Laboratory 10. Defending Web Applications

DEPARTMENT OF COMPUTER SCIENCE. UNIVERSITY OF OVIEDO

Computer Security | 2022 – 2023 (V2.7 *"S-81 Isaac Peral"*)

CONTENT

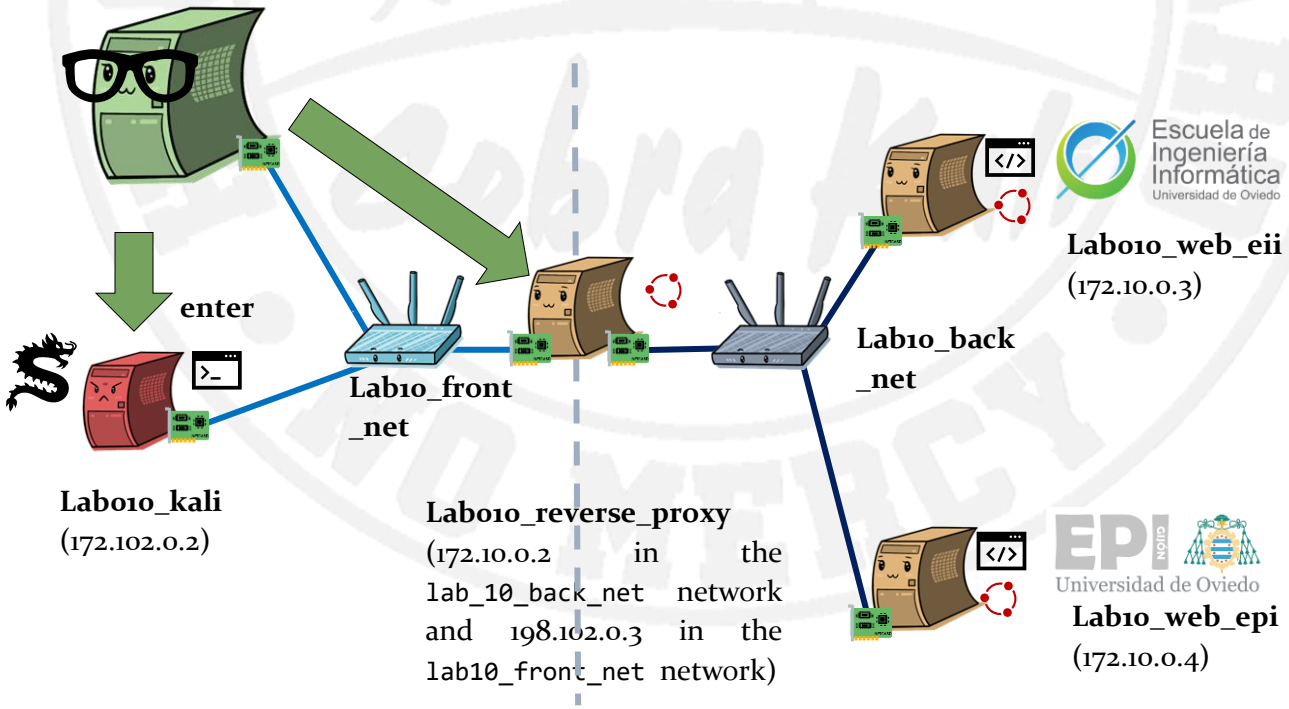
THE INFRASTRUCTURE OF THIS LABORATORY

The provided **Lab 10 infrastructure** follows the typical conventions of previous labs and has the following containers:

- We have a **Kali** "attack container" with tools to "attack" the proxy. It only communicates with the proxy. Its IP is `198.102.0.2`.
- Communicated with it, an **Apache reverse proxy** will act as an intermediary between the *Kali* and the other web servers. At the beginning of the lab the proxy is not configured yet, so communication between the *Kali* and the backend web servers is not possible (we will enable it in this laboratory). The proxy machine has IPs in two different networks, one to communicate with the *Kali* (`198.102.0.3`) and another to communicate with the web servers (`172.10.0.2`). Apart from the typical folder to share content, content in `/volume_data/rules` is mapped to the `/rules` folder.
- At the other side of the proxy, all communicated with its own private network, we have a pair of **web servers**, `lab10_web_eii` (IP `172.10.0.3`) and `lab10_web_epi` (IP `172.10.0.4`).
- Also, the host *Ubuntu VM* can communicate with any machine in the infrastructure, as it automatically has connection with any network interface we created.

NOTE: Due to the problems that some of you are experiencing with *Ubuntu* repositories and laboratory building in some cases, we have created two scripts that delete all already created *Docker* images. If a `prepare__labXX.sh` script gives you problems, please delete your images and run the script `prepare__labXX.sh` again, to force the system to rebuild all of them, fixing all these problems. To do that, we have developed a pair of scripts that you can find adjacent to this lab report in the *Virtual Campus*. Once downloaded to your VM, to run them you should:

- `sh borrar_imagenes_ubuntu.sh`: To delete *Ubuntu* images
- `sh borrar_imagenes_kali.sh`: To delete *Kali* images



BLOCK 1: INSTALLING A REVERSE PROXY

Apache as a reverse proxy installation

Practical application: *You need the ability to isolate a web server from external access, so the service is accessible, but you do not expose the real server*

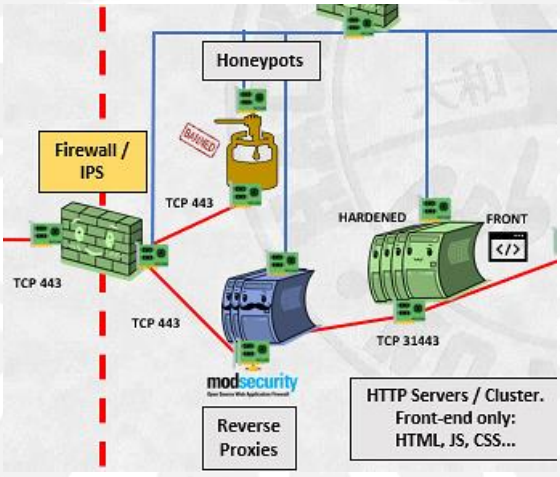
One of the main principles of creating a successful machine infrastructure is **specialization**: each machine has a purpose, and all of them should be specialized and optimized to fulfill it. Like pieces of a puzzle, these servers are put together in the infrastructure to collaborate, so the joined functionality of all of them will achieve a powerful result with a higher level of security.

One of the main components of a secure infrastructure like the one we saw in the theory topics are **reverse proxies**. These servers are a "**barrier**" between the clients and the infrastructure provided services:

- Clients do not really know any detail about the infrastructure that is serving them (how many servers there are, software installed…).
- Clients can only access those services that are intended to them. Other services like internal, auxiliary, or debugging ones cannot be located at the other side of the proxy by any client. Reverse proxies act as "gates" (or unique access points) to public parts of a potentially much more complex infrastructure.

These unique access points are also usually heavily fortified, so they can stop attacks for all servers placed "behind" them. **This means that we have one single defense point for multiple defended servers, saving time and specializing resources**. This way, defended machines should only worry about serving content (while maintaining a minimal security level)

The infrastructure of this laboratory is made to reflect the one presented in the theory topics. Placing a reverse proxy like this "on the front" is a must to prevent damages provoked by the potential vulnerabilities on the web servers. Concretely, we are modelling this part



The proxy container in **Lab 10 infrastructure** **already has** the *Apache2* web server installed (`sudo apt install apache2`), its proxy module (`sudo a2enmod proxy`), and its HTTP proxy module (`sudo a2enmod proxy_http`). Therefore, **the installation part is already covered for you**. To configure *Apache2* to serve as a reverse proxy you must edit the file `/etc/apache2/sites-enabled/000-default.conf` (the default website configuration file) so every request made to this server "from the front" (depending on the specific URL location) will be redirected to each web server "on the back". For example, you can redirect requests to "`<proxy IP>/epi`" and "`<proxy IP>/eii`" to the indicated IPs of both web servers. To ensure transparent reverse proxying of the machines we need to use the directives `ProxyPass` and `ProxyPassReverse` (https://httpd.apache.org/docs/2.4/mod/mod_proxy.html) over the specified locations. Both should point to

the same corresponding server IP. Therefore, we should create two `Location` entries in that file with this structure, both inside the existing `VirtualHost` block:

```
<Location "/<URL>"> #For example: "/eii"
    ProxyPass "http://<Server IP>/"
    ProxyPassReverse "http://<Server IP>/"
</Location>
```

Save, close the file, and restart *Apache2* (`service apache2 restart`). We should check now if it works by issuing requests to any of the URLs from the **Kali** system **using the Reverse proxy's IP**. You can also launch *Firefox* from you *Ubuntu VM*, as it has communication with all the networks we created for this laboratory, to test that the navigation correctly goes to the corresponding web pages, or just using `curl` from the *Kali* container. If you are curious, **more complex proxy configurations can be found here**: https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html

NOTE: Please, do not forget to copy the file `/etc/apache2/sites-enabled/000-default.conf` to `/shared` to preserve it and restore the changes in case the proxy container was accidentally shutdown.

Expected Results: This activity will be finished when you can successfully connect to each web server from the specified URLs of the proxy machine.

BLOCK 2. WEB APPLICATION FIREWALLS

WAF Installation

Practical application: *You need to protect your web applications against multiple common threats without modifying the application's source code*

The *Apache2* reverse proxy in the **Lab 10 infrastructure** also has the `mod_security2` ***Web Application Firewall*** installed to intercept incoming requests directed to our protected systems (`sudo apt install libapache2-mod-security2`).The WAF is installed, but unable to work because it lacks the appropriate rules that indicate it how to examine requests and determine if they are dangerous or not. These rules must be installed, and this is precisely what we are going to do.

Install an updated *OWASP ModSecurity Core Rule Set (CRS)*

To install the latest set of *ModSecurity Core Rules*, we should first move and change the name of the default **ModSecurity** file to use an initial default recommended configuration (`sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf`). Then, **from the Ubuntu VM** download the latest version ***OWASP ModSecurity CRS*** from *Github*: `git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git` (install `git` if necessary, but the *Vagrant* and prebuilt VMs already have it). Copying the downloaded folder to the `/volume_data/rules` folder of the `lab10` infrastructure will make this folder to appear into the container proxy filesystem (`/rules`). Then, we can continue.

*(**NOTE**: We also provided a sample (outdated, but enough for our purposes) set of rules in a zip file as part of the **Lab 10 infrastructure** contents).*

Once inside the proxy, go to the downloaded rules directory and copy and rename `crs-setup.conf.example` to `crs-setup.conf`. Then copy `rules/` as well.

```
cd owasp-modsecurity-crs
cp crs-setup.conf.example /etc/modsecurity/crs-setup.conf
cp -r rules/ /etc/modsecurity/
```

Now we must modify the file `/etc/apache2/mods-available/security2.conf` to match the path of the downloaded files. This means modifying the value of the first `IncludeOptional` to match the `crs-setup.conf` file path (if it does not already have a correct value) and add another `Include` directive pointing to the rule set. The configuration file must be like this (remove other contents). Restart *Apache2* again so that the changes will take effect.

```
        <IfModule security2_module>
                # Default Debian dir for modsecurity's persistent data
                SecDataDir /var/cache/modsecurity

                # Include all the *.conf files in /etc/modsecurity.
                # Keeping your local configuration in that directory
                # will allow for an easy upgrade of THIS file and
                # make your life easier
                IncludeOptional /etc/modsecurity/*.conf
                Include /etc/modsecurity/rules/*.conf
        </IfModule>
```

NOTE: Again, do not forget to copy this file to preserve and restore it in case of errors.

Check that the WAF is up and running

OWASP CRS builds on top of *ModSecurity* so that existing rules can be extended. This is a very complex topic that exceeds the objectives of this course, but we will make a demo to show you how that can be done. Detailed info about rule extension can be found in the **official *ModSecurity* Wiki**: https://github.com/SpiderLabs/ModSecurity/wiki/Reference-Manual-(v2.x)

One of the first things we must consider is that *ModSecurity* is a very complex software, and sometimes **their rules interfere or give false positives over legitimate requests** in our production environment. For example, depending on the rules that have been activated, *ModSecurity* could forbid **access to machines when using their IPs directly**, as this is considered the beginning of an attack attempt (in real situations this is mostly done by scanners, spiders, or bots). In a real situation, this can be prevented using a DNS or modifying the `/etc/hosts` file to give a name to be resolved when accessing the IP. **You should not encounter this problem in this laboratory.**

To check if *ModSecurity* is providing protection, we can make the following tests:

- Go to the default *Apache2* configuration and add two additional directives in the default website configuration file (`/etc/apache2/sites-enabled/000-default.conf`).
  - The first directive enables **ModSecurity** in **interception mode** (`On` detects incoming threats and block them) instead of **detection mode** (`DetectionOnly` detects incoming threats but just log them, which is useful when testing that the WAF is not blocking legitimate requests).
  - The second directive adds a new **ModSecurity** test rule that will be incorporated to the CRS ruleset. This simple rule fires when somebody uses the URL parameter `testarg` with a value that contains the string `ssi`. The response of the WAF is to `deny` the request serving a HTTP 403 `status` error page, logging the incident with the `"SSI test rule fired!"` message. This is just a simple example of how the WAF rules are created, that also checks that the rule engine is working correctly.

```
<VirtualHost *:80>
   <Location …>
     ... # Proxy configuration
   </Location>
     ServerAdmin webmaster@localhost
     DocumentRoot /var/www/html

     ErrorLog ${APACHE_LOG_DIR}/error.log
     CustomLog ${APACHE_LOG_DIR}/access.log combined
     ...
     SecRuleEngine On
     SecRule ARGS:testarg "@contains ssi" "id:1234,deny,status:403,msg:'SSI test rule fired!'"
 </VirtualHost>
```

Restart *Apache2* again and then access the page. If it loads correctly, then intentionally trigger the rule.

Expected Results: This activity will be finished when you can successfully check that *ModSecurity* is running using a custom rule that intentionally fires the engine. You can also find traces of the blocked requests in the appropriate *Apache2* log files (`/var/log/apache2`)

- The second test we are going to make is to test that the CRS rules are being read. To do this, we will trigger an intentional **Remote Command Execution** attack warning, by crafting a request that matches the CRS rules that prevent these kind of web attacks, like passing any parameter with the `/bin/bash` value

Expected Results: This activity will be finished when you can successfully check that *ModSecurity* is running using a fake *Remote Command Execution* (RCE) attack that intentionally fire the engine with the CRS rules. You can also find traces of the blocked requests in the appropriate *Apache2* log files (`/var/log/apache2`). Does the log identify the type of attack that has been attempted?

WAF against "the evil"

Using the following cheatsheet, launch the *Nikto* web server scanner installed in the *Kali* attack container of the **Lab 10 infrastructure** against the proxy to see what happens.

Nikto 2.1.6 Cheatsheet (ingenieriainformatica.uniovi.es)
Lightweight web server vulnerability scanner

https://cirt.net/Nikto2

GENERAL USAGE
nikto -host <url> [options]

NOTES
+ means that the option requires a value
Options in stronger bold font are detailed in separate tables

OPTIONS
-config+: Use this config file
-dbcheck: check database and other key files for syntax errors
-Display+: Turn on/off display outputs
-evasion: Encoding technique to use to avoid WAFs, etc.
-Format+: save file (-o) format
-Help: Extended help information
-host+: target host/URL
-id+: Host authentication to use, format is id:pass or id:pass:realm
-list-plugins: List all available plugins
-mutate: Guess additional file names
-no404: Disables 404 checks
-nossl: Disables using SSL
-output+: Write output to this file
-Plugins+: List of plugins to run (default: ALL)
-port+: Port to use (default 80)
-root+: Prepend root value to all requests, format is /directory
-ssl: Force ssl mode on port
-timeout+: Timeout for requests (default 10 seconds)
-Tuning+: Scan tuning
-update: Update databases and plugins from CIRT.net
-Version: Print plugin and database versions
-vhost+: Virtual host (for Host header)

EXAMPLES
nikto -Display 1234EP -o report.html -Format htm -Tuning 123bde -host 192.168.20.10

EVASION OPTIONS	
1	Random URI encoding (non-UTF8)
2	Directory self-reference (/./)
3	Premature URL ending
4	Prepend long random string
5	Fake parameter
6	TAB as request spacer
7	Change the case of the URL
8	Use Windows directory separator (\)
A	Use a carriage return (0x0d) as a request spacer
B	Use binary value 0x0b as a request spacer

MUTATE OPTIONS	
1	Test all files with all root directories
2	Guess for password file names
3	Enumerate user names via Apache (/~user type requests)
4	Enumerate user names via cgiwrap (/cgi-bin/cgiwrap/~user type requests)
5	Attempt to brute force sub-domain names, assume that the host name is the parent domain
6	Attempt to guess directory names from the supplied dictionary file

TUNING OPTIONS	
1	Interesting File / Seen in logs
2	Misconfiguration / Default File
3	Information Disclosure
4	Injection (XSS/Script/HTML)
5	Remote File Retrieval - Inside Web Root
6	Denial of Service
7	Remote File Retrieval - Server Wide
8	Command Execution / Remote Shell
9	SQL Injection
0	File Upload
a	Authentication Bypass
b	Software Identification
c	Remote Source Inclusion
d	WebService
e	Administrative Console
x	Reverse Tuning Options (i.e., include all except specified)

Expected Results: This activity will be finished when you can successfully check that *ModSecurity* log and stop the *Nikto* scanning attempts. Knowing that *Nikto* is quick, do you think that the WAF is hindering the scanning attempts, so the results came back much slower?

Another test we can make is to fire some of the HTTP `nmap` NSE scripts we tested in the previous laboratory against the proxy and see if the result changes or the attempts get logged by *ModSecurity*.

Expected Results: This activity will be finished when you can successfully check some Nmap NSE scripts that work with web servers against the proxy and see their results. Knowing that *Nmap* is quick in this scenario, do you think that the WAF is hindering the scanning attempts, so the results came back much slower?

WAF against Web attacks

Practical application: *You need to test that your WAF is really blocking requests*

XSS

We will follow a very similar procedure to the previous one to see if the WAF triggers when an **XSS attack attempt** is made. This way, instead of passing the attack using `GET` we will put the attack payload as a `POST` parameter thanks to the `-d` option of the `curl` command. To trigger the WAF we will pass a typical XSS test string: `<script>alert('test')</alert>` with `curl <proxy IP>/ -d "<script>alert('test')</script>"`

Expected Results: This activity will be finished when you can successfully check that *ModSecurity* log and stop XSS attack attempts. Does the log identify the type of attack that has been attempted?

SQL Injection

Like the XSS tests, we perform a **SQL Injection attack detection test** passing as a `POST` parameter the string `DROP DATABASE users;`. Check also that only identifies correct SQL syntax introducing typos in the provided string to see if it captures it or not.

Expected Results: This activity will be finished when you can successfully check that *ModSecurity* log and stop SQL Injection attack attempts. Does the log identify the type of attack that has been attempted?

BLOCK 3. NETWORK INTRUSION DETECTION SYSTEMS

Suricata NIDS

Practical application: *You need to detect potential threats in your network so you can later take actions against them*

This lab exercise could also be placed on the network lab (lab 7), but it has much more sense in combination with a reverse proxy, so it has been placed here. The goal is to introduce you in common surveillance mechanisms used in SOCs and other monitoring infrastructures, so you can get the idea of what it is captured and how it can be used to detect and react against ongoing attacks. This is only a part of this kind of software. Real serious deployments use much more complex infrastructures and software like *Security Onion*.

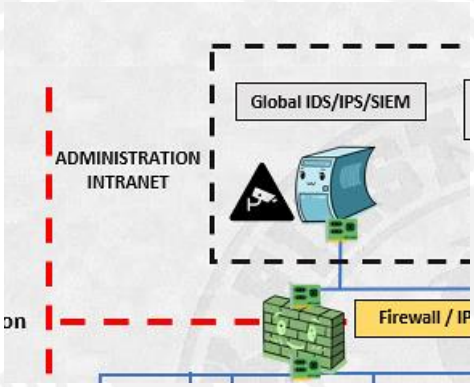
Security Onion and *Suricata*

Suricata is a network-based intrusion detection system (IDS) (and more things 😊) that can be easily installed over an *Ubuntu Server* machine. ***Suricata* monitors network traffic and look for security events that can indicate an attack or compromise**. It performs multi-threaded analysis, natively decode network streams, and assemble files from network streams on the fly. *Suricata* is also a part of a much more complete *Linux* distribution called ***Security Onion*** (https://blog.securityonion.net/). This is a free and open-source *Linux* distribution for threat hunting, enterprise security monitoring, and log management. It includes, among many other security tools, software like (**NOTE**: This information about the included software is only provided in case you are curious, and to show you what it is required to monitor an enterprise-grade network in real life. It is not a part of this lab):

- ***Elasticsearch*** *(https://www.elastic.co/es/):* open source and distributed analysis engine for any type of data, structured or not. It has a REST API, and it is the main component of the popular ***ELK Stack*** (*Elasticsearch*, *Logstash* and *Kibana*), that you already know from the ***Information Repositories*** course, to collect and analyze log information (like the one you will obtain from *Suricata*). *Beats* are agents that send data to the *ELK Stack*.
- ***Logstash*** *(https://www.elastic.co/es/logstash):* another part of the *ELK Stack*, it is an open source and free data processing pipeline that accepts data from a variety of sources, transforming and sending them to other places.
- ***Kibana*** *(https://www.elastic.co/es/kibana):* the third component of the *ELK Stack* is an open source and free GUI to view *Elasticsearch* data and browse the information collected by the *ELK Stack*.
- ***Snort*** *(https://www.snort.org/):* A reference *Open-Source Intrusion Prevention System (IPS)*. Uses a series of rules that help define malicious network activity and to find packets that match against them. It also generates alerts for users or can be deployed inline to stop these malicious packets, as well. *Snort* has three primary uses: As a packet sniffer like `tcpdump`, as a packet logger (useful for network traffic debugging), or as a full-blown network IPS.
- The own ***Suricata*** *(https://suricata-ids.org/):* engine is capable of real time intrusion detection *(IDS),* inline intrusion prevention *(IPS),* network security monitoring *(NSM)* and offline *PCAP* files (stored network traffic) processing. Inspects the network traffic using and has *Lua* scripting support to detect complex threats. It also supports standard input and output formats (like YAML and JSON). It has integrations with tools like existing *SIEMs, Splunk* and the *ELK Stack.* **This is the part we are going to see here**.
- ***Zeek*** *(https://zeek.org/): An Open-Source Network Security Monitoring Tool (NMST)*, formerly known as *Bro.* It is a reference platform for flexible network security monitoring.
- ***Wazuh*** *(https://wazuh.com/):* free, open source and enterprise-ready security monitoring solution for threat detection, integrity monitoring, incident response and compliance.
- ***Sguil*** *(https://bammv.github.io/sguil/index.html): GUI* that provides access to real time events, session data, and raw packet captures. Facilitates *Network Security Monitoring* and event driven analysis.

- ***Squert*** *(http://www.squertproject.org/):* Evolution of *Squil* that adds an improved and more usable web interface to view and perform queries stored in *Squil* databases. It allows to group and view alerts from a *NIDS* (like *Snort* or *Suricata*), alerts from *Host-based Intrusion Detection Systems (HIDS)* like *OSSEC (https://www.ossec.net/), PADs (Passive Asset Detection)* and events from *Zeek*. All this information combined allows a complete view of what happens in an infrastructure and identify how many machines or network connections have been affected by a security incident, for example.
- ***NetworkMiner*** *(https://www.netresec.com/?page=networkminer):* open-source multiplatform *Network Forensic Analysis Tool (NFAT).* It can be used as a passive network sniffer/packet capturing tool to detect operating systems, sessions, hostnames, open ports etc. without putting any traffic on the network. Can also parse PCAP files for off-line analysis and to regenerate/reassemble transmitted files and certificates from them.

All of this allows to create a network of distributed sensors on the network infrastructure of a company, so it will be an ideal candidate to fulfill the IDS role in the SNI we saw in the theory classes:



Unfortunately, **dealing with *Security Onion* is way beyond the objectives of this course**, so we will deal only with *Suricata*. An updated version of *Suricata* from the official repositories (not from sources) **is already installed in the proxy of the Lab 10 infrastructure**, so we have only to configure its rules and test it. This comprises just adding the repositories (`sudo add-apt-repository ppa:oisf/suricata-stable`), update the available ones (`sudo apt update`), and install the software itself (`sudo apt install suricata`).

Install and setup Suricata

Suricata is a signature-based *Intrusion Detection System*, so the next step is to get **rules** to teach it how to deal with traffic. To do that, we can use the ***Emerging Threats*** rules (http://rules.emergingthreats.net/). This is a free repository for *Suricata* rules, although there are other rule sources. To do this, you have to download the rules (`wget http://rules.emergingthreats.net/open/suricata/emerging.rules.tar.gz`) and place them on the `/volume_data/rules` folder of this laboratory so they appear under the `/rules` folder of the proxy machine. There is an old version of these rules on the lab files if you do not want to download them. Once this is done, decompress them (`tar zxvf emerging.rules.tar.gz`) and move the `rules` folder contents to the `/var/lib/suricata/` folder already created in the proxy. This leaves the following routes for interesting files:

- *Suricata* configuration files: `/etc/suricata/`. The main configuration file we are going to modify is `/etc/suricata/suricata.yaml`
- Default *Suricata* rule files folder: `/etc/suricata/rules/` (different from the one we are going to use)
- *Suricata* log files (to see what threats it identifies): `/var/log/suricata/`. Here the `fast.log` file contains the triggered rules. Another important log file is `eve.json`, but we are not going to use them in this laboratory.

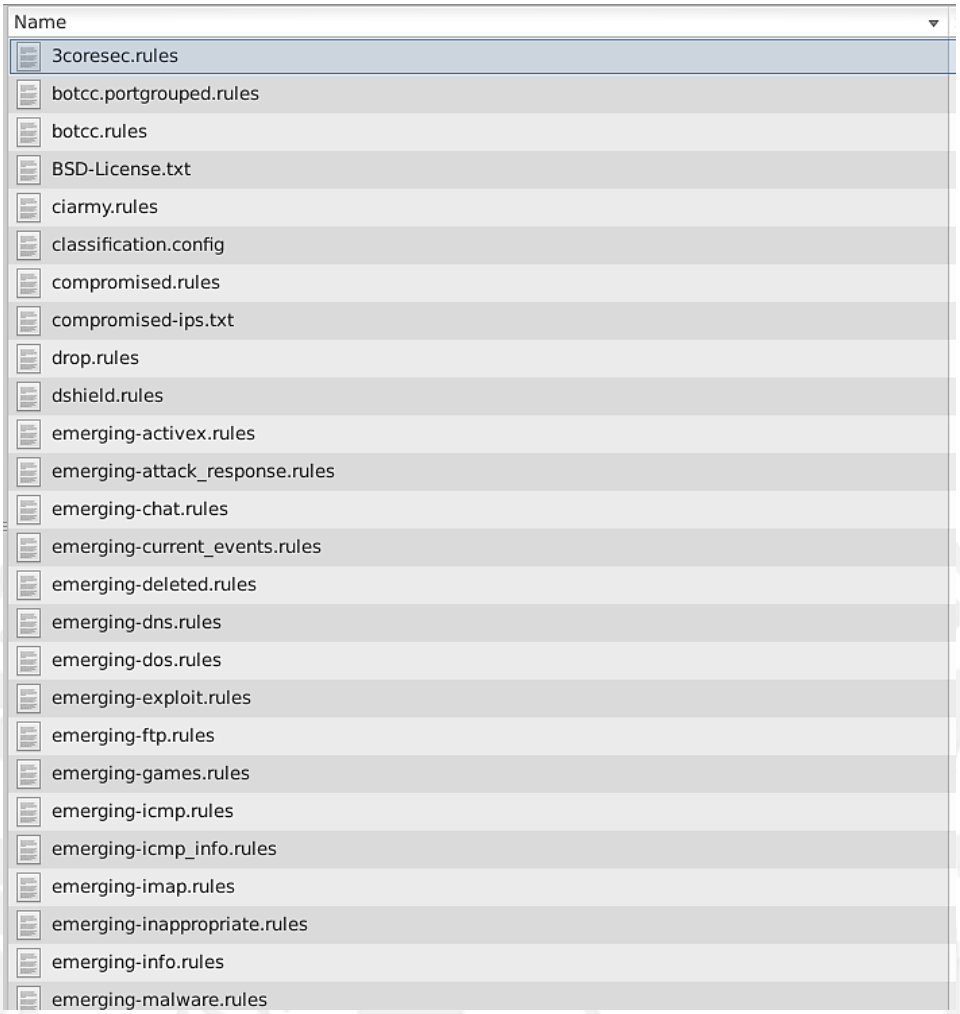
Expected results: This activity will be finished when *Suricata* has the *Emerging Threats* rules correctly placed in the intended location, and all the necessary *Suricata* elements have been located to ensure the installation is correct.

Configure Suricata

The `/etc/suricata/suricata.yaml` file must be modified to finish the NIDS configuration. The first thing we must do is to ensure that the IP to protect is listed as the one in the `HOME_NET` variable, as this allows the rules to know what is external and what is local traffic. Note that this setting accepts networks (CIDR notation), but as we are a "single entry point" reverse proxy machine, and not a firewall that controls access through different networks, in this case a single IP is the correct setting. **Set this variable to the proxy IP that is in the same network that the attacking *Kali* container** (`lab10_front_net`) in the **Lab 10 infrastructure**. Other IPs will be treated as the `EXTERNAL_NET`. Consequently, **make sure that the network interface that must be surveilled by *Suricata* is the correct one**: most likely you are using `eth1` (not `eth0`) to communicate with the "external world" (the *Kali* container), so you must locate the interface name in this file and change it accordingly.

Next, we must enable the rules that *Suricata* is going to use to examine traffic. To do this, we must ensure two things:

- That in the previous file, the `default-rule-path` parameter points to the folder in which we copied the *Emerging Threats* rules (`default-rule-path: /var/lib/suricata/rules`)
- To choose the rules we want to activate and list them in a different line (preceded by `-`) bellow the `rule-files:` entry. To do that, examine the file names and content description of the rules you downloaded, and choose the ones that you think it will be appropriate for your current setup. To test *Suricata*, **please ensure** that at least `emerging-user_agents.rules` and the `emerging-scan.rules` are included.



- Finally, restart *Suricata* to enable the new configuration: `service suricata restart`

NOTE: As in previous modified files, we recommend you save it just in case copying the file to `/shared`.

Expected results: This activity will be finished when *Suricata* has the appropriate *Emerging Threats* rules enabled to match the expected services the proxy machine must deal with, and the networking environment of this laboratory, differentiating them from the rules that could be desirable in a real setup.

Test Suricata Functionality

To test that *Suricata* is working we are going to perform two tests.

1. **The first test comprises** a suspicious connection from the proxy to the "outside" of the proxy (for example the **Ubuntu VM**). This connection meets two conditions:
   - It needs a connection target, and we can easily provide one if we install an *Apache* web server in our **Ubuntu VM** (if not already done) and try to send a request to it (install it with `sudo apt install apache2`)
   - It fakes a request of a known trojan: *ChilkatUpload*, modifying the request *User-Agent*. To do that: `curl -A "ChilkatUpload" 198.102.0.1` (assuming this is the IP of the VM in the `lab10_front_net` network)

   Apart from that, you may be wondering how our proxy can send a suspicious request to the outside. There are two possible reasons for this: either the proxy itself is compromised or the proxy is also a **forward proxy** and one of the machines it supervises is compromised. What is a forward proxy? The opposite of what we defined as a reverse proxy: machines that are linked to a forward proxy use

it as the only means to achieve connections "to the outside" (for example, Internet), and therefore every request of these machines made to external ones will pass through the proxy, that may be able to monitor them with *Suricata* and proper configuration. It is the same philosophy of the *Burp* proxy we saw in seminars but applied to machine networks instead of single machines.

Expected results: This activity will be finished when *Suricata* detects and logs the request that is run from the proxy to the outside, detecting a Trojan. You also understand why this can be used to detect compromised machines behind the proxy.

2. **The second test** performs a request from "outside" the proxy to the proxy itself. In this case we will use a "noisy" `nmap` scan that performs standard detection of service versions and standard operating system detection (see the network enumeration lab to see how to do this scan).

Expected results: This activity will be finished when *Suricata* detects and logs the `nmap` scan attempt that is run from the outside to the proxy.

Keep in mind that this is just a basic (but working) installation of *Suricata* on *Ubuntu 18.04* that gets you started in how *Network Intrusion Detection Systems* operate. Much more complex options and operation modes can be used by consulting their documentation page. A real-world deployment requires additional supporting software, that you can have with an installation of the previously mentioned ***Security Onion*** *Linux*.

BLOCK 4: AUTOMATED SCA AND SAST IN APPLICATION DEVELOPMENT

SonarQube Automated Application Vulnerability discovery tool (SAST, with pluggable SCA module)

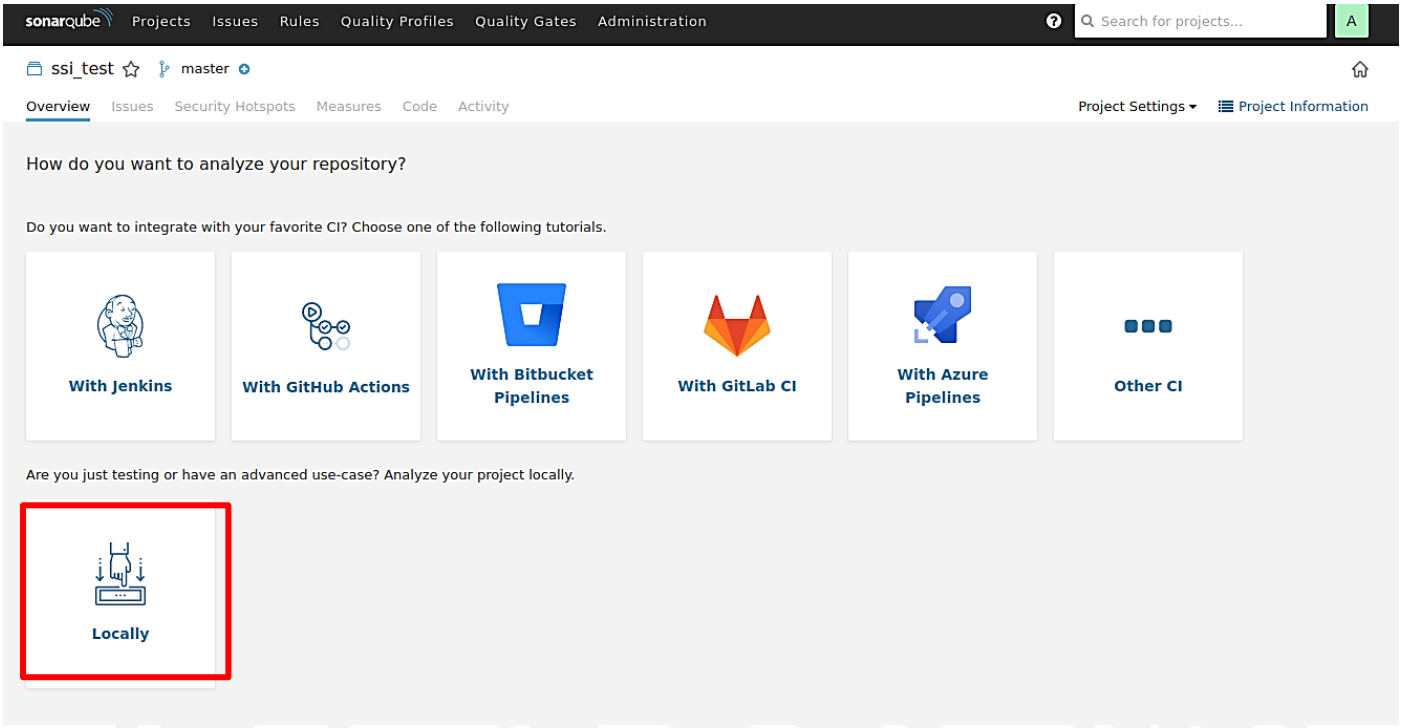
Practical application: *You need to automatically detect dependency and source code vulnerabilities in your applications*

SonarQube (https://www.sonarqube.org/) is an automated code quality and security review tool that performs static code analysis (**SAST**, as shown in the theory topics) of our application source code. It supports around 20 different programming languages (mainly *Java* or *C#*) looking for bugs, vulnerabilities, security hotspots, code smells, code quality metrics, etc. It integrates with *Maven*, *Gradle*, and other build platforms also used in other courses such as **ASW** or **SDI** so, if our project uses them, *SonarQube* will be very easy to setup and use. It also accepts modules that enable it to perform other functionalities. The following two exercises does not require the **Lab 10 infrastructure** running, so please stop it to save resources.

Automated SAST with *SonarQube*

This tool can identify bugs that have escaped our manual review and tests. This is the **IPS** course domain, so in this laboratory we will focus on finding vulnerabilities that may be present in the source code. To use *SonarQube* we need to follow these steps **from your Ubuntu VM**:

- Provide a project in a supported language to analyze (preferably in *Java* or *C#*, using *Maven* or *Gradle*). If you do not have one of your own, you can download the zip file from this repository (https://github.com/appsecco/dvja) to test how this tool works. This zip is also provided as part of the **laboratory 10 materials**. Unzip the file in a known folder of your VM. From now on we will assume that this test project is used, if you use your own one you need to adapt these instructions.
- Install *Maven* **in the Ubuntu VM** (`sudo apt install maven`)
- Run a *Sonarqube* container by running the `run_sonarqube.sh` script that we include in the **laboratory 10 materials**. It will download 0.5Gb of information and may take some time to complete.
- Log in to http://localhost:9000 with the *System Administrator* credentials (login=`admin`, password=`admin`). Changing the password on first login is mandatory.
- To analyze a project:
  - Click in *Create new project manually*.
  - Give the project a *Project key* and a *Display name* and click the *Set Up* button (ex.: `SAST_SSI`).
  - Click on *Analyze your repository locally*.



- See the generated token and click continue. Then, choose *Maven*. This is because our test project builds with *Maven*, but, as you see, there are many other options you can use in the future (like for example, your TFG 😉).
- Now you will see a command to analyze your application associated with the *Sonarqube* project you just created. Note that the token and the project key are the ones that were generated when you set up the analysis.
- Go to the project folder that you unzipped in your VM and run the command that the interface of *SonarQube* indicates to analyze the project inside this folder. The command will be something like this.

```
mvn sonar:sonar \
  -Dsonar.projectKey=testPrimeroSSI \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.login=686c129bbb6ac40a02534d7eb02fb8393037482c
```

- Once the analysis ends, the *SonarQube UI* will automatically update with the analysis results. You will have *Bugs* and *Code Smells* categories with elements that are more related with ***IPS*** or ***CPM***. The categories that belong to our course are *Vulnerability* **and the *Security Hotspots* tab**.
- Open the *Security Hotspots* tab and see the 7 possible problems it reports. Answer these three questions with the information the tool provides:
  1. Do you think that the *Authentication* security issue is real or a false positive?
  2. According to the description of *SQL Injection* vulnerabilities made in the **theory topic 6**, do you think that the related security issues identified can be potentially dangerous?
  3. According to what we reviewed in the **theory topic 2** (*cryptography*), do you think that the application is using a strong hashing method?

Expected results: This activity will be finished when you are able to successfully perform a SAST analysis of an application with *SonarQube* to search for security issues and vulnerabilities and correctly interpret the findings.

Automated SCA with *SonarQube*

NOTE*: *If it is possible, try to increase the RAM of your VM to at least 4Gb for this exercise. If can be doable with the default VM, but it will exhaust the VM RAM in the process and crash after generating the necessary report. The following description assumes you cannot increase the RAM.

Sonarqube is a SAST tool, and it does not perform SCA functionalities by itself. However, its plugin system allows us to incorporate a complete and reputable SCA tool into the build pipeline. This tool is the *OWASP Dependency Check* software: https://owasp.org/www-project-dependency-check/. To do this follow these steps:

- Shut down the VM and, if possible, increase its RAM.
- Boot the VM again and login as you normally do. Locate the file `pom.xml` of our test project (in the folder in which you previously unzipped it), open it and ensure you added the lines that appear in red here:

```
        …
        <plugin>
              <groupId>org.eclipse.jetty</groupId>
              <artifactId>jetty-maven-plugin</artifactId>
              <version>9.3.0.M2</version>
              <configuration>
                  <stopKey>CTRL+C</stopKey>
                  <stopPort>8999</stopPort>
                  <systemProperties>
                      <systemProperty>
                          <name>xwork.loggerFactory</name>
                          <value>com.opensymphony.xwork2.util.logging.log4j2.Log4j2LoggerFact
ory</value>
                      </systemProperty>
                  </systemProperties>
                  <scanIntervalSeconds>10</scanIntervalSeconds>
                  <webAppSourceDirectory>${basedir}/src/main/webapp/</webAppSourceDirectory>
                  <webAppConfig>
                      <descriptor>${basedir}/src/main/webapp/WEB-INF/web.xml</descriptor>
                  </webAppConfig>
              </configuration>
          </plugin>
          <plugin>
            <groupId>org.owasp</groupId>
            <artifactId>dependency-check-maven</artifactId>
            <version>6.5.2</version>
            <executions>
                <execution>
                    <goals>
                        <goal>check</goal>
                    </goals>
                </execution>
            </executions>
          </plugin>
      </plugins>
…
```

- Repeat the same process of the previous SAST analysis to launch *Sonarqube* analysis. Now, as part of the *Sonarqube* automated analysis, it will also perform a SCA analysis of the test project dependencies. This requires a substantial amount of time (it needs to download all the available CVE information) and may also crash due to lack of resources, especially if you couldn't increase the RAM of the VM. In any

case, a report will be generated in: `<the folder in which you unzipped the test application>/dvja-master/target/dependency-check-report.html`

- Analyze the report: are there vulnerable dependencies? Can you obtain information about each vulnerability? (description, score…)
- Shutdown the VM again and restore the RAM to its original amount if appropriate (If you changed it).

Expected results: This activity will be finished when you are able to successfully analyze an application with *SonarQube* to locate vulnerable dependencies and analyze the generated report with information about them.

Combining the proxy with previous techniques

Once we have this hardened proxy on place, and by taking advantage of the activities we made in previous labs, think about how you would combine them so the protection could be even greater in a real scenario. To do that, answer these questions:

- What will you do to the base OS of the proxy?
- If the proxy must have a public `ssh` remote connection installed for administration reasons, what additional security measures will you enable for this service?
- Now that we have an exposed web proxy, will you install `fail2ban` in the proxy? If the answer is yes, what additional steps will you take from the ones we saw in a previous lab?
- Will you allow `nmap` scans on this machine? What will you do to avoid them?
- If you want to get rid of the public `ssh` connection in the proxy, but still be able to connect to it through `ssh` for administration purposes, what will you do?
- If you are worried about vulnerabilities in the code of your applications, what countermeasures will you use to prevent them as much as possible?
- If you can answer the previous questions, what do you think about the security of the resulting machine and the protection it provides to the underlying infrastructure?

Expected Results: This activity will be finished when you know what to do to provide further hardening of the proxy machine using techniques we saw in previous laboratories.

BADGES AND SELF-ASSESSMENT

NOTE: You have a version of this badge table in an editable format available in the *Virtual Campus*. You can use this file to easily create a document in the format you want and take extended notes of your activities to create a log of what you did. Remember that this material is key to keep track of your self-evaluation.

Badge Level	Unlocked when	Unlocked?
	You understand what a reverse proxy is. You can answer this question: *Does a reverse proxy really serve websites by itself?*	
	You understand what the main functionality of a WAF is	
	You know how to launch web scans using `nikto`	
	You know what *Security Onion* is	
	You can check if *ModSecurity* also provides protection against some `nmap` scans	
	You know how to update *Suricata* rules	
	You understand the purpose of a SAST tool and the type of results it outputs	
	You understand the purpose of a SCA tool and the type of results it outputs	
	You can answer these questions: *Does a reverse proxy completely isolate a serving infrastructure from its potential clients? What is the reason of using two totally different network ranges?*	
	You can answer these questions: *Is the custom rule we used testing the ModSecurity engine itself or the Core Rules we downloaded? Is this enough to test that the whole infrastructure is working?*	
	You can answer these questions: *Is the fake RCE attack we created testing the ModSecurity engine itself or the Core Rules we downloaded? Is this enough to test that the whole infrastructure is working?*	
	You can answer these questions: *Are ModSecurity blocked requests logged? In that case, In which file?*	
	You can answer this question: *Does a WAF protect against automated attack attempts?*	
	You can answer this question: *Does a WAF protect against typical web attack (XSS, SQL Injection…) attempts? Does it identify the type of attempted attack? Can therefore a potential GUI that reads the WAF logs show its users the attack types that are being produced?*	
	You understand the purpose of the rules of a NIDS system. You can answer these questions: *is the amount and quality of the rules that we provide to a NIDS the key to the effectiveness of their detection capabilities? Do you think that investing (e. g. pay a subscription) in proven and reputable rules from trusted maintainers is worth it?*	
	You understand how a NIDS and a forward proxy can be combined to detect compromised machines in an internal network	

	You can answer these questions: *What determines the number of rules we need to activate in an IDS? Do you think that public-facing servers (internet) and those in LANs should use the same rules?*	
	You can analyze an application source code using *SonarQube*. You can answer these questions: *Do you think that this type of tools locates all possible errors? If not, do you think their usage is convenient?*	
	You can successfully setup a working *ModSecurity 2* WAF protecting a server infrastructure	
	Considering that a WAF identifies an attack only if the syntax of the language is correct (XSS, SQL Injection…), reason about what might be the main technique that can be used to bypass a WAF protection	
	You can answer these questions: *Why is a reverse proxy important from a security point of view? Do you think that the concept is like encapsulation in object-oriented programming?*	
	You can answer this question: *Do you think that WAFs provide ultimate attack protection, or it should be treated as an extra layer of protection?*	
	You know how to combine techniques we saw in previous laboratories to further harden the proxy in more scenarios	
	You can answer these questions: *Does Suricata detect potential problems, or does it also block offending requests? Do you understand the difference between Intrusion Detection Systems and Intrusion Prevention Systems?*	
	You can answer these questions: *What do you have to do to detect a threat using Suricata while it is happening? Is it effective to do it manually? What type of tools do you think you will need to do it? Do you understand why integrated software bundles like the one offered in Security Onion are necessary then?*	
	You clearly understand the type of vulnerabilities that may be prevented by a WAF like *ModSecurity* vs the ones prevented by a static analysis tool like *SonarQube*	
	**Hold the door!** You can protect a web server using a WAF-fortified reverse proxy with intrusion detection capabilities, and locate vulnerabilities in your code or its dependencies deploying automated SCA and SAST tools	