- Go into your browser's settings and **set it to "*Safer*" or "*Safest*",** depending on whether you are going to access well-known websites or not.

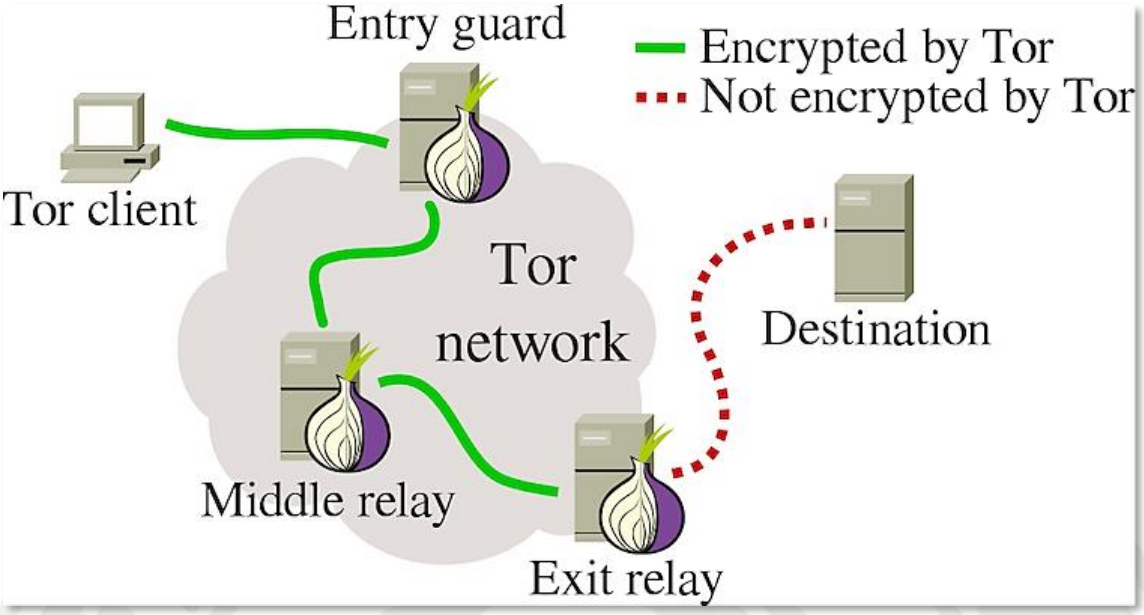
> 🔍 ***Safest may cause some websites to not work (blocks JavaScript)***

- **Install the Ad Blocker** recommended in the **Lab 2** exercise **L2B4_NAVEGAR**.

> 🔍 ***ToR Browser is built with the Firefox engine, so it uses the same plugins.***

- Now connect to the ToR network and use the address from the previous exercise to **know your IP and geolocation**. *Where are you?* Open the VM's *Firefox* and do the same. *What is going on? Are all VM services routed through ToR or just this browser?*
- If I tell you that **the route taken to get to that machine (which is called** the "exit node"**)** changes from time to time and **that those intermediate nodes only know the previous and the subsequent ones in their "chain"**, *do you understand why your connection is anonymous within the ToR network?* Take a look at this image (**Source**: https://theconversation.com/tor-upgrades-to-make-anonymous-

publishing-safer-73641) to get a better idea, but assume that the "path" from your *ToR* client machine to the output node typically has **more than two nodes** in between



- If I also tell you that **the exit node also changes over time**, *do you understand anonymity also from the point of view of the destination you connect to?*
- Try navigating to *Wikipedia*, a newspaper, etc. *Can you? Is there a decrease in connection speed?*
- Try now to browse to the website of the University of Oviedo or to that of any bank. *Can you? Can you access that page from the VM's Firefox?* If I tell you that the exit nodes of the ToR network **are known at all times**, *why do you think this happens and one browser works while the other does not?*
- *Do you think that the reason this happens has something to do with the **ToR – Dark web – Anonymity** relationship?*
- *What if you try to use Google from ToR Browser? Do you understand why DuckDuckGo shows you by default? What is the main feature of this search engine?*
- *What do you think would happen in the above cases **if you activated the ProtonVPN VPN and then browse via ToR Browser**?*
- *What if, instead, you browse via ToR to a remote machine (not linked to your identity) that is the one with the VPN active? What would you get in both cases? Do you now understand why some criminals "squat" other people's machines?*
- *If I tell you that VPN providers have a legal obligation to keep logs of who connects to their services from each site for X amount of time for police investigations, do you understand the difference between using a VPN and using ToR now that you know roughly how it works?*
- *What do you think the Tor Browser identity reset option is for? What kind of PCs do you think it would be most useful on?*

Testing a *.onion address*

Previously we talked about how `.onion` addresses are often used to **hide websites from criminals**, but they also have uses to **avoid censorship** of access to certain sites and social networks. One company that allows this is *Meta* with its *Facebook* `.onion` address which you can find here: https://en.wikipedia.org/wiki/Facebook_onion_address.

Copy the address and browse to it in your *Tor Browser* connected to ToR. *What comes out? What if you do it in a standard Firefox? Do you now understand how to access any* `.onion` *link you have? Do you also understand*

*that, like any link shortener, you have **to trust 100% of the source** because there is no way to know what is really behind it?*

> 🔍 ***You do not have to log in to Facebook at all, just get to the login***

Exercise L7B4_VPNP2P: Using P2P VPNs: *ZeroTier*

Description of the activity / Practical application

It consists of using the commercial P2P *VPN ZeroTier* in its free version to **familiarize yourself with a P2P VPN service and the main differences with a traditional one.**

Expected results

You can **set up a *ZeroTier P2P VPN* between two VMs** and check that both are communicating with each other as if it were a local network. You can do this **with two MVs you own or a colleague's VM**. You can answer these questions and those that arise throughout the exercise:

- *Do you understand how this system gives you control of the VPN, and you do not rely as much on a server as in the case of ProtonVPN?*
- *Do you understand how you can use this service to securely connect from a PC in the lab to your home?*
- *Do you understand that even if you do not have permissions to install programs on a PC in a lab, you have them in the VM, and you can therefore communicate the VM with your home PC without problems if both have a ZeroTier client?*
- *Can you think of how to combine the use of a firewall with ZeroTier so that you can only access services from the network you set up with ZeroTier?*
- *Do you now understand that you can leave services (Remote Desktop, shared folders...) on a machine in your home without exposing them on the Internet, only accessible to the machines behind your router (you never open ports on the router for them, so they are not visible "from the outside"), and that thanks to ZeroTier you can access them from anywhere?*

Additional information required to do it

In this section we are going to try to explain how to learn how to use **a P2P VPN service called *ZeroTier***, which would allow you to access any device from anywhere in a decentralized way (without depending on a central server like *ProtonVPN)*, secure and easier to install than your own VPN.

> 🔍 ***This P2P VPN is ideal for implementing telework in our future company, or connecting to our home having complete control of the entire service that provides it and, normally, for free because you will not need much infrastructure in these cases***

Traditionally, as we saw with *ProtonVPN*, when using a VPN service, it is assumed that a client's need is **to impersonate a machine that is in another location**, which is the one that actually accesses the remote service, and thus bypass some type of regional block or restriction that exists depending on where the client connects. This use of VPNs is traditional and very popular, but it has nothing to do with the one we are going to look at now.

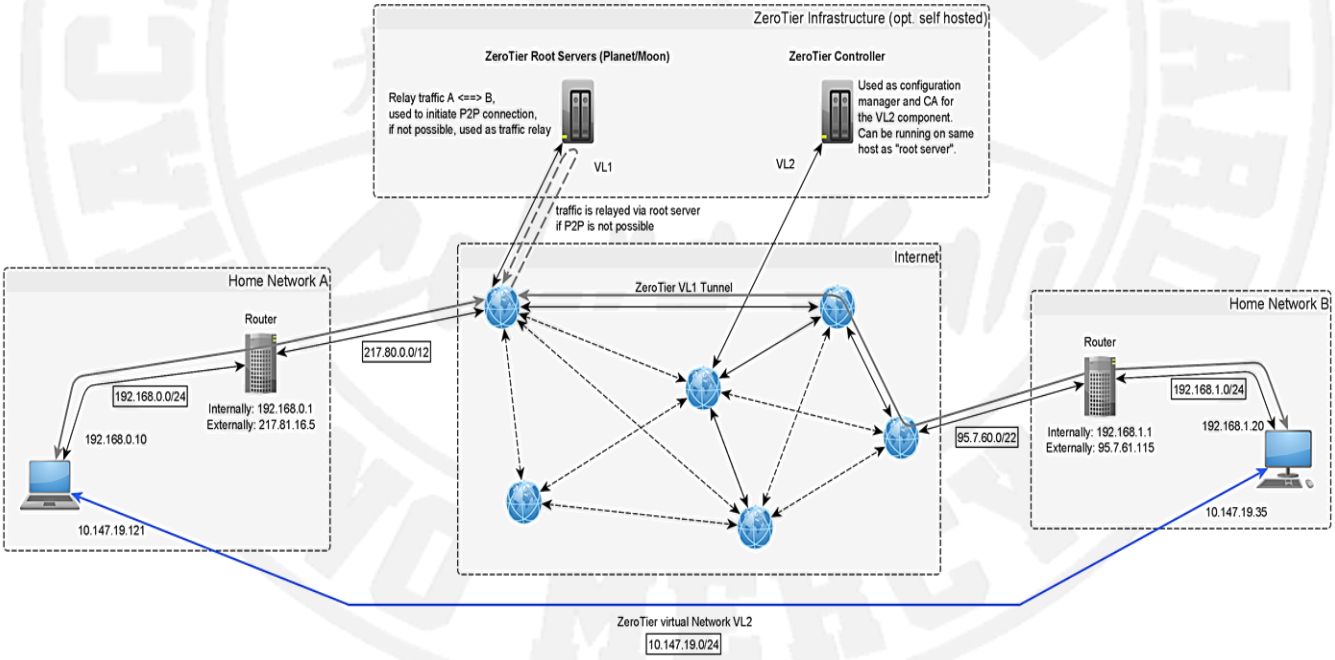
In the case of a P2P VPN**, you do not connect to a third-party server** (your VPN provider) to access remote services offered by another company. Now the remote service is really a machine that you have in your home or office. In other words, it is a way to connect **from a device that you own,** or can access to, to **another device that you also own,** or have access to, over the Internet and securely.

And to be able to make a private network with all these devices.

On the other hand, if we were to use a third-party VPN service to access a machine we own, we would be transferring personal data to that service, and we would also have to configure that machine to be **visible over the Internet** in a way that is correct for that VPN service. We do not expose where we connect from, but we do expose what we connect to. We could configure it to only allow connections through this third-party VPN, but it is something more complex that is beyond the scope of this course.

If this option is not our goal, the other option, which **is to install our own VPN server** (e.g. *Wireguard,* https://www.wireguard.com/) is technically viable, but also complex and slow especially if we authorize many devices. Therefore, it is a possible solution, but less practical in our specific scenario.

Peer-to-peer VPN **services** like ***TailScale*** or ***ZeroTier*** are the ones that solve this problem. Thanks to them, we can install their *software* **on each of our devices that we want to use from remote sites** and, in this way, instead of depending on a central server, form a network of devices that can see each other as if they were on a local network, regardless of where they are in the world. The following image (https://blog.rico-j.de/zerotier-one/) shows how this kind of VPNs work



This is the main advantage of these technologies that, by eliminating the central point, it removes availability and performance problems. We also get rid of the problem of making the service visible to the outside, because usually the client is in charge of doing all the management for us

Therefore, we would have a simple way to **share resources that we have in a secure way**, minimizing the risk of someone unauthorized being able to access them. This is because, unless some type of vulnerability is discovered in the service, only we will be able to establish a connection with it through the internet, which is totally private and inaccessible to unauthorized people.

Please never leave a Remote Desktop visible on the Internet. This will eventually make you the victim of an attack

The *ZeroTier P2P VPN service*

ZeroTier (https://zerotier.com/) is a company that provides a free and open-source *P2P VPN service* (also called a *"universal* switch") **that is easier to set up** than a traditional VPN. Its servers are responsible for putting the devices that are going to communicate in contact, and **it is these that create an encrypted tunnel between them**. From there, they maintain **end-to-end encrypted communication**. In this way, in addition to increasing the confidentiality of communication, latency is reduced.

It is important to note that ZeroTier puts the weight of communication on our devices, with the privacy advantages that this entails (our traffic does not go through the servers of any company)

With *ZeroTier,* you will be able **to create a VPN for up to 25 devices** for free, more than enough for personal use. Building larger VPNs requires a non-free plan (https://zerotier.com/pricing/). In addition, it is compatible with many devices (*Windows,* Mac*, Linux*, Android , iOS...*),* so you can incorporate the VPN and have secure access to a lot of hardware you already have.

ZeroTier installation

To install *ZeroTier*, the first thing you have to do is create a free account here: https://my.zerotier.com/login. Registration **demands a real email account** because it requires verification. Once registered, we will see the welcome screen. In it, you will see the "***Create A Network***" button to create a VPN, but you will already have one created beforehand that is perfect for testing. We can create multiple VPNs if we want. *What do you think this would be good for?*

Each *ZeroTier* **network has a unique 48-bit identifier** (which will need to be used later for devices to link to the created network. In addition, it is assigned a random provisional name, but it is convenient to change it to another one that makes sense to us.

It is important to leave the network as "PRIVATE" so that any device that joins it **will have to** be approved beforehand. Public networks do not require approval to connect, and they are for other scenarios. There ***are more options*** in the *Advanced* section, but **with the default settings it should work without further work**.

ZeroTier automatically selects a range of IPs for the VPN to be created. Notice that these are private IP addresses, such as those mentioned in Topic 1 of theory

The ***Members*** section is where you authorize the devices you want to have active on the VPN. You will also see data from the last time a device was connected, among other information, which is very useful for detecting potential intruders or unauthorized use. Initially, however, a network will appear without added devices, of course.

The "***Flow Rules***" section allows us to configure a **firewall *and its rules*, which allows us to** decide the type of traffic we want to admit or not on the network (protocols, ports, etc.). This is an advanced use, which allows us **to implement an extra layer of security** in our VPN if necessary. The setup process is not trivial and requires studying how to create the rules, but it has **help built into the application**. For this exercise we can leave the default settings.

Keep in mind that if we limit traffic in a network to one type of protocol or port, because it is the only thing we share in it, we are eliminating several possible attacks on our devices at once

Add devices

To add devices to the newly created network**, we have to install the *software* that controls the VPN on each of them**. We have clients for the devices we mentioned earlier here: https://www.zerotier.com/download/. For *Linux* distributions we have to install the software that acts as a client using one of the methods indicated on the *ZeroTier website*. The entire process of installing and adding each client to the network **is done under the command line**.

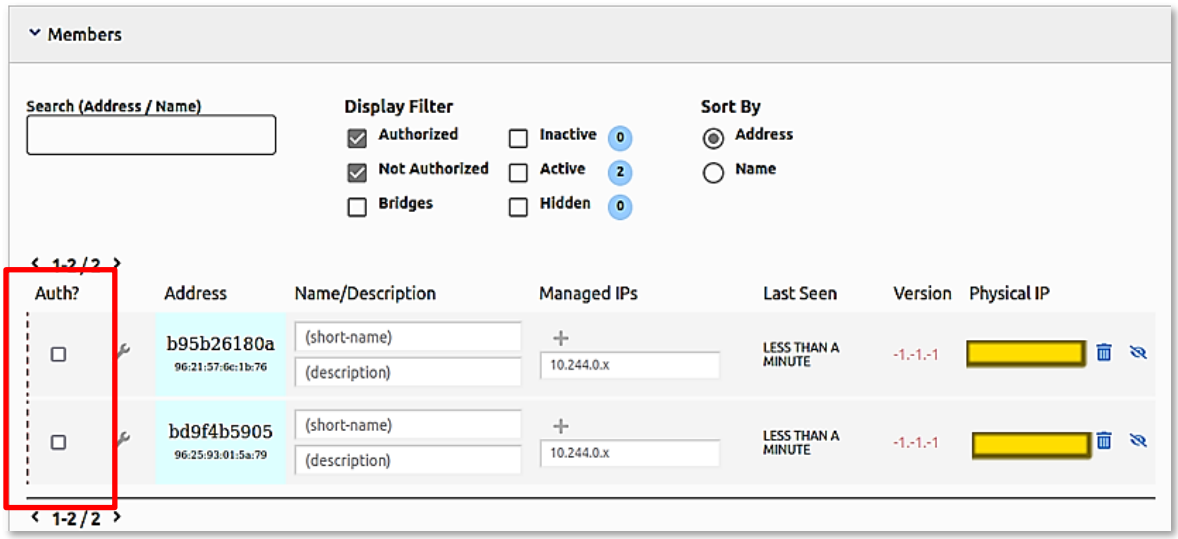
There's also an official Android app if you want to use your phone as a client: https://play.google.com/store/apps/details?id=com.zerotier.one

Once the client has been installed, from *Ubuntu*, on each of the devices that are going to be part of the network, we must do two things beforehand:

- Make sure **curl** is installed: `sudo apt-get install curl`
- Run `sudo zerotier-one -d` to boot the service on each node to be added to the network

Now we can add each of the devices that are part of the network ***with*** `sudo zerotier-cli join <the full network ID that we got on the ZeroTier website>`. The rest of the options can be left as they are. Since the network we created is private and we have used the device to ask for being admitted into it, the *ZeroTier* administration website will show the **devices that have joined our VPN**. It is now that **we must accept** that they connect in order to continue working.

How is it possible that the devices were simply added? Because they knew our network ID. If someone finds out (we should not reveal it to anyone else) they can try to connect a device, but that device will only be able to see the others on the network if we check its authorization box, as we will see shortly



If we click on the column "***Auth?***", we will give access to the devices we want and we can use them on the network, being able to see each other in the same way as if they were on a local network.

🔍 ***However, it is advisable to rename them so that they can be easily identified when the network grows in size.***

By doing so, from that moment on, *ZeroTier* **will assign an IP** from the private range we selected earlier to the client device and then it will be able to use it. To add more devices to the VPN, so that you can establish a connection between all of them and prove that a peer-to-peer VPN really does its job, you have to repeat the client installation process for each device.

To carry out the tests, you can try pinging between devices using the IP assigned by *ZeroTier* to each of them or try to access a service that one member offers (for example, a website, remote desktop...) from other network members. If it works, we have done everything right. Needless to say, by being able to choose **the range of IPs, we can limit access to the services offered by the devices only to IPs in that range with *the firewall*** of each device (such as the `firewalld` in block 1), which makes access to the services even more secure.

🔍 ***ZeroTier can work inside virtual machines that work with a NAT connection to egress the Internet. This means that you can connect to within a VM of yours from anywhere in the world, so not only are you securing the connection but, also, if your device is compromised and the attacker can use it to connect to your P2P VPN, their actions will be contained within the corresponding VM, greatly limiting their ability to cause problems***

Badges & Self-Assessment

NOTE: You have a version of this badge table in editable format available on the *Virtual Campus*. You can use this file to create a document in any format you want and take extended notes of your activities to create a log of what you have done. Remember that the material you make can be taken for laboratory tests.

Badge Level	Unlocked when	Unlocked?
	You know how to allow or deny services/ports from a *firewall*	
	You know how to add network interfaces to firewall zones	
	You can allow or deny connections from individual IP addresses or networks	
	You can answer this question: *What kind of attacks could potentially mitigate if a firewall puts a request/time limit on a service?*	
	You can answer this question: *What do you think is the purpose of logging/auditing attempts of explicitly forbidden connections in the firewall?*	
	You can answer these questions: *Why do you think it is useful to block machines from performing scans?* ***HINT****: Do you scan machines as part of your normal routine?*	
	You understand how to perform port forwarding and how to connect machines in different network segments separated by *firewalls*	
	You understand why blocking IPs via DNS is useful. You can answer this question: What *is the advantage of using PiHole to block IPs instead of a browser add-on that blocks malicious IPs?*	
	You can answer these questions: *Do you think* `fail2ban` *replaces a firewall, or does it complement it?*	
	You can answer this question: *What kind of attacks does* `fail2ban` prevent?	
	You can answer this question: *Do you think a temporary block is enough to prevent a DoS attack attempt?*	
	You can answer these questions: *Why do you think purposely leaving ports open can be a useful scan detection measure? Logically, what ports would you leave open in this way?*	
	You can answer these questions: *What happens when a machine is blocked by* `portsentry`*? Can the locked machine contact the one it blocks using any service or not?*	
	You can answer these questions: Do `portsentry` locks affect sc*anning speed? If the answer is yes, do you think this can also be used as a security measure?*	

	You can set up a connection with a traditional VPN service like *ProtonVPN* and use it to browse the internet more anonymously	
	You understand what a NIDS is for and the kinds of things it detects regarding the network traffic that passes through it	
	You can answer this question: *How many things do you think you can do to* secure `ssh` connections from a network standpoint? (think combining techniques we looked at here and previous labs)	
	You can protect yourself from `ssh` attack attempts (and similar attacks on other services that do) with `fail2ban` predefined jails.	
	You can enable a "port scan trap" on a machine without conflicting with normal *firewall* behavior. You can answer this question: *if you can collect blocked IPs, what would you create with them?*	
	You understand the basics of the ToR network, `.onion` links, and the pros and cons of using it, as well as the difference from a traditional VPN	
	You can build a network of your devices with a P2P VPN with *ZeroTier* and manage them securely, as well as distinguish these VPNs from traditional ones	
	**Darknet protection**: Your knowledge of network security allows you to deploy a fortified server capable of withstanding the most common network attack attempts against typical services and as an end user. In addition, it allows you to use the network anonymously, including using your services in a way that only you can see and access under normal circumstances	