



INFORMATION SYSTEMS SECURITY ENGINEERING (ESSI)  
COMPUTER NETWORKS SECURITY (SRC)  
NETWORK SECURITY (SR)  
HOMEWORK TP#5

---

# NETWORK SECURITY TOOLS HOMEWORK

---

February 28, 2022

Henrique Santos

Departamento de Sistemas de Informação  
Universidade do Minho ©

## Contents

<b>1 Fw – Introduction</b>	<b>3</b>
1.1 Summary of tasks . . . . .	3
<b>2 Tasks – fw (basic)</b>	<b>4</b>
2.1 Server initial setup . . . . .	4
2.2 Client initial setup . . . . .	6
2.3 Fine-tune the personal firewall . . . . .	7
2.4 Writing rules . . . . .	8
<b>3 Tasks – fw (advanced)</b>	<b>9</b>
3.1 Virtual Lab Architecture . . . . .	9
3.2 Impeding access to a website . . . . .	12
3.3 Add a schedule to rules . . . . .	13
3.4 Web server public . . . . .	14
3.5 Limit number of connections . . . . .	15
<b>4 NIDS - introduction</b>	<b>17</b>
4.1 Summary of tasks . . . . .	17
<b>5 Tasks – NIDS (basic)</b>	<b>17</b>
5.1 Setup Suricata . . . . .	17
5.2 Testing and fine tuning Suricata . . . . .	20
<b>6 Tasks – NIDS (advanced)</b>	<b>24</b>
6.1 Using scanning tools to test Suricata . . . . .	24
6.2 Using Pytbull . . . . .	26
<b>7 Tasks – NIDS (complementary, even more advanced)</b>	<b>29</b>
7.1 Preparing for ELK . . . . .	30
7.2 Installing ELK . . . . .	30
7.3 Preparing pfSense to send Suricata alerts to Logstash . . . . .	33
7.4 Adjusting Logstash . . . . .	37
7.5 Shaping Kibana . . . . .	41
7.6 Final remarks . . . . .	45

---

# 1 Fw – Introduction

This work aims to:

- improve knowledge about network security functions, using appropriate tools;
- improve knowledge about technologies and protocols involved with the network perimeter security;
- familiarise students with iptables and its operation in a personal firewalls;
- familiarise students with a tool (**pfSense**) to deploy several network security functions, like firewall and Intrusion Detection;
- develop competencies on the use of the **pfSense** platform, to filter network traffic and detect intrusions; and
- familiarise students with a virtual environment created for cybersecurity laboratory experiments.

To achieve the above objectives, this exercise is divided in two parts:

- In the first part, we propose a simple exercise involving two VMs linked by a private network. A VM implements a typical server, and that is where the personal firewall will be settled through iptables. The other VM is used as a client for testing purposes only.
- In the second part, the virtualization architecture is expanded to accommodate a network firewall (pfSense), forcing some adaptations. Some scenarios are provided, serving as a source of requirements to define traffic filtering rules. Besides, pfSense's modularity is also explored to expand its function into Intrusion Detection functionality.

## 1.1 Summary of tasks

1. Preparation of the virtualization environment (server, client and firewall), giving particular attention to network modes. This process is considered trivial, but some more details will be provided in the advanced tasks. Anyway, the time devoted to setup the experience is relevant to develop system administration skills.
2. Installation (if necessary) and configuration of the required services in the server, besides the programs **iptables** and **system-config-firewall**, to implement a personal firewall (on the recommended CentOS the effort will be minimal, but if you choose another OS you will have to identify equivalent programs).

NOTE: If you choose CentOS 7 it is possible that it comes configured to use the **firewallD** service by default – in production, this service is much more efficient for firewall

management and iptables use, with the added cost of making much more obscure the use of iptables, which is not what we want in this exercise; therefore, you may have to disable firewallD service to have the system-config-firewall available<sup>1</sup>.

3. Test the security of the server, provided by the personal firewall (version 1).
4. Tune the personal firewall configuration (version 2), and test again.
5. Adjust the virtual environment to accommodate a network firewall (pfSense), including its installation and configuration.
6. Develop rules for some network contexts, and test them (including tools for network traffic generation – **nping**, in particular).

Preliminary note: for safety and performance reasons, you should always keep the software updated. This observation also applies to VMs, unless there is some specific indication for not doing so.

## 2 Tasks – fw (basic)

### 2.1 Server initial setup

At the server and from a console:

1. Assuming you are using CentOS 6.x, you need to install the FTP service – all other necessary services should already be installed. To install the FTP service you can use **System** → **Administration** → **Add / Remove Software** and choose, from the servers category, "secure FTP – Very Secure FTP Daemon"); then you need to activate and initiate the desired services, which are "httpd", "sshd" and "vsftpd" (**System** → **Administration** → **Services**, by selecting each of the desired services and first enabling it – **enable** – and then starting it – **start**); you may also want to configure properly the keyboard and other aspects of the interface.

After finishing run the command `netstat -l | grep "tcp"`, which will allow you to check whether the desired services are available (LISTEN) and the respective TCP ports; you can also try to open the homepage and access FTP and SSH services, everything in the local host; register in your logbook the obtained results and discuss any eventual discrepancies and the workarounds.

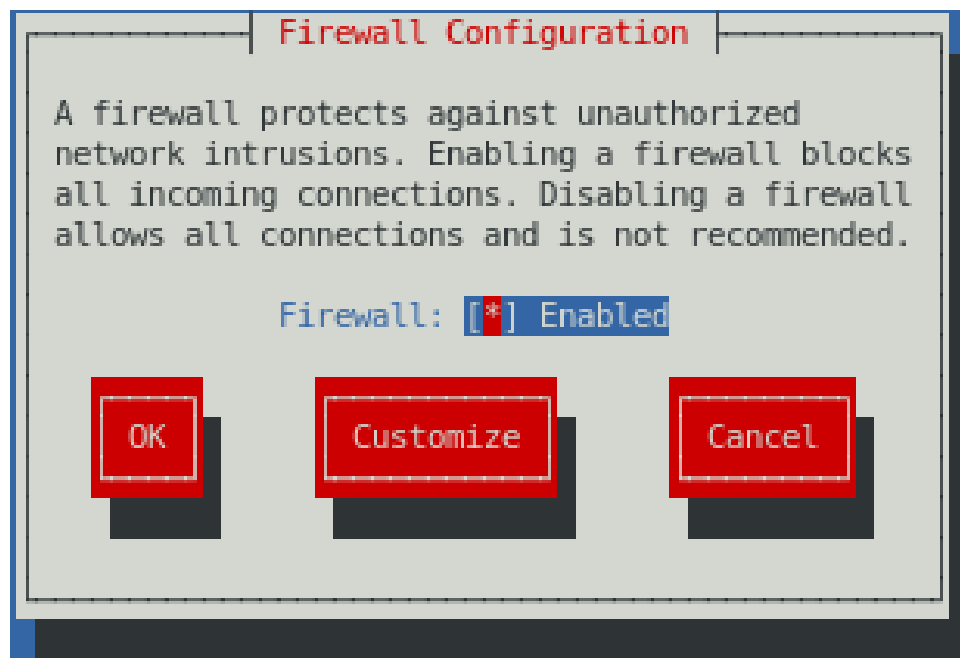
---

<sup>1</sup>A summary of instructions to carry out this change of configuration is available at <https://www.digitalocean.com/community/tutorials/how-to-migrate-from-firewalld-to-iptables-on-centos-7>

2. Execute the command `system-config-firewall-tui` (as mentioned before, it should be available in any Linux implementation based on the Fedora project, which is the case of CentoOS). This command allows setting the firewall in very simple way via `iptables`.

You should get a screen like the one shown in Figure 1, which shows the firewall status. In this window (with the <Tab> key, or the cursor movement keys, and the <space> key) select the **Enable** option, and then select the **Ok** button, which will end the program execution – depending on the implementation you are using the firewall may initially be enabled already, not requiring any change in this case.

3. Execute `iptables -L -v`, register its output, and comment:
  - (a) the default policies for each of the chains INPUT, FORWARD and OUTPUT;
  - (b) the general security level (network security policy) and other information that you can extract from the output obtained.
4. For security reasons, execute the command `iptables-save > iptables.dump` and save the file `iptables.dump` (which you should attach to your logbook). If at any stage of the exercise you feel lost, you can always come back to this point, restoring the tables to the actual state, with the command `iptables-restore < iptables.dump` (but before doing so it is prudent to clean all tables using `iptables -F` command).
5. Using again the `system-config-firewall-tui` command, disable the firewall and rerun the command `iptables -L -v`
  - (a) Register and review the new ruleset, discussing if it is safer or not, and why.
6. Enable the firewall again, reversing the operation performed in the previous step.



**Figure 1:** Main interface window of system-config-firewall-tui

## 2.2 Client initial setup

At the Client and from a console:

1. Start by checking the connectivity by running the command `ping <srv-ip-add>` where `<srv-ip-add>` is, of course, the IP address of the server.
2. Execute the command `nmap -sS <srv-ip-add>`
  - (a) What information did you get? Make time to explore some of the additional features of nmap (see, for example <https://nmap.org/book/port-scanning-tutorial.html>).
3. Execute the command `w3m http://<srv-ip-add>`

Note: This command executes a browser that works in text mode, similar to the popular lynx (available in other Linux implementations, such as Ubuntu), and it is used the same way; depending on the client you are using, you may have to install one of those programs.

  - (a) Are you able to view a page? Record the received output.
4. Now execute the command `ftp <srv-ip-add>` (you may need to install it too).
  - (a) Are you able to establish a connection? Record the received output.
5. Finally, execute the command `ssh <srv-ip-add>`

Note: if this is the first time you connect to the server using SSH it is natural that you receive a request to accept (or deny) the public certificate from the server. If this happens you must accept the certificate.

- (a) Are you able to establish a connection? Record the received output.

In principle, you have protected well enough the server... To the point of not allowing it to answer (almost) any service request. Compare the observed behaviour with the expected one from the network security policy you got in the previous task – comment any divergences.

## 2.3 Fine-tune the personal firewall

Going back to the server:

1. Run the program `system-config-firewall-tui` again. This time select the **Customize** option. With the arrows and space keys select **FTP**, **SSH**, and **WWW** services, which we want to be accessible (trusted). If you want to open a few more services (or ports), you can do it. When finished choose the **Forward** option, which takes you to a window where you will be able to authorize ports that are not directly identified by service names, like in the previous window – no need to change anything at this stage. Continue choosing the **Forward** option, which will take you to the following phases of the configuration:

- (1) the selection of network interfaces you want to have full access (nothing to change);
- (2) network interfaces you want to be masked (nothing to change);
- (3) activation of the port forwarding function (nothing to change);
- (4) the ICMP filter, which lets you select the ICMP commands that the firewall will filter – select all but the **Echo Request (ping)** option;
- (5) finally, you reach the custom rules editor, which let you build specific rules (no need to make changes, also).

After finishing the configuration cycle, you will return to the home window where you will select the **Ok** button and accept the changes, ending the program execution.

2. Execute the command `iptables -L -v` again.

- (a) Record the observed changes and interpret the various rules that have been changed, in the light of options chosen in the previous operation.

Now, at the client and using the console:

3. Run the command `ping <srv-ip-add>`

- (a) Do you see any changes from previous running? Is this what you would expect?

4. Execute the command `w3m http://<srv-ip-add>`

- (a) Are you able to view a page this time? Record the received output. To quit the program you must press the "q" key.

5. Execute the command `ftp <srv-ip-add>`

- (a) Did you manage to get a connection this time? Register the obtained output.

If you got a connection but cannot log in, that is not strange, since you have not configured vsFTPD server (which you should be using). But perhaps you succeed if you try the user "anonymous", without a password...

6. Finally, execute the command `nmap -sS <srv-ip-add>` again.

- (a) Record the information you get as output. Compare it with the previously obtained and reflect on the current security level, namely the implications on the network security policy.

Going back to the server:

7. Execute the command `ping <cl-ip-add>`, where `<cl-ip-add>` denotes the client IP address.

- (a) Register the obtained result? Is this what you would expect?

8. Execute the command `iptables -L -v` again.

- (a) Record the changes you observe and try to interpret them in light of the activity developed during this task.

## 2.4 Writing rules

Under normal circumstances, the rule set is designed and verified offline, and not in an interactive mode, as before. A good start point is taking the file obtained by the command `iptables-save` (as previously described) and editing it, carefully, adding all the new rules, or deleting the unwanted ones. It is highly recommended to keep track of all changes, with the respective justifications and additional details, since this is a critical operation, and the documentation is essential for maintenance.

With that in mind, this last task aims to prepare an iptables file with a ruleset that we typically would like to have in a workstation's firewall, as a baseline protection level.

1. To get an initial template, clear all tables and save the resulting ruleset in a file (we already used the necessary commands – but it should not be difficult to find one in the Internet if you have any difficulty).
2. Edit the obtained file, fulfilling the following requirements:



- (a) Default policies for all tables should be DROP.
- (b) All established or related TCP connections should be accepted, both in INPUT and OUTPUT tables – i.e., after the TCP connection request was accepted.
- (c) All traffic on the loopback interface should be accepted.
- (d) Since we are using a default DROP policy, it is necessary to accept outbound DHCP requests, so the workstation can get the IP address, the netmask, and other important information – required switches: `-p udp --dport 67:68 --sport 67:68`)
- (e) Likewise, DNS lookups should also be accepted in the OUTPUT table. DNS can run over TCP or UDP, in both cases targeting port 53. When considering UDP, to match the port number (switch `--dport 53`) it is necessary to precede it with the `-m udp`.
- (f) Other fundamental services, or ports, that should be accepted:
  - Remote management (allow inbound SSH, port 22)
  - Access to the web (allow outbound HTTP (port 80) and HTTPS (port 443))
  - Access to mail relay (allow outbound SMTP, port 25)
  - Keeping internal clock synchronized (allow NTP requests over UDP, using port 123 both as source and destination)
  - Check external hosts (allow outbound ICMP)
- (g) Whenever possible and to make each rule more specific – less chances to get stuck – include the identification of the network interface involved (`-i <iface>` or `-o <iface>` switches).

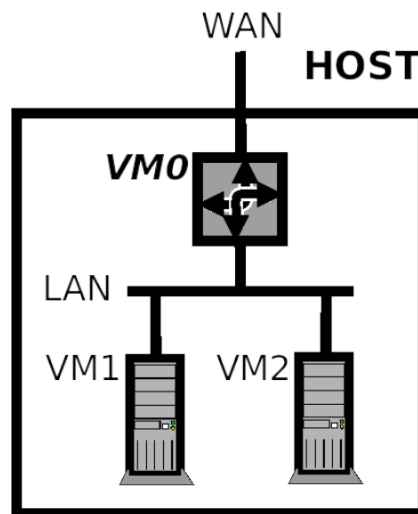
After finish editing you can test the file loading it with the command `iptables-restore`, debugging eventual errors detected by iptables, and trying to operate the workstation, utilizing all involved protocols, until it runs smoothly. You should register all experiences in your logbook.

### 3 Tasks – fw (advanced)

#### 3.1 Virtual Lab Architecture

Following the target architecture presented in Figure 2, we need first to configure the virtualization environment, in a way to allow the virtual machines VM1 and VM2 (which simulate internal computers, clients or servers) to connect to a private virtual network (LAN, in the figure), together with the firewall (pfSense, VM0 in the figure), which assumes also the routing function to the outside network (WAN, in the figure). For that purpose, the firewall needs a second network interface to connect to the external network, for which a bridge connection will be appropriate.

Concerning the internal virtual network, none of the usual virtual network modes (NAT and Bridge) allow the type of operation described above, since VMs are typically connected to the



**Figure 2:** General architecture to accommodate a firewall in a virtual environment

host and the hypervisor manages all those functions. Besides, if DHCP is used for LAN, it must be implemented by pfSense and not by the host since it will not be reachable. Virtual Box has a network mode named **Internal Network**, that fulfils those requirements<sup>2</sup>. But other virtualization platforms also include similar solutions.

This way, and with VirtualBox, the configuration is very straightforward, being only necessary to follow the next steps.

1. There are several documents in the web covering the pfSense installation (like [this one](#)). Anyway, it is important to highlight that since a firewall does not need any peripheral device, and to keep it with the minimal extras principle, when configuring the VM we can safely remove the USB, audio, serial port, and floppy support. Concerning memory and disk space it is better to use the recommendations for hardware requirements.
2. Concerning pfSense VM network configuration, activate Adapter 1 as Bridge (to the host physical network interface), and Adapter 2 as **Internal Network**. Give a name of your choice to the Internal Network (e.g., PrivLAN), and in the Advanced configuration section, chose the type **Paravirtualized Network (virtio-net)**<sup>3</sup>, and set the **Promiscuous Mode** to **Allow VMs**. This gives you more options if you need to analyse traffic, even if it is not expected to do.
3. All other VMs (VM1 and VM2 in Figure 2) should have **only one active adapter, configured precisely as Adapter 2 above**. Check those configuration details carefully since

<sup>2</sup>The VirtualBox documentation includes a very detailed description of this mode, at [https://www.virtualbox.org/manual/ch06.html#network\\_internal](https://www.virtualbox.org/manual/ch06.html#network_internal).

<sup>3</sup>With Linux machines, this type of interface has higher performance, since some operations are executed in hardware, in the network interface card.

an error will compromise the results and may be difficult to detect later.

4. During pfSense installation, the system will ask you to select an interface for the WAN function and another one for the LAN function. In the first case you will choose the one associated to Adapter 1, and in the second case, the one associated to Adapter 2 (you may need to check the MAC addresses in the VirtualBox settings windows, to make the correct choice).
5. After pfSense installation, you will see the IP addresses of each of the network interfaces. The one connected to the WAN will have an IP address provided by an external DHCP server (assuming it is configured as Bridge, as suggested). The one connected to the LAN, by default, will have the IP address 192.168.1.1/24. You may have to change this address, mainly if the WAN network uses the same network address (very common if your host machine is also in a private network) – but you may want to change it by any other reason. The console interface provides you a command to do that – `2) Set interface(s) IP address` –, also allowing to configure IPv6 and DHCP. For the exercise you need to configure DHCP, but not IPv6.
6. After setting up VM1 and VM2 and starting them (it is not relevant which VMs you choose, so let us keep CentOS as a server, and Kali as a workstation), we should test the connectivity between VM1 and VM2, between both and VM0, and also the VM1 and VM2 Internet connectivity.
7. It is possible to configure pfSense from its console (at least partially), but it provides a **remote web-based application** which is much more effective. From any of the other VMs, but preferably from Kali, open the browser, and using the URL `http://192.168.1.1` (assuming the default LAN address) we have access to a remote web application to fully configure and manage the firewall. In the first execution, it uses default credentials and will enter a setup wizard, to configure interfaces and the administration account (it is not relevant for this exercise). Anyway, it is a good idea to explore the interface using the official documentation, or any of the tutorials available on the web to install and setup pfSense. It is also a good opportunity to observe the initial rules pfSense includes, through the menu (**Interfaces** → **WAN**) for both LAN and WAN, and **make a prior assessment of the security level, concerning the default settings**.
8. After being familiar with the remote console, and before entering the core tasks, we need to make a small adjustment. Select **Interfaces** → **WAN**, and scroll down to the last section, **Reserved Networks**. There are two options, both selected by default, which implement two fundamental rules for ordinary network firewalls, on the WAN interface: **Block traffic that comes from private networks and loopback addresses**; and **Block bogon networks**, targeting traffic that uses reserved or not assigned IP addresses (the rules are

self-explanatory). However, the first one should be removed since we are running the lab in a virtualized environment, using private addresses. If we keep the rule, we will not be able to access VM1 or VM2 from the WAN, making it impossible to simulate and test external accesses. Unselect that option and press **Save**, following by **Apply Changes**, and selecting **Firewall** → **Rules** check that the respective rule was removed from the WAN section.

Now, we are ready to go.

### 3.2 Impeding access to a website

Context: after observing how collaborators are using the Internet during labor time, the CEO of your organization believes that employees are wasting too much time on Facebook. He discusses with you the possibility to modify the Internet Use Policy, including a clause to block access to Facebook. It is your job to enforce that policy. First, we need to determine Facebook's IP address, for which you are going to use **host** and **whois** commands (depending on your environment and location, parameters may need to be adjusted, and results may differ):

```
prompt:~$ host -t a www.facebook.com
www.facebook.com is an alias for star-mini.c10r.facebook.com.
star-mini.c10r.facebook.com has address 157.240.212.35
prompt:~$ whois 157.240.212.35 | grep NetRange
NetRange:          157.240.0.0 - 157.240.255.255
```

In a rule specification, we may need the IP address in the CIDR notation, which we can get using the inetnum interval obtained with command **whois**, eventually using a CIDR calculator (if it is not obvious), such as the one provided at <https://www.ipaddressguide.com/cidr> – that way, the obtained result is **31.13.83.0/24**.

1. Using VM1, or VM2, open the browser and try to access [www.facebook.com](http://www.facebook.com) (it should be working; if not, then you need to debug your firewall configuration).
2. Open the pfSense remote console, and select **Firewall** → **Rules**. Next, move to the LAN section, and choose the ↓ **Add** option (add a rule at the bottom) – this is a very specific rule and should be evaluated after the most generic ones, except those that eventually counteract what we are trying to filter (like default rules), in which case **we may have to move it up**. Configure a rule to **reject TCP** traffic in the **LAN interface**, coming from any machine in the **LAN net**, and going to the **network 31.13.83.0/24** (any machine in the Facebook network). Select also the Log option, and give the rule a **description** of your choice. Check if the rule is in the right position, and do not forget to **Save** and **Apply Changes**.
3. Go back to the browser tab where you initially opened the Facebook home page, and try to reopen it. You should be unable to connect this time – otherwise the rule is not being verified, you are accessing a page from the browser cache, or there is an alternative route

(you can check it with the `traceroute` command). Go back to the pfSense remote console, refresh the page, and check the **States** indicator of the rule you have created. Register the values indicated.

Edit the rule, changing the action from **Reject** to **Block**, and try again to access. Did you note any difference? Comment.

4. You can also modify the rule to target the server at 31.13.83.36 instead of all network, and specific ports (80 and 443) instead of any port. Comment on the advantages and limitations of doing so.
5. Select **Status** → **System Logs**, and choose the **Firewall** section. Register the evidence of the previous activity. In this window, we also have the possibility of checking the traffic that was blocked, and automatically generate a rule to accept it, or add the source IP address to a blacklist (a flexible mechanism, but we must use it carefully).

### 3.3 Add a schedule to rules

Context: Following the previous decision, you discussed with the CEO the results of the last measure, highlighting the number of complains you received from employees, and their reaction. They are still accessing Facebook from their smartphones and personal devices, getting even more distracted. So, you decided to restrict the application of the previous rule to a limited period, from 10 am to 4 pm, explaining to the collaborators why this is important for the organization. Now, it is easy to make this modification.

1. We need first to create a schedule, using **Firewall** → **Schedules**, and the **Add** option. It should have a name and **one or more time ranges**. Ranges are defined choosing in a calendar specific days, or weekdays (selecting them on the head of the calendar – in that case, the month has no meaning). Altogether, the schedule is a placeholder, including all time-ranges in which the rule will be applied. We can also add a description to the schedule and the individual ranges. Create a schedule following the specification above, and **Save** it.
2. Select **Firewall** → **Rules**, and edit the rule you create previously. In the **Extra Options** section, select **Display Advanced**. Scroll down until you see the **Schedule** field, and using the pick list choose the created schedule. Save the modifications, and back in the rules window, we will immediately see the modification in the schedule column, with the indication if the rule is being applied at the current time.

Adjust the time range (adding a new one and deleting the other) to a nearby period, so you can test the rule as before, using the browser. Do not forget to register the results, including information from logs.

**Note:** It is possible to implement this feature in iptables too, using the **module time**.

### 3.4 Web server public

Context: Most likely, we will need to make the internal HTTP server (VM2) accessible to the outside world, the web – after all, that server was deployed to serve clients on the internet! Since the server is installed in a private network, with a private IP address, we will need to implement a NAT rule to make it accessible. The firewall in the WAN side has the only public available network address. The rule will force to redirect all incoming packets (from the WAN) with TCP port 80 (HTTP), to VM2 (LAN), at the same port. The server will receive a connection request from the firewall (gateway port). The response will be sent to the firewall, which, in turn, will forward it to the client. That address translation taking place inside the firewall is what gives the process the name NAT. We will now create such a rule.

1. Select **Firewall** → **NAT**. We are interested in the **Port Forward** section – to explore all possibilities, click on the small button with a question mark, at the upper right side of the window. Now, select one of the Add buttons – assuming there are no rules previously created, it is not relevant which one we use.
2. The necessary fields to setup are:

**Interface** where the packets are entering the firewall, which is WAN.

**Protocol** we are interested on forwarding TCP packets, only.

**Destination** is the IP address the incoming packet should contain. In our case, and since we have only one public address, we can chose WAN address.

**Destination port range** we are only interested in port 80 (or HTTP) – from and to fields should have the same value in our case. If we choose **Other**, then we need to enter the numeric value in the **Custom** filed.

**Redirected target IP** here we must insert the IP address (private address on LAN) of the internal server.

**Redirect target port** in our case, the server operates on port 80 also (or HTTP)

**Filter rule association** the creation of a NAT rule requires the creation of one linked rule for WAN interface. If we select **Add associated filter rule**, that rule is created automatically, and that link will be signalled in the NAT rule itself (a cross-arrow line icon). Otherwise, you must create that rule by yourself, which can be a tricky task. Open a browser in the host machine, and enter the URL `http://<WAN-IP>` (in case you have doubts, the IP addressed is available in the pfSense text console – VM0 –, or in **Status** → **Dashboard**). You should get the home page of the internal server. You will get the same result if you try from any other computer in your host local network.

### 3.5 Limit number of connections

Context: Once we have a web server open to the Internet, we start being vulnerable to a denial-of-service (DoS or DDoS – see also ??) attack type. We are particularly aware of the possibility of an attacker to make more connection requests than our server is capable of handling. Furthermore, after analyzing the pattern of accesses to the server, we know that in regular operation it is not expected to have **more than twenty connection requests per second**. So, the obvious solution is to prepare a rule in the firewall to block any attempt to violate that value.

1. First, we need to explore a way of testing this type of rule. It is not practical to ask users to make requests at that rate, mainly because it is not possible to control that process. Besides, attackers use software tools to do that, and so the solution is to use also those tools. There are several tools, exploring a broad range of techniques to deploy DoS or DDoS attacks, but for the purpose described we are going to use **nping** (an open source tool, part of the Nmap project, for network traffic generating, and response analysis – more information at the home page <https://nmap.org/nping/>). If you have nmap installed you already have Nping too. If not, you can install it from the home page, or the package repository for your distribution. From a console at host execute

```
nping -tcp-connect -p 80 -rate=20 -c 40 <WAN-IP>
```

where WAN-IP is the firewall IP public address. The switches used are self-explanatory, but in short, we are generating 40 (-c) TCP connection requests (-tcp-connect) to the computer with <WAN-IP> address, at port 80 (-p), with a of 20 packets per second (-rate). The output will show each attempt result, and a final summary like:

```
Max rtt: 1.662ms | Min rtt: 0.451ms | Avg rtt: 1.370ms
TCP connection attempts: 40 | Successful connections: 40 | Failed: 0 (0.00%)
Nping done: 1 IP address pinged in 1.98 seconds
```

It shows us the minimum, maximum, and average server response time, the number of responses missed, and the total time (about two seconds, as expected). You can try to raise the number of connection requests per second until you reach a limit. You can also use Wireshark to inspect the traffic generated – these experiments should be reported and are very helpful to raise your skills with those tools and network traffic, in general.

2. Move now to the pfSense remote console, and Select **Status** → **Rules**. In the WAN section locate the NAT rule that forwards traffic to the internal server, and select the edit function. Scroll down until you reach **Extra Options** section, and select **Display Advanced**. There are two related fields that we need to setup: i) **Max. src. conn. Rate**, which specifies the maximum number of connection requests allowed, per host; and ii) **Max. src. conn. Rate**, which specifies the period that applies to the previous parameter. Together, those parameters

- define the rate limit expressed in connections per second – note that the condition can only be used with TCP packets. Fill in those fields according to our requirements (maximum of twenty connections per second). After saving and applying changes, you will get back to the rules screen, where we can note a new symbol (small sprocket) indicating the rule has extra options (if you pass the pointer over the symbol the system will show you the settled parameters).
3. From the host, rerun the previous **nping** command, but this time raising the rate and packets count (e.g., **-rate=30 -c 30**). The output now should evidence that some connections were not fulfilled, as expected. Now, selecting **Status** → **System Logs** and the **Firewall** section, there should be some entrances showing the refused connections, by a Rule named **virusprot overload table (1000000400)**. In practice, pfSense flagged the source IP address of the host and put it in an internal table used to store forbidden IP addresses that violated the connection count conditions (whatever they are). We can check it selecting **Diagnostics** → **Tables** and choose **virusprot** table from the drop-down list. There should be only one entry, which will remain there for one hour after the last connection attempt is registered (unless we delete it before).
  4. Next, run the same **nping** command from the other VM, first without exceeding the limit imposed, and after with overloading values (e.g., **-rate=40 -c 80**). Register the output and comment on the effect adding pieces of evidence from logs and tables, as before

The previous exercise did not cover all possible scenarios of interest, and, definitely, not all pfSense features. In particular, the monitoring operations allowed by **Status** and **Diagnostics** menus are critical for a regular surveillance operation, which is a primary job of a Cybersecurity Engineer. The **Dashboard**, as an entry point, can be configured to provide excellent first-view indicators of network problems. The small '+' button in the upper right corner allows choosing several **widgets**, like the **Traffic Graph**, or the **Firewall Logs**. Of course, a real environment has nothing to do with the lab one, and that is the reason no related task was proposed. Even so, you should be now capable of developing your master skills concerning firewalls (both personal and network levels).

A final remark concerning the rule verification process: along with the exercise, you may have noticed some error indications when you tried to set up some parameters that a specific rule does not accept. This is a feature most firewalls provide for what we can call a basic check. However, keeping a ruleset coherent is more laborious and requires a more robust understanding of how firewalls work and network traffic behaves. This is, and always will be, a demanding research area.



## 4 NIDS - introduction

### Network Intrusion Detection

For this exercise we will use Suricata, but with the firm conviction that the skills developed will be useful to work with other tools. Besides, Suricata is available in pfSense, as an external module, being very easy to put it to work, and at the border of the network side-by-side with the firewall, which is a recommended position to implement a NIDS. Nevertheless, using alternative implementations is very easy, and instructions are available both in the documentation area of each system, or in the web. So, we will start with the architecture previously used in the firewall exercise in the proposed virtual lab (see section 3.1).

#### 4.1 Summary of tasks

1. Set up and test Suricata within pfSense, with emphasis on the rule set selection and preparation.
2. Set up a MITM attack and test the detection capacity of Suricata. For this task, we will need all the virtual machines working and a tool named **Ettercap** available in the attacker machine (Kali already have it installed).
3. Create rules for a specific purpose, exploring the ambiguity that is typically included in such rules.
4. Review the alerts produced by Suricata and explore some of the tricks to minimize false positives.

## 5 Tasks – NIDS (basic)

### 5.1 Setup Suricata

Suricata can be downloaded in binary format for almost all OSs, in source code, integrated with network security platforms (such as Security Onion, or OSSIM), or as an external package along with pfSense. Since we already have an infrastructure with pfSense in our virtual lab, from the previous exercise, we will take the last option – follow the instructions in the first task of section 3.1, if it is necessary to install pfSense, or follow the official documentation, when deciding to implement a dedicated box. However, with the chosen strategy, we may need to adjust the resources allocated to pfSense, following Suricata requirements: a minimum of 4GBytes of memory, and two or more processors depending on the host available – to take advantage of the Suricata multi-threading capabilities.

1. Starting with the web interface, and selecting **System** → **Package Manager**, we get a page with the list of installed packages (none, by default). In the tab **Available Packages** we

have access to a long list of packages, and among them, Suricata. Search for it – taking the opportunity to look at the list of packages available this way – and press the + **Install** button. The process is fully automatic, installing Suricata and Barnyard.

Note: Barnyard is spooler-like utility used by Snort and Suricata to speed up the alert registering process, implementing the interface with the database.

2. The next steps consist on performing a basic configuration, setting up the essential options, but leaving many others with default values, which usually are adequate – the modification of some options requires a deep understanding of the Suricata architecture, requiring an additional effort. Under **Service** menu there now should be an entrance for **Suricata**. Selecting it will open the configuration page, starting with the **Interfaces** section (it should be empty, initially).

Moving now to the **Global Settings** tab we will be able to configure one of the most important components, **the rules to be used**, which is accomplished with the following steps:

- (a) There are several options, and some of them are related to paid services – the business case of IDSs is mostly supported on the rules providing process, behind which there is a lot of continuous research work. Fortunately, there are also some free versions, mainly supported for testing and to demonstrate how an IDS works. We will use the free versions (**ETOpen** and **Snort Community Ruleset**), of course, but it is useful to give a look at the web pages of the service providers and, eventually, to register with Snort VRT (no payment required) to have access to the Free Registered User rules.
- (b) It may be also important to configure the rules **Update Interval** – in a production system this can be critical, but in this exercise one week will be enough.
- (c) Selecting the **Live Rule Swap** capability allows to automatically restart Suricata after a rule update – we will leave it unselected for now.
- (d) The **GeoIP DB Update** feature (selected by default) is also useful.
- (e) Finally, we can configure the time interval to keep hosts blocked, if such an intrusion response is being used (fifteen minutes would be a good choice, with a prototype deployment).

Some of the features described may be defined by a Security Policy, and in a production system, it must be set according. After finishing this phase we must press the **Save** button.

3. After configuring the rules, we need to update the local stored rules, selecting the **Updates** tab. There is a list of installed rule sets, the date of the last update, and their signatures (it should not ready at this stage). Pressing the ✓**Update** button will do the required job, and after a short time, the list will show the final result. We can also check for any problems during the update process by viewing the management rule set log - by pressing the **View** button.


4. Next, we will jump into the **Interfaces** section, and proceed by adding an interface, using the + **Add** button. The first option is the interface name, which Suricata inherits from pfSense, and assuming the configuration from the previous exercise, we will have two possible interfaces: WAN, and LAN. This also means that we can monitor any or all the networks created with pfSense. We will start with WAN selected (however, the choice is not relevant, and it could be LAN, as well), and now the more complex job begins, which is the rules settings and fine-tuning.

📌 Note: we are leaving all other settings with default values, but there are three groups deserving particular attention: i) the **Logging Settings**, which allows defining what Suricata will be logging, in which formats, and where, under assumption that performance and storage space are being taken carefully (e.g., when we decide to send alerts to the system log, the system should be prepared to store much more logs and in a persistent way). Enabling JSON log may also be interesting, in particular when looking for integration with other tools, like the visualization ones; ii) the **Alert and Blocking Settings**, with the option **Block Offenders**, which, when checked, will force Suricata to block any IP address that generates an alert (the blocking time was previously configured in the Global Settings section); and iii) the **Performance and Detection Engine Settings**, which allows controlling important parameters, with implications on available resources – we may change the **Detect-Engine Profile** to high, when using a powerful host.

- (a) Selecting the **Categories** tab will take us to the section where rule sets can be chosen. Scrolling down the window will show the rule sets from Snort Community and ETOpen repositories, previously selected. At this point, we do not have any criteria to choose specific rule sets, so we are going to press **Select All**.

We will also check the **Resolve Flowbits** option. This is an interesting feature forcing linked rules (by *flowbits*) to be automatically selected. The *flowbits* mechanism implemented by Suricata is part of the rule construction language, allowing to set a named *flowbit* in a rule, which will be tested in another rule(s) – this is particularly useful with TCP or application protocols, where an alert must only be issued after some preconditions are met, imposed by previous packets, helping to reduce false positives or redundant alerts. After setting all the details, we need to **Save** the configuration.

- (b) Now we will go to the **Rules** section, where we can control the operating rules, from each rule set. The first rule set is already selected and scrolling down will show the respective list of rules, along with the information about the activation state (see the legend above the list), its action, the unique identification number (SID), the main components of a rule (IP addresses and ports), and the messages logged when a match occurs.

 from the Categories section, if we click on a rule set, we will jump directly to this page, with that rule set already selected.

We can change the state, enabling or disabling each rule, and we can see the rule itself clicking over the SID. Selecting **Active Rules** in the category box will show we that, in the present configuration, we have more than twenty thousand rules, showing the necessity of fine-tuning, otherwise we will get a large number of false positives. Again, we have no criteria, at this point, to unselect any of the rules, of any of the rule sets, so we will leave the list unchanged. After finishing the modifications, we need to click on the **Apply** button, to make them effective in Suricata.

- (c) We will conclude the interface configuration with the **IP Rep** section, concerning the IP Reputation feature, which we will disable. Suricata implements an open architecture strategy concerning IP reputation, where a central server interacts with sensors to manage the reputation lists. The server can use globally accessibly reputation lists or implement its own strategy. Without that infrastructure, Suricata will keep feeding the internal lists using the local alerts information, only.
  - (d) The other fine tuning options within the **Interfaces** section will not be addressed now, but it may be worth to look at the Variables section, where we can define variables with internal machine names that Suricata will use to help interpretation of alerts and logs.
5. After finishing the configuration, it is now necessary to **start the interface monitoring function**. This is accomplished through the **Services / Suricata / Interfaces** page, pressing the icon with a small arrow under the **Suricata Status** column. Another useful operation we can do now is duplicate the configuration just performed, for another interface. Pressing the middle button (two small rectangles overlapped) under the **Actions** column, will open the initial interface configuration page, already with the LAN interface selected (in case there are only two, and the WAN was the one previously used) and fully configured with the same rules enabled, only requiring to press the **Save** button. After that, we can start the monitoring operation on the second interface, too. Note that if there are not enough memory, Suricata will give an error – if that happens, it is required to shutdown pfSense and modify the VM memory size. After starting the interfaces, the icons under the **Suricata Status** change, showing a green circle with a checkmark indicating the running status, and two more buttons, one to **restart** the service and another to **stop** it.



## 5.2 Testing and fine tuning Suricata


After starting the monitoring service, Suricata will probably begin showing some alerts immediately, depending on the activity of your network, including the host and the external network in use. Thus, the next steps can produce slightly different results in each case.

1. **Managing alerts** is accomplished through the **Suricata / Service / Alerts** page, where it is possible to visualize alerts originated in each interface, selecting it in the **Instance to View** box – all the elements in the page are considered self-explanatory. In principle, the LAN traffic is residual and will not generate any alerts until we force some action. The WAN interface has a different behavior. VMs may access the Internet as part of updating processes, or general information access services, like the pfSense web application itself – internal modules of the web interface access external DNSs, which triggers a rule since internal machines are supposed to access the DNS only through the gateway. Anyway, we can force some activity, e.g., executing the update command in the CentOS machine (`sudo yum update`, or equivalent in one of the VMs), while watching the **Alerts** page (it is not necessary to perform the update, but just forcing the search for updates).
  - (a) Register in your logbook the different alerts you get and try to relate them with your system's activity (the above description may help you).
  - (b) Take the description of each alert type and perform a google search. Do those alerts correspond to real threats?
2. Even at a small scale, it is evident that the number of alerts can increase rapidly, in a noisy fashion, making it difficult to analyze them efficiently. Fortunately, there are some easy options to help fine tune Suricata, from the **Alerts** page itself:
  - (a) In each line describing an alert, there are three kinds of icons, namely, i) a **small magnifying glass**, allowing to perform a reverse name resolution operation, for both the source and destination IP addresses – naturally, it only gives useful information for public IP addresses, not private ones – (try it and register the result); ii) a **small red 'x'** in **GID:SID** column, allowing to disable the rule and remove it from the current rule set – we can always re-enable it later; and iii) a **small 田**, allowing to add a rule to a **suppress list**, with no conditions (when pressing it in the **GID:SID** column), or conditioned by an IP address (when pressing it in the **Src** or **Dst** columns) – Suppress list are used to register rules that we want to stop generating alerts, but that will not be disabled from the rule set (particularly useful when we want to stop alerts for a rule, linked to a particular IP address).

The suppress mechanism has some details that need to be highlighted. When suppressing a rule, Suricata creates automatically a suppress list, adds the rule to it, and links it to the interface. If more rules are suppressed the same way, they will be added to the same list. Suppressed lists can be managed through the page **Services / Suricata / Suppress**, where we can edit, delete, or create lists (hand made), and jump to the interface where a list is first instantiated (small ► boxed icon in the **Actions** column).

To gain some experience with this fine-tuning mechanism:

- i. from the alert page create a suppress list from the rule whose description includes "ET POLICY GNU/Linux YUM...", **tracked by the source IP address** – using the  button;
- ii. next, do the same with the rule whose description includes "ET TOR Known Tor...", but this time **tracked by the destination address** – if this alert is not present in your case, you may safely choose another one;
- iii. Go to the **Suppress** page and press the edit option (small pencil icon) in the Actions column, for the list just created (there should be only one); dedicate some time interpreting the way Suricata defines the content of a list, and see the examples at the bottom of the **Suppression List Content** block, which present some of the usual constructs used (very important when creating entries manually, for particular purposes not related to the automatic actions from alerts page);
- iv. back to the **Suppress** page, click on the small arrow icon in the most right position, in the Action column, which will take us to the edit interface setting page of the WAN interface; scroll down until the **Alert Suppressing and Filtering** block, where we can see the suppress list just created attached to the interface; the drop-down button allows us to attach another list or the default list – **if a list is attached to an interface it can not be removed, in the Suppress page**;
- v. edit the list removing the second entry (keeping the one related with the package management activity) and save it; go to the **Alerts** page and check that there is only one rule in the suppress list (the one with a **circled 'i'** icon replacing the ); and
- vi. in the CentOS VM execute the update command again and verify the generated alerts, to confirm the excluded one (if there were no errors).

 The excluded alert was marked as a "Potential Corporate Privacy Violation", assuming there was such a policy stating, for instances, that regular desktop computers cannot perform updates. If the target machine was a server, for which it is expected to have daily updates, the suppression was adequate to remove that false positive.

- (b) **Filtering alerts display** can be accomplished from the Alerts page, pressing the '+' icon in the **Alert Log View Filter** bar. Not surprisingly, it is possible to filter alerts by identification, IP addresses, protocols, data, description, and classification (all relevant piece of information belonging to alerts). That allows viewing just the alerts satisfying the conjunction of parameters indicated, being useful to see, for instances, the number of alerts in a specific time interval, the alerts associated with a particular host, or the number of alerts with a given identification. Notwithstanding the usefulness of this feature, the analysis capacity it allows is limited – **external tools are necessary for improved visualization and analysis**. The ultimate goal of previous operations is to identify rules, or rule sets that **it is safe to disable or suppress**. That is far from

being an easy or quick task, requiring a lot of effort, Google searches, and sharing experiences through some interesting forums, like the one used by pfSense users, at Netgate (<https://forum.netgate.com/category/53/ids-ips>). As a starting point, but always recognizing that there is no single solution, as there are no two identical environments, and similar risk aware perceptions, the following guides can be useful:

- i. **Informative alerts** (non suspicious traffic) are raised by traffic variants that can be linked to network components particularities and after evaluated, can be safely disabled;
- ii. **Classification** and **priority** are two important details, allowing to capture the nature of the alert and the severity level. They are both defined in a file named `classification.config` (within the pfSense implementation it is located at `/usr/local/etc/suricata` directory, and we can access it through a shell in the pfSense VM, or remotely through `ssh`, after enabling it). Priority can be any number between 1 and 255, but in the default classification file only values from 1 to 4 are used – higher values mean higher priority.

As an example, in our case, there is probably a large number of alerts classified as "Potentially Bad Traffic", priority 2, resulting from traffic from the pfSense WAN interface, directed to an external name server (port 53) – we can check it using the reverse name function –, with SID 2013743, and a description indicating a query to a suspicious no-IP domain. This traffic is generated by the pfSense web application, and it can be safely ignored. Instead of disabling the rule, which we want active for other occurrences, we can suppress it when involving the WAN interface. After doing that, we can close and start the web application again and check if we are still receiving those alerts. After suppressing this rule and the one related to updates (previously described), we should have now a much more quiet IDS – it may also be useful to clean the alerts with the **Clear** button, eventually saving them first with the **Download** button.

- iii. Alerts related to internal IP addresses and associated to services we trust, suggests rules not adjusted. Depending on the situation we can again suppress the rule, or even disable it, or edit the rule to adjust it.
- iv. Alerts related to **services (TCP or UDP ports) we are not implementing are irrelevant** and the associated rules can be removed. After all, even if this traffic is malicious, it will not cause any harm since no machine will receive it. We may be missing the opportunity to identify an attacker running a scanning, but the cost of false positives is higher.
- v. Not all rule sets previously selected are relevant, and some of them will never be used. If possible, a very effective strategy is disabling those unnecessary (or even undesired) rules. Furthermore, we selected both ET Open Rules and Snort Community Rules, which have a lot of common or very similar rules (despite, by default, several rules are

already disabled). ET Open Rules are frequently considered enough, and from it we may find essential the following rule sets (to which it is necessary to add the rule sets associated with specific services):

- emerging-attack\_response.rules
- emerging-bootcc\_portgrouped.rules
- emerging-bootcc.rules
- emerging-ciarmy.rules
- emerging-compromised.rules
- emerging-current\_events.rules
- emerging-dos.rules
- emerging-dshield.rules
- emerging-exploit.rules
- emerging-malware.rules
- emerging-scan.rules
- emerging-shellcode.rules
- emerging-trojan.rules
- emerging-worm.rules

Overall, fine-tuning an IDS is a continuous and challenging task, and any organization immensely appreciates those mastering the skills necessary to engineering an efficient rule set.

## 6 Tasks – NIDS (advanced)

### 6.1 Using scanning tools to test Suricata

In this phase, we will start forcing some traffic that, despite not being offensive, it is not benign, either. In large, that traffic is linked to network scanning operations, through which it is possible to detect active hosts, active services within hosts, and even to identify versions of applications and OSs. Nmap is one of the most popular tools available for the purpose and the one we will use. But before, it is better to configure the pfSense dashboard (**Status** menu) to include the usual **System Information** and **Interfaces** boards, and, at the right side, the **Traffic Graphs** and the **Security Alerts** boards (this last one should be configured to show at least ten alerts). Optionally, we can hide the WAN information since we are going to work mainly over the LAN, generating the activity with the Kali VM, and targeting the local virtual network or the second VM.

1. From a console in Kali, and while visualizing the traffic activity and the Suricata alerts in the pfSense dashboard, in the background, execute



```
nmap -PS -v <LAN-add>
```

where <LAN-add> specifies the LAN address in CIDR notation, and the -PS switch indicates the type of scan. With that command, Nmap will "ping" all hosts, using TCP SYN packets using the most usual ports. Observe the output, the generated traffic volume pattern and, in particular, if Suricata produced any alert. If not (the most probable result), **should it have?** To look for an answer, we need carefully investigate the `emerging-scan.rules` file already referred and prepared to detect that type of activity. It can be accessed with any text editor, or through pfSense's Suricata service menu, as explained previously. If you find a rule that you think should generate an alert with the above command, **indicate which modifications you must perform for that purpose.**

Try with other options of the -P switch (see the help output) and **compare the results.**

2. Next, following the same steps, execute the command

```
nmap -sS -v <srv-ip-add>
```

where <srv-ip-add> specifies the IP address of the target server, and -sS selects a port scan also based on TCP SYN packets and using a standard range of TCP ports. Proceed the analysis of the result as above, including the investigation of the rules, and the use of alternative -s switches (in particular the -sX, generally referred as more intrusive). To go a deep further with the analysis, complement it observing the real traffic, with Wireshark.

3. To complete this phase, execute the command

```
nmap -A -v <srv-ip-add>
```

where the -A switch enables Nmap to perform a full scan on the target, including services and versions. This operation is more intrusive and performed at the services level and not at the network level. This time we should receive some alerts. **Take note of those alerts and try to justify them with the type of action performed with Nmap.**

Another useful tool to test an IDS is **Hping** which is similar to the well-known ping, but allowing to operate with several other protocols, and many other features that can be used to simulate some attacks. Kali includes the last version, `hping3`, with a useful description available [here](#).

4. Keeping the same screen organization (i.e., with the pfSense dashboard in the background, showing the traffic graphics and Suricata alerts) and from a Kali terminal, execute the command

```
hping3 -S -flood <srv-ip-add>
```

which will generate SYN packets (-S), but without complete the 3-way handshake, and as fast as possible (-flood). This is a simulation of a DoS attack, known as **TCP SYN flood**,

targeted to the server. While executing the command observe in the pfSense dashboard, the traffic volume, the CPU usage, and if the activity generated any alert. As before, look to the `emerging-dos.rules` file content to justify the eventual alerts generated, since it includes the rules (supposedly) to detect that type of attack.

5. Next we will execute the command

```
hping3 -S -flood -rand-source <srv-ip-add>
```

which is the same but with an additional option to force Hping to use random values for source IP address. This time we should get some new alerts, but not associated with a DoS attack. *It is an excellent opportunity to investigate the reason of those alerts, and if the alerts correspond to the true nature of the activity.*

## 6.2 Using Pytbull

For the next exercise, we will use **Pytbull**, a powerful public domain modular framework to test IDS/IPS, developed in python. It uses some other tools, such as **Nmap**, **Tcpplay**, **Nikto**, **Ncrack**, and **Hping**, to simulate network-based attacks. We need to install it in the Kali VM, but most of the required tools are already there, in particular **Ncrack** – which, otherwise and following Pytbull’s documentation, would need to be compiled, what may be a difficult task in a strict Linux environment like Kali.

Pytbull’s basic operation consists of i) generate the traffic mimicking selected attacks, targeting a machine (supposedly a server) specified in the command line, eventually forcing the target to download files from external machines (depending on the type of test); ii) download the alert file generated by the IDS, from the target machine, through FTTP; and iii) check if each test was, or was not detected, and build a report. After running the tests, Pytbull initiates an HTTP service at port 8080 in the local host, through which we can access the report and observe the IDS effectiveness. This feature assumes the IDS is running in the target machine, or otherwise it will not be possible to access the alert file. Our lab configuration (see Figure 2) does not match that and so, to run Pytbull, we **need to create a fake empty alert file in the server** (`/var/log/suricata/fast.log`), which makes the Pytbull result analysis useless. **This is not an issue since we are not interested in the Pytbull report, but only in the traffic it generates** – however, Pytbull is one of the most efficient tools to test an IDS and mastering it may be an important skill, in particular, because we can create or modify the tests crafted to our own purposes.

According to Pytbull’s documentation, in its last version (2016), it runs more than 300 tests divided into 11 classes, in several possible configurations. For our purpose the **stand-alone mode** is enough, dispensing the Pytbull server, essential only to simulate client-side attacks, which we will not deploy – see the documentation for additional information.

1. In our environment, installing **Pytbull** consists only on decompressing and placing it in the recommended location (`/opt/pytbull`). Anyway, it is advisable to follow the documentation instructions, but not trying to compile **Ncrack**, as referred above. Next, it is necessary to configure Pytbull, which is accomplished through the `config.cfg` file located in the `conf` directory, and using any text editor. The file has eight sections, and some of them require special attention:

**CLIENT** where it is enough to enter the Kali VM's IP address and network interface linked to the internal network.

**PATH** where we only need to uncomment the correct `alertsfile` field; take note of the path and file name since it is necessary to create it in the target machine (the server VM in our lab installation), as an empty file – you can use the `touch` command. Pytbull will try to get this file using FTP, and so the service must also be running – in the previous exercise we configured the server with `vsftpd`, which is adequate for the job.

**ENV** where the paths for all used tools must be correctly set. That can be hard since the paths may vary from one implementation to another. To find programs' path in Linux we can use the `which`, the `locate -b`, or even the `type -p` commands; to locate any other file we can use the `find / -name` command. As a reference, Listing 1 presents a possible configuration for this section.

**Listing 1:** Pytbull configuration file

```
[ENV]
sudo          = /usr/bin/sudo
nmap          = /usr/bin/nmap
nikto         = /usr/share/nikto/nikto.pl
niktoconf     = /etc/nikto.conf
hping3        = /usr/sbin/hping3
tcpreplay     = /usr/bin/tcpreplay
ab            = /usr/bin/ab
ping          = /bin/ping
ncrack        = /usr/bin/ncrack
ncrackusers   = data/ncrack-users.txt
ncrackpasswords = data/ncrack-passwords.txt
localhost     = 127.0.0.1
```

Even if we are running a small set of tests that do not require all tools, Pytbull checks it when starting and will stop in case of an error.

**FTP** where we need to enter the credentials to access the FTP server, as root (for our reference configuration, with the CentOS and `vsftpd`, the username is root and the password is `osboxes.org`)

**TESTS** where we select any or all of the 11 test classes, setting a '1' for the ones to run, and a '0' for the others. There is no way of selecting individual tests in a class.

2. For the first attempt select only **ShellCodes**<sup>4</sup> tests, and from the directory where pytbull was installed (/opt/pytbull) run the command

```
./pytbull -t <srv-ip-addr>
```

while observing in background the pfSense dashboard, as before. When prompted, choose the first option to run a new campaign, and accept the aware notice. In case an error comes up, it is necessary to correct some configuration parameter in the configuration file – the Pytbull's documentation includes some comments on possible errors, too.

Pytbull will identify the individual tests as they are running. **Register the obtained alerts, as well as the system performance indicators (indicative values during the process)**. Jump to the **Services / Suricata / Alerts** page to get more information about the alerts (priority, explored service, and alert class). Search on the web for more details pertaining to the ShellCode itself. **Can we consider that as a true positive? Should we had received some more alerts? If yes, what is missing?**

Finally, stop the Pytbull's web server execution pressing **Ctrl+C**.

3. Next, we will repeat the previous experience, but selecting **evasionTechniques** tests class. Pytbull will lunch several tests, using Nmap, Nikto, Icmp, and even Javascript, over TCP and UDP. This is a very reach group of tests that will produce a large number of alerts. Execute Pytbull using the same procedure as before, but it may be a good idea to clean alerts first. After finishing and stopping Pytbull, move to the Suricata's alerts page and **try to relate the alerts with the activity generated, always with a focus on the efficiency concerning false positives and true positives**.

After completing the previous exercises, most probably you end up with thousands of alerts of different types and embedding an enormous amount of information, making almost impossible to extract useful knowledge to support effective security response decisions – unless you know precisely what you are looking.

To address that difficulty, we need more than just pure Security Engineering skills. Given the number of variables and data available, we need to resort to some **data analysis skills**. However, the technological development in that area produced very complex frameworks and tools, supported on also sophisticated analysis methods, **forcing a long learning curve**, in particular when the data gets immensely big – mainly when dealing with large organisations and multiple log sources, well above the single NIDS we used in our simple lab. So, in real scenarios, it is probably more efficient to segregate the Security Engineering functions from

---

<sup>4</sup>A ShellCode attack consists of embedding a piece of malware in a packet payload, aiming to have it executed in the target machine, and most likely, opening a shell that will let the attacker to gain access.

the Data Analyst functions, despite being true, they need both to work together for a better outcome. For small organisations, the amount of data is more modest and, almost certainly, it is impossible to allocate enough human resources to follow the above strategy. In such cases, the person in charge needs to accomplish the job by him/her self.

## 7 Tasks – NIDS (complementary, even more advanced)

To help with the above issues, the next exercise aims to **develop basic data analysis skills** with common tools usually available in security operation rooms, namely **ELK** (**Elasticsearch**, **Logstash**, and **Kibana**) stack, **Sguil**, and **Squert**. **Sguil** is a first level application over the NIDS, allowing to handle alerts in real-time, organizing the information by simple dimensions, like categories or severity. It also allows low-level access to the alerts and the packets. But even for this low-level access and by performance reasons, it is evident the necessity to separate the capture function from the analysis function, which is accomplished, in the Sguil case, by a **MySQL** database and a spooler module named **Barnyard**. Besides freeing up the NIDS from the time-consuming storage function, this spooler interprets the raw alerts and rearrange the information in a tabular format ready to submit to the database. Barnyard can be set up within pfSense also, but it requires a database implementation, which should be remote to not overdue the firewall and NIDS primary functions.

**Squert**<sup>5</sup> implements another level over the Seguil database structure. Squert is a web application that provides context information and several model techniques, like time series, which allows visualizing NIDS alerts exposing information that is not obvious or easy to get from simple tabular representations.

The ELK stack, with its main components, follows a similar approach but it is more powerful, flexible, and embracing. Logstash plays a similar role to the Barnyard but allows the interface with a multitude of log sources, including Suricata and Snort. Its generic three-stage pipeline architecture (input, filter, and output) facilitates the customisation of virtually any data source. Information is structured in a JSON format and sent to Elasticsearch, a distributed, multi-tenant capable search engine, based on the Apache Lucena project, implementing a similar function to the MySQL database, but in a much more versatile way. Kibana is a web-based visualisation engine designed to explore the Elasticsearch content through a reach set of visual primitives and search queries. That is the same role played by Squert but, again, at a more complex level. Of course, all that power comes with a cost, concerning the computational resources required. Besides, by its very nature, ELK is not suitable for real-time monitoring, being more appropriate for aggregate analysis in extended time windows.

**Security Onion** is an open source Linux based distribution specially crafted for Intrusion Detection Systems, and it includes all the previously referred tools, in addition to some others

---

<sup>5</sup>The project seems to be inactive since 2016, but several security solutions still use it.

used for network monitoring in general. A minimal non-production implementation requires, at least, eight gigabytes of memory and four CPUs – the required disk space depends, obviously, on the amount of logs to store. It can execute in a virtual environment, like our lab, but its performance is limited, unless we use a powerful workstation. So, in the next exercise, we will propose a limited ELK implementation, extending Kali and pfSense systems. This option aims to promote some practical work in a simple and accessible learning environment, but keep in mind that in a real scenario the Security Onion or similar platforms are preferable for the job.

## 7.1 Preparing for ELK

After working with Suricata and pfSense, it should be evident that looking at alerts on an individual basis is almost useless. At least, the time and counting dimensions (absolute and relative), as well as the localisation of interlocutors, are obviously important. Looking at Figure 3 and comparing with the previous output provided by pfSense, we can easily see it. The two bar graphics at the top show the number of alerts, in intervals of 30 seconds and in a time window of 1 hour, for the WAN (left graphic) and the LAN (right graphic) networks. The bottom left bar graph shows the rank of the top five alert type, while the bottom right map shows the location of external IP addresses, through a color gradient scheme coding the number of accesses. In the next steps, we will see how to set up our lab to have Kibana exhibiting that type of dashboard.

## 7.2 Installing ELK

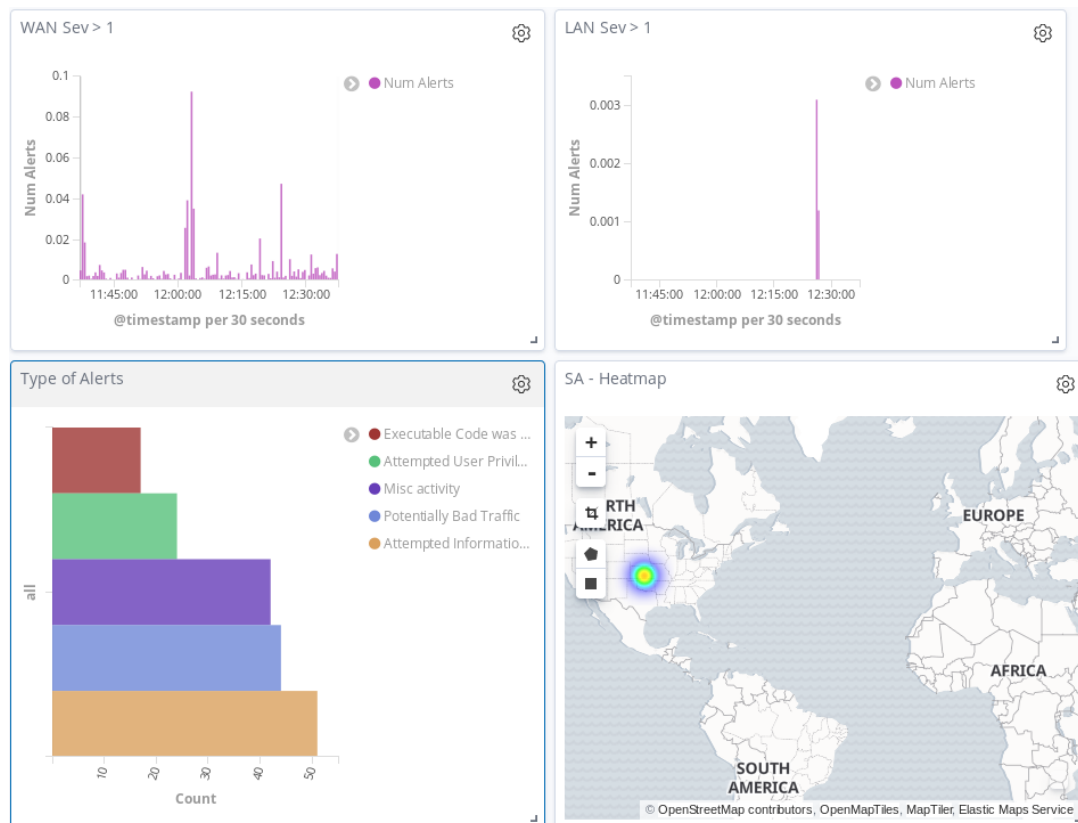
We start by installing ELK on the Kali VM – in the following setup steps, we assume execution at the root level; if not, you will need to use `sudo` for most of the commands, as usual. Installing ELK is a straightforward process since the three modules (Elasticsearch, Logstash, and Kibana) are all available from the same repository, frequently referred by Elasticstack. As previous requirements, it is necessary to have Java (JVM) and `apt-transport-https`, which are probably already installed – if not, concerning Java look for instructions at the official site; concerning `apt-transport-https`, it is available in the standard repository. It is also recommended, as a minimum, to rise Kali's memory to **5GBytes**, and give it **2 CPU cores**, at least.

Concerning the Elasticstack, first we need to download and install the GPG key through the command:

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | apt-key add -  
and then add the Elasticsearch repository, through the command:  
echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | \  
tee -a /etc/apt/sources.list.d/elastic-6.x.list  
note that you may need to adjust the version number.
```

Next, and as usual, we need to update the package list executing `apt update`. After that, it is only necessary to install each package through the following sequence of commands:

```
apt install elasticsearch
```



**Figure 3:** Example a dashboard prepared with Kibana

```
apt install logstash
```

```
apt install kibana.
```

Now, there are some details to configure in each module.

1. **Kibana:** the configuration file it is located at `/etc/kibana/kibana.yml`. The relevant settings for now are `server.port: 5601` and `server.host: "0.0.0.0"`, meaning the server will respond to TCP port 5601 (the default) and bind to all interfaces (you can bind it to a specific local interface indicating its IP address, but that is not necessary for our architecture).
2. **Logstash:** it is configured creating one or more files located in the directory `/etc/logstash/conf.d`. By ease of handling reasons, it is better to create one file for each pipeline stage (input, filter, and output); but keep in mind that Logstash will process and concatenate all files inside the directory, and so eventual configuration errors will be reported as if there was only one file. Concerning Logstash input and output, the suggested content for the configuration files is (filenames are arbitrary):

- (a) `01-inputs.conf` – to allow Logstash to accept Syslog inputs via port 5140 using TCP or UDP, and (essential) **Elastic Beats** via TCP port 5044. Beats is a generic designation

for a full range of log shippers developed by Elastic, as lightweight agents running on source devices, to deliver log data to Elasticsearch. One of those agents is the **Filebeat**, available also for pfSense and capable of forward Suricata alerts formatted as JSON files – we are going to configure it in the next task. The configuration file content for the purpose described can be found in Listing 2.

**Listing 2:** Input Logstash configuration (01-inputs.conf)

```
#syslogs via TCP/5140
input {
  tcp { type => "syslog"
    port => 5140}
}

#syslogs via UDP/5140
input {
  udp { type => "syslog"
    port => 5140}
}

# Elastic Beats input
input {
  beats {
    port => 5044
  }
}
```

- (b) **30-outputs.conf** – to forward incoming data to Elasticsearch module, running in the local host, and to generate an index for each day; indexes are a key mechanism within Elasticsearch and choosing the right schema is very important; daily indexes seems a good choice, but for a system with low activity it may be enough weekly indexes. For the above purpose, the configuration file content can be found in Listing 3.

**Listing 3:** Output Logstash configuration (30-outputs.conf)

```
output {
  elasticsearch {
    hosts => localhost
    index => "logstash-%{+YYYY.MM.dd}"
    # for weekly indexes (xxxx is the year the week starts)
    # index => "logstash-%{+xxxx.wk}"
  }
  # stdout { codec => rubydebug } #useful for debugging
}
```

The filter stage also needs to be configured, but we will do that after describing the **File-**



**beat** configuration for clarity reasons. Logstash is a powerful mechanism to aggregate and consolidate logs, and the above configuration is a basic one. In particular, concerning Syslog messages, we will get a single text string, and filters will be essential to define proper data fields. The JSON format provided by Suricata includes already data fields making it simple to process – the Elasticsearch documentation provides more information about Logstash and filters (either available and to customize).

3. **Configuring services:** after installation we will end up with three new services, namely **elasticsearch**, **kibana**, and **logstash**. We have now the option of initiate the services manually, using the **service** command, repeating the process each time we boot Kali, or configure them to start automatically at boot-time, using the command **systemctl** – the next table summarizes both options.

Manualy	Start automatically
	<code>systemctl daemon-reload</code>
<code>service elasticsearch start</code>	<code>systemctl enable elasticsearch.service</code>
<code>service kibana start</code>	<code>systemctl enable kibana.service</code>
<code>service logstash start</code>	<code>systemctl enable logstash.service</code>

The Elasticsearch should now be ready to use, even if it has no data since we did not prepare the data feeder (Suricata, in pfSense). Anyway, we can check the services status, using, for example, the **service <service-name> status** command (replacing <service-name> by each service name), and, more interesting, we can access Kibana through a browser with the URL **http://localhost:5601**. In the first execution, Kibana lets the user to experiment with internal sample data – an effective way to get familiar with the interface – or just start exploring it, which makes sense only after having some data. We can also check the Elasticsearch engine pointing the browser (or using **curl**) to **http://localhost:9200**.

Logstash is a critical (and tricky) module, and even if it is up and running, it still can produce errors while processing the pipeline configuration files. However, **it will not display any error**, and debugging needs to be done through the logs, which, in our case, are stored at **/var/log/logstash/logstash-plain.log**. Consult the file (e.g., using the **tail** command) and see if there are any line indicating an error and if that is the case, in which pipeline stage.

Finally, using the system resource monitor and with all services running (even without data) check the memory and CPU utilisation, which will give a clue about the necessity to allocate more resources to Kali.

### 7.3 Preparing pfSense to send Suricata alerts to Logstash

PfSense has its own package set, and it does not include Filebeat. However, pfSense is based on FreeBSD, and it is possible to add packages from the standard repository, where Filebeat can

be found – but keep in mind that pfSense performs automatic updates only over its repository, meaning we have to update Filebeat manually, whenever it is necessary.

The first step consists of locating the correct Filebeat package. The best suggestion is to start at the official [FreeBsd web site](#) and search for instructions on using packages. Alternatively, searching the web will allow you to find it.

1. From the pfSense console open a shell (option 8) and execute the command

```
pkg add <URL of package>
```

copy the URL found above – you should not have a Copy/Paste facility between the host (where you performed the search) and pfSense, where you are running the command; so, you need to type it. The package is installed under `/usr/local`, the executable is located at `/usr/local/sbin/filebeat`, and the configuration file at `/usr/local/etc/filebeat.yml`.

2. During its development process, Filebeats changed the way it uses some modules. Unfortunately, the beats package on the FreeBSD did not incorporate all modifications, and it will be necessary to add some modules manually (namely, logstash). That can be accomplished by:

- (a) Download the Linux (64-bit) Filebeat package from [Elastic web site](#) and decompress it (running `tar -xvf <packagefile>`), at the host. Entering into the extracted directory, we are interested in the `module` and `modules.d` directories content.

- (b) Transfer those directories to the server is not immediate. As a suggestion, you can try temporarily install an FTP server in Kali (vsftp is perfect, but the installation may require some effort). The above indicated directories need to be copied to `/var/db/beats/filebeat/`, with the same name – that is assumed later, in the configuration file. We should now be able to enable the necessary modules for your architecture, namely **logstash**.

3. The next step is to configure Suricata to generate alerts to a JSON file. From the pfSense web interface, selecting **Services** → **Suricata**, and then the **Interfaces** tab, we will be able to edit any of the interfaces (we can start with the LAN), clicking over the respective small pencil icon. Once at the **Edit Interface Settings - LAN** page, you must scroll down until the **EVE Output Settings** section, where you will first click on the **EVE JSON Log** check-box. The section will expand, showing a lot of log options. There are some mandatory settings: i) **EVE Output Type** should be **FILE**; ii) **EVE Log Alerts** check-box should be selected; and iii) **EVE Log Alert details** check-box should be selected for **Log a packet dump with alerts**, too. All other details are optional and not necessarily related to alerts, but we can select those associated with the most common protocols (HTTP, SMB, TFTP, TLS, and SSH), if enough resources are available. When finished, we need to press the **Save**

button and repeat the process for the other interface (WAN).

📌 Note1: selecting DNS Traffic, Suricata Stats and Traffic Flows, will generate several logs continuously which can become a disk space issue.

Note2: the logs (one `eve.json` file for each interface) will be stored in a subdirectory whose name is related to the interface, under `/var/logs/suricata`. After configuring the interface, it is possible to check the file with any editor or text reading command.

4. Keep in mind that Filebeat is an agent responsible for collecting the local information (`eve.json` files) and preparing a stream to submit to Logstash, located at the Kali machine (in our case). So, it needs to know the type of logs, the location in the remote computer, the location of the logs in the local computer, and the module to use. This configuration is a multistage task.

- (a) The configuration information is stored in the file

`/usr/local/etc/filebeat.yml`, that we need to create – YAML format uses indentation to define the scope, so we should use spaces instead of TAB chars.

Note: pfSense includes only simple text editors (vi, and edit), without copy/paste facility. So, it will be easier to create the configuration file at the host and send it to pfSense using FTP, as before.

Listing 4 shows the proposed content for the configuration file (obtained from [here](#)).

**Listing 4:** Proposed Filebeat configuration file

```
##### Filebeat global options #####
filebeat.config:
  modules:
    enabled: false
    path: /var/db/beats/filebeat/modules.d/*.yml
#----- File prospectors -----
filebeat.prospectors:
- input_type: log
  paths:
    - /var/log/suricata/*/eve.json*
  fields_under_root: true
  fields:
    type: "suricataIDPS"
    tags: ["SuricataIDPS", "JSON"]
#----- Logstash output -----
output.logstash:
  hosts: ["10.10.100.50:5044"]
#----- filebeat logging -----
```

```
logging.to_files: true
logging.files:
  path: /var/log/filebeat
  name: filebeat.log
  keepfiles: 7
```

Detailed information about this configuration file can be found in the documentation at the Elastic web site, but even so it is important to highlight some aspects:

- The prospectors section defines the place where local logs are (all `eve.json*` files), and also some tags ("SuricataIDPS" and "JSON") to add to a field "type" in each record, to make it easier to locate them when searching with Kibana, later.
- Output to Logstash will be via the IP address indicated, using TCP port 5044. **If the Kali's LAN interface is using DHCP**, it may get a different address at boot time, and we may loose connection. So, it is pertinent to go back to Kali now and change its network configuration to use static IP address, for the LAN interface.
- Filebeat will log operations in a local file, named `filebeat.log`, located under `/var/log/-filebeat` – **it may be necessary to create this directory**. A new log file is created when system initiates, or when the log file reaches the default limit of 10MBytes (this can be configured, also). In our configuration, Filebeat will keep 7 files of past logs, naturally the most recent ones.

(b) Now we need to enable the **logstash** module, using the command

```
filebeat -c /usr/local/etc/filebeat.yml modules enable logstash
```

and check if it is enable, with the command

```
filebeat -c /usr/local/etc/filebeat.yml modules list
```

which will show both the enabled and disabled modules (logstash should be the only one enabled).

(c) Finally, we can test the configuration using the command

```
filebeat -c /usr/local/etc/filebeat.yml test config
```

which hopefully will not report any error in the configuration file (otherwise you are required to correct it).

5. As we are using Filebeat with pfSense, and despite being a FreeBSD based implementation, to make it start at boot time require some special operations. When installing the package a startup script (named `filebeat`) is created in `/usr/local/etc/rc.d`. However, it must have the `.sh` extension, which can be done easally with the `cp` or `mv` commands.

The script assumes some settings to be configured in `/etc/rc.conf`. Again, by a pfSense implementation detail, that file is overwritten at boot time and any modification we could have made will be ignored. To overcome this limitation we can create a `rc.conf.local` file to allow Filebeat to start on boot. This can be done with the following command sequence:

```
echo "filebeat_enable=yes" » /etc/rc.conf.local
```

```
echo "filebeat_conf=/usr/local/etc/filebeat.yml" » /etc/rc.conf.local
```

We can now reboot pfSense and verify if Filebeat is running using the `ps` command. We can also verify if it is producing logs using `tail -f` to see the most recent entries of the `/var/log/filebeat/filebeat.log` file.

6. Filebeat keeps track of Suricata events that it already has sent, and takes care also of eventual communication errors. In all, it ensures that any log is actually sent once. This is a great feature, but sometimes we wish to resend logs, specially when performing tests. To do that it is necessary to delete the Filebeat registry, executing the command

```
rm /var/db/beats/filebeat/data/registry
```

and restarting it, using the `filebeat.sh` script, executing

```
/usr/local/etc/rc.d/filebeat.sh stop
```

```
/usr/local/etc/rc.d/filebeat.sh start
```

## 7.4 Adjusting Logstash

This task is focused on the filtering capacity of Logstash and how to use it to empower the information associated with Suricata alerts. It is a continuation of the previous task aimed at ELK preparation. But now, we have a clear idea of where the data is coming from and how it is delivered, which is fundamental to understand Logstash's pipeline role.

At this point we have two Logstash configuration files, one for input (`01-inputs.conf`) and the other for output (`30-outputs.conf`) – see Section 7.2, if necessary. We will start adding a simple filter file (`10-pfsense-filter.conf`, with the content shown in Listing 5, for which it is important to highlight:

- the **input** and **output** sections are included, but commented out, being there for debugging. When developing a filter, it is helpful to submit it to Logstash and check if the output is what we expect. So, if we uncomment the input and output functions, and remove the `if [type] == "suricataIDPS" {` statement (and the correspondent `}` character), we are able to execute:

```
/usr/share/logstash/bin/logstash -f ./10-pfsense-filter.conf
```

Now, providing some input through the keyboard, or even better, **copying and pasting real alerts from the `even.json` file**, it is possible to observe the filter output;

- concerning the filter operation, the `if` statement restricts its application to logs for which the **type** field equals **"suricataIDPS"** – remember we configured Filebeat to insert that field, precisely to distinguish the alerts generated by "our" Suricata (see also Listing 4), since Elasticsearch may be processing logs from several sources.

- The **json filter** processes the parameter passed as 'source', and it creates a field in the output record for each JSON field it parses.
- The **date filter**, as the name suggests, extracts dates and times from fields (timestamp field, in this case) and uses that information as the Logstash time stamp for the event – otherwise, it will use the current machine time.

**Listing 5:** Basic Logstash filter file

```
#input { stdin { } }
filter {
  if [type] == "suricataIDPS" {
    json {
      source => "message"
    }
    date {
      match => [ "timestamp", "ISO8601" ]
    }
  }
}
#output { stdout { codec => rubydebug } }
```

We will now add three very informative elements to the alerts using Logstash filtering resources. One is the geo-location of public IP addresses, the other is the domain name (FQDN – Fully Qualified Domain Names) and TCP service names, and the last is the alert's rule origin.

1. Logstash includes a filter, named **geoip**, that takes an IP address and a proper database, and inserts the respective geo-location (if available) in a field also passed as parameter. The database is provided by MaxMind (there is a free version and a paid one, with more information), and we need to install a specific module, available as a package. First, we need to add the package location and then install it, using the following commands:

```
add-apt-repository ppa:maxmind/ppa
apt-get update
apt install geoipupdate
```

Next, it is necessary to edit the file `/etc/GeoIP.conf`, to use the free databases versions, making sure the edition IDs line is: `EditionIDs GeoLite2-Country GeoLite2-City`

After saving the file we need to update the databases executing **geoipupdate** – you can also program automatic updates, e.g., weekly, using the **crontab** facility.

Finally, we need to append the code in Listing 6 to the filter block in the configuration file `10-pfsense-filter.conf`.

**Listing 6:** Filter's code to insert geo-location

```
# Suricata Alerts: set the geoip data based on src_ip
if [event_type] == "alert" {
  if [src_ip] {
    geoip {
      source => "src_ip"
      target => "geoip"
      database => "/usr/share/GeoIP/GeoLite2-City.mmdb" }
    mutate {
      convert => [ "[geoip][coordinates]", "float" ] }
  }
  else if ![geoip.ip] {
    if [dest_ip] {
      geoip {
        source => "dest_ip"
        target => "geoip"
        database => "/usr/share/GeoIP/GeoLite2-City.mmdb" }
      mutate {
        convert => [ "[geoip][coordinates]", "float" ] }
    }
  }
}
```


Notes about the code:

- The first 'if' statement limits the filter application to alert type events, which is our main goal.
  - The second 'if' statement checks the `ip_src` field, and only execute `geoip` if it exists. If it does not exist and if there is no `geoip.ip` field previously created, then `geoip` is executed with `dest_ip`.
  - The `mutate` filter is a generic field manipulation, in this case, used for a conversion operation required to allow the posterior processing of the coordinates' values by Kibana (consult Logstash documentation for additional information).
  - The filter will add to the record a "**target**" structure (named `geoip`) with several fields pertaining to the geographical location – you can check it using the debugging procedure describes above to test the filter file.
2. To generate the FQDN associated to an IP address we will use a filter primitive named `dns` with the capacity to perform a reverse IP lookup operation. But since there are no fields to receive the names, we need to create them first, using the `mutate` primitive.
- To replace the TCP port number by a user-friendly name we will use the `translate` filter, which takes a simple dictionary (CVS, JSON or YAML format) and a number, and returns

the corresponding name (adding the field, if necessary). Translate is a plugin module we need to install, executing

```
/usr/share/logstash/bin/logstash-plugin install \ logstash-filter-translate.
```

The dictionary must exist in `/usr/share/logstash/dictionary/` – we can download one from web, e.g., from [here](#).

 The CSV file can be in a DOS format and the translate filter does not recognize it. We can overcome the issue executing the `dos2unix` on the file. **Remember that Logstash does not report any errors and to detect this issue the only way is to consult the Logstash's logs.**

The code to perform these operations is in Listing 7, which we need to append to the filter file `10-pfsense-filter.conf`, as before.

**Listing 7:** Filter's code to insert FQDN and service names

```
# Add FQDN via reverse DNS lookup
mutate {
  add_field => { "src_FQDN" => "%{src_ip}" }
  add_field => { "dest_FQDN" => "%{dest_ip}" } }
# DNS reverse lookup
dns {
  action => "replace"
  reverse => [ "src_FQDN" ] }
dns {
  action => "replace"
  reverse => [ "dest_FQDN" ] }
# Add TCP/UDP Service names
translate {
  dictionary_path =>
    '/usr/share/logstash/dictionary/PortN2ServN.csv'
  field => "dest_port"
  destination => "dest_port_serviceName" }
```

3. To make alerts a little more user-friendly, we can also add information about the source of the rule that fired the alert. The alerts include a field indicating if the rule is from Emerging Threats (signature is ET), or from Suricata (signature is SURICATA). Based on that information, the code in Listing 8 creates some fields with complementary information, using the `mutate` primitive. As before, this code must be appended to the filter block of the configuration file `10-pfsense-filter.conf`, which is now finished – in case of having some difficulties editing the file and putting it all together, a complete version is available [here](#).

**Listing 8:** Filter's code to add complementary alert information



```
# Add additional fields related to the signature
if [alert][signature] =~ /^ET/ {
  mutate {
    add_tag => [ "ET-Sig" ]
    add_field => [ "ids_rule_type", "Emerging Threats" ]
    add_field => [ "Signature_Info",
      "http://doc.emergingthreats.net/bin/view/Main/%"
      "[alert][signature_id]}" ] }
}
if [alert][signature] =~ /^SURICATA/ {
  mutate {
    add_tag => [ "SURICATA-Sig" ]
    add_field => [ "ids_rule_type", "Suricata" ] }
}
```

## 7.5 Shaping Kibana

Kibana has several features to explore, but concerning its fundamental operation and the goals of this exercise, it is essential to work with **Searches**, **Visualisations**, and **Dashboards**.

1. We access Kibana via browser, pointing it to `http://localhost:5601`, as explained in Section 7.2. Kibana uses **index patterns** to access information stored in Elasticsearch, which uses indices – see Listing 3, where we configured Logstash to generate daily indices with the prefix "logstash-".

Selecting the menu **Management** and then clicking on **Index patterns** under Kibana section, a form will come up to define an index pattern, in a two-step process. Start typing `logstash*` and the list of indices matching the string will appear. We only need to be sure that all the indices we are interested in appear in the list (in our case, and particularly in the first run, if all activity occurred in the same day and without rebooting there will be just one index). If there are no indices the most probable reason is Logstash not being feeding Elasticsearch because of an error in the pipeline – as referred, to debug it you must i) check if `even.json` and `logstash-*.log` files are being generated; and ii) check the `logstash-plain.log` file for errors reported by any of the pipeline modules.

Moving forward to the **Next step**, it is convenient to define which field to use as a time reference. In Section 7.4, when adjusting Logstash filter, we set "timestamp" as our time reference (see Listing 5) and Logstash derived the `@timestamp` field, which we should now select by the drop-down list in the **Time Filter field name** box.

📌 Logs and alerts, in general, are an aggregation of time-series. So, the time field is always the primary dimension of analysis, which justifies its use as an essential filter. Furthermore, Kibana's first filter is time ranges, which will not work if Time Filter was not set.

The process finishes pressing the **Create Index Pattern** button, which brings up all the data fields available (it may be necessary to press the refresh button, the circular double arrow icon at the top right area). **Take some time inspecting the data fields and types.** Having defined the index pattern, we can now explore the three main operations.

2. The search feature is mainly explored through the **Discover** menu option. By default Kibana exhibits the **Last 15 minutes** events – see the top right bar – which may show nothing. However, clicking on the actual selection and defining a larger value, should show some events. If there are no events, it is only necessary to force some activity, following the same procedures of the previous exercises.

When there are events available a counting graphic will come up, along with the list of events, ordered by the timestamp, and showing also the **\_source** field, which contains the complete event description (has no type, naturally). In the left column we have the **Selected fields** area, and the **Available fields** area. Scrolling down this (long) list, and selecting the field **source**, will expand it to show the details – top values. Passing the cursor over the field name will show an **add** button, which **allows to add it to the Selected fields group**, and also **appearing as a header in the list of events**, in the right side. In front of each source value, there are **two small lenses**. The one with a **'+' sign creates a filter to show all events with that value**, the one with a **'-' sign will create a filter to not show the events with that value**. The filters appear in the top area of the window, in the **Add filter +** bar, which we can use to create filters manually, too.

**Some suggestions to experiment with this feature:**

- (a) Explore the **source field**, or in **\_iface field**, to show events only from the LAN or from the WAN.
- (b) Explore the **event.type field** (particularly important when we have alerts and logs mixed).
- (c) Explore the **alert.severity field**.
- (d) Explore the **alert.signature field**, and the **alert.category field**.
- (e) Dedicate some time exploring other features of the UI, like the possibility of defining a time-window click-dragging on the graph, and using filters. It is also possible to save any view for later use, through the **Save** menu.

Finally, and as new events arrive, there can be some **Available fields** tagged with a **'?'**, meaning no defined type, because of the lack of values, and making it impossible to use them in filters. We can overcome it going to the **Management → Index Patterns** menu and refresh the field list (circular double arrow button at the top right).

3. There are many different ways to view alerts, depending on the purposes of the analysis and, in some way, on the security team's expertise. Suricata (as well as other network security tools)

generates a large number of data items and to aggregate (or co-relate) them in a meaningful and efficient way require a long learning curve. In this process plays an important role the capacity to visualise specific parts of the information, some of them in real-time, others in off-line mode, using different representation forms. Mastering this analysis tool is an essential goal of a Security Engineer working in a SOC (Security Operations Center), and Kibana offers an excellent framework to do that.

In this exercise it is proposed to explore three different strategies to visualise alerts, as an initial path to that long learning journey – there is no assumption about the relevance of those three ways; however, their implementation covers a significant number of techniques in this matter.

- (a) The first one (probably one of the most striking) is a map, identifying the geographical location of IP addresses communicating with our network system – that is why we added the GeoIP interface to Logstash. Selecting **Visualize** from the main Kibana menu will take us to a page where we can create a visualisation, or find anyone previously created (e.g., created when trying Kibana with internal data). Going on with the process to create a new one, we will select **Region Map** from the template list.

Next, we reach the first phase (common to all visualisation creation operations), consisting of choosing the data source. We can select a saved search, create a new search (more specific), or select an index (from the previous exercise we should have only one, **logstash\***), which will be our choice.


- Now, under the **Data** tab, we will keep for **Metrics** the count value, and for **Buckets**, clicking on the **shape field**, we select **Terms** for aggregation (which is the only option, for the object we are creating). That will expand the shape field form, and for **Field** we must select **geoip.country\_code2.keyword**. By default, values are ordered in a ranking and the **Size** allows to control how many different items appear. So, to see more than the top 5 IP addresses, we should rise the value (e.g., 20). As an option we can add a label yet.
- In the **Options** tab, make sure that vector map's option is World Countries, and join field's option is ISO 3166-1 alpha-2 code. All the rest, we can modify according to personal preferences.
- Finally, we can check the result pressing the **Apply chances** button, the triangle-like control at the top right side of the form. In case there are no results, we may need to enlarge the time-window (or even generate some alerts, as described in previous exercise).

To finalise the process, we must save the object, through the **Save** menu.

- (b) While maps are great to give an idea of the interlocutors geographical dispersion, counting critical alerts, and segregate them by internal and external networks, is an essential first

view about the state of security. To do that we are going to create a new visualisation, this time of the vertical bar type. But first we are going to create and save a search, through the **Discover** menu, using as filters:

- "type is suricataIDPS", restricting to events coming from Suricata – relevant in case we are receiving from other sources; and
- "event\_type is alert", restricting to alerts – in case Suricata's configuration includes logs from other sources (e.g., DNS, DHCP, flows, etc.)

Save the search with a suggestive name (e.g., "SuricataAlerts"), and then select **Visualize** →  → **Vertical Bar**. As source data we select the created search, and after it will be displayed the form to configure the object, along with a graph showing the result of the search in one column (and assuming the default configuration, which we will modify).

- In the **Data** tab, we will keep the **Y-Axes** as suggested (**count**) in the **Metrics** section. In the **Buckets** section, and selecting **X-Axis**, we will choose for **Aggregation** the **Date Histogram**, and for the respective **Field** the **@timestamp**. Concerning the Interval to consider when counting events, keeping the **Auto** option will allow Kibana to adjust the axis resolution to space and time-window. However, we can change according to our needs.
- The other two tabs, **Metrics & Axes**, and **Panel Settings** allow us to configure visual settings, but the default values are adequate – as usual, take some time exploring the different possibilities.

Now, since we want to segregate the data by the two networks (LAN and WAN), we will add a filter using the **Add a filter** + control, above the form. In the fields section, we should enter **in\_iface**, with the **"is"** operator, and the value **"vtnet1"** (assuming this is the value passed by Suricata). We can optionally give the filter a name, and customise a label for the graph (e.g., LAN Network). Finally, we must save the object.

To create a similar visualisation but for the WAN network, all we need to do is (starting from the previous one):

- modify the filter value to **"vtnet0"**; and
- modify the label according (if it was set previously), and save it, but assuring to select the option **Save as a new visualisation**, and changing the name, obviously.

Moreover, we can fine-tune these objects adding to the search or the filters another condition to assure **we will not see alerts for which the severity level is 1**. This can be a controversial point since despite being considered not critical, those alerts can be informative. However, they are not being eliminated but only hidden to give more visibility to severity level 2 and 3 alerts.

- (c) In a more detailed analysis the **'top N'** alert categories may also be very informative. For this rank-like visualisation we will use a horizontal bar graph, selecting **Visualize**

→ ☐ → **Horizontal Bar**. As source data we select the defined search for Suricata alerts. In the Buckets section we need to select the **Split Series** type and, for aggregation, select **Significant Terms**, associated to the field `alert.category.keyword`. We need also to indicate the size, which refers to the top list number of elements (5 will be adequate) – remember the possibility to define a label. To see the result we need to press the button **Apply changes**, and then save the object.

Selecting the **Visualise** menu should now show the 4 visualisation objects created. we are now ready to create a dashboard.

4. Selecting **Dashboard** from the main Kibana's menu will take us to the dashboards management window. If there are no previously created dashboard, the only option is to create a new one. Otherwise, there will be a list of available dashboards we can choose from.

The creation process is straightforward. A new blank window comes up, where an **Add** button brings a list of previously created visualisations. In our case, we will select the four we have just created, and then we can move them around and resize, according to our preferences. It is crucial to adjust carefully the time window, which naturally applies to all graphs. Finally, we must save the dashboard (**Save** menu), optionally saving the time frame within the dashboard – without that option, Kibana will use its default time-window, which may not be what we want, especially if we are creating dashboards to analyse past events, with old Elasticsearch indexes.

Along the previous exercises with Kibana it should have become clear its power and complexity concerning the different possible ways to manage the security alerts. We approach only the fundamental aspects, but there are a lot more to explore. **Dedicate some time trying other visualisation forms, as well as functionalities we did not refer, like the machine learning capabilities.** However, you need to be careful with the limitations of running it in a virtualisation environment.

## 7.6 Final remarks

The ELK stack can work with many more monitoring tools, including Snort (similar to Suricata), Bro (a Behavioral-based IDS), OSSEC (a Host-based IDS), and a plethora of other systems. As previously referred, Security Onion is a Linux box including several of those tools. The effort to put to work any of those tools in an efficient way is enormous, and the supporting community is a perfect forum to expand our knowledge and expertise in using those tools. Nevertheless, there is a long and hard way to master the intrusion detection capacity at a professional level. However, as they say, *"the way is made by walking"*.

An excellent complementary and consolidation exercise consists of repeating this same activity but using Security Onion instead of the Kali, where we installed ELK. There are other required modifications, starting with pfSense elimination, since Suricata is already available in

Security Onion, and reserving at least 8Gbytes of RAM and 4 CPUs cores, which are the minimum requirements. But assuming a similar operation, we will need at least two more VMs, one as the target (can be the CentOS again), and one as the attacker, for which Kali is one of the best options, clearly without ELK. Getting computational resources for this configuration is a challenge, and a possible solution is to use a free account in a Cloud Computing provider – e.g., Google Cloud Platform gives you a reasonable credit to use such resources for one month, which is enough to complete the training activity.