Source: Bing Chat AI

LAB 10. DEFENDING WEB APPLICATIONS

COMPUTER SCIENCE DEPARTMENT. UNIVERSITY OF OVIEDO

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

Content

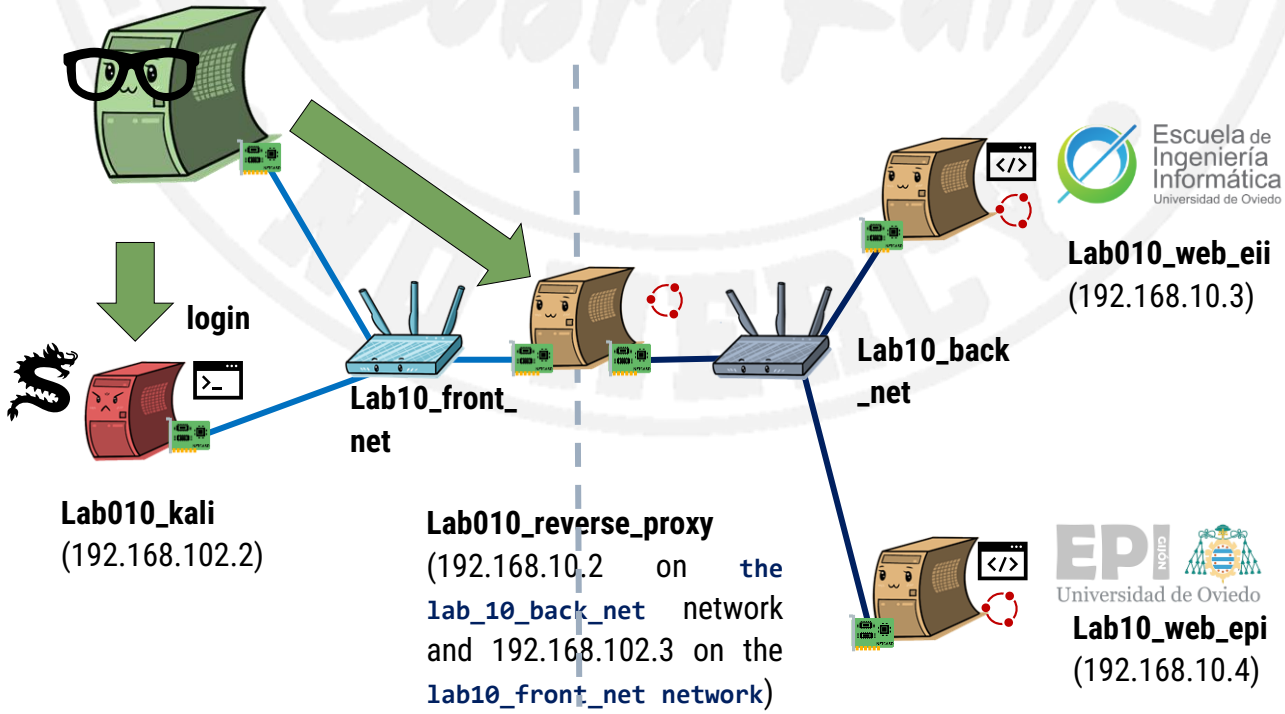
Infrastructure of this laboratory

The **Lab 10 infrastructure** provided follows the same conventions of previous labs, and has the following containers whose IPs are shown in the diagram:

- A **Kali** "attack container" with tools to "attack" the proxy. It only communicates with the proxy.
- Communicating with it, an ***Apache 2* 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 yet configured, so communication between *Kali* and the *backend* web servers is not possible (we will enable it in this lab). The proxy machine has IPs on two different networks, one to communicate with the *Kali* and one to communicate with the web servers. In addition to the typical resource sharing folder, the contents of `/volume_data/rules` are mapped to the `/rules` folder.
- On the other side of the proxy, communicating with its own private network, we have a couple of **web servers,** `lab10_web_eii` and `lab10_web_epi`.
- In addition, the *Ubuntu* virtual machine can communicate with any machine in the infrastructure, as it automatically has a connection to any network interface that we have created.
- **NOTE**: To avoid permission issues during exercises, it is recommended to work as `root` (`sudo su`) on both the *Kali* container and the proxy.

NOTE: If you have problems with space or building the labs, we have created two scripts that delete all *the Docker images* already created. If a `prepare__labXX.sh` gives you problems, please delete all the images and run `prepare__labXX.sh` again to force the system to rebuild everything, which should fix all the problems. These scripts are on the *Virtual Campus*. Once downloaded to the VM, to run them you should do:

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



Block 1: Installing a reverse proxy

Exercise L10B1_REVPROXY: *Apache 2* installed as a reverse proxy

Description of the activity / Practical application

You need to know how **to isolate a web server from outside access** so that a service is accessible without exposing its actual server

Expected results

This activity will end when you can successfully connect to each web server **from the specified URLs on the proxy machine**.

Additional information required to do it

One of the fundamental principles of building a successful machine infrastructure is **specialization:** every machine has a purpose, and all machines **must be specialized and optimized to fulfill it**.

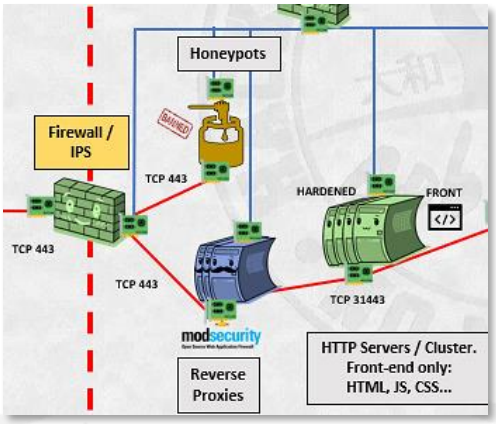
🔍 ***Like pieces of a puzzle, these servers come together in an infrastructure to collaborate with each other, so the joint functionality of all of them will achieve a more complete result with a higher level of security.***

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

- Customers cannot really know any details about the infrastructure that is serving them (how many servers there are, software installed...).
- Customers can only access **those services that have been created to be public**. Other services such as internal, auxiliary, or debugging services cannot be located on the other side of the proxy by any client. Reverse *proxies* act as "**gateways**" (or single access points) to public parts of a potentially much more complex infrastructure.

These single access points are also usually heavily fortified, so they can stop attacks for all servers placed "behind" them. **This means that we have a single point of defense for multiple defended servers, saving time and making resources specialized.** In this way, defended machines only have to worry about serving content (maintaining, of course, a minimum level of security).

The infrastructure of this laboratory is made to reflect the one presented in the theory topics. Placing a reverse proxy like this "at the front" is a necessity to avoid problems caused by potential vulnerabilities in the web servers. Specifically, we are modeling this part.



The proxy container in **the Lab 10 infrastructure** **already has** the *Apache 2* web server installed (`sudo apt install apache2`), its proxy module (`sudo a2enmod proxy`) and its HTTP proxy module (`sudo a2enmod proxy_http`). Therefore, related **to the installation, there is nothing to be done**.

To configure *Apache 2* to serve as a reverse proxy, you need to edit the `/etc/apache2/sites-enabled/000-default.conf` file (the default website configuration file) so that every request made "from the front" to this proxy server is redirected to each "behind" web server depending on the specified URL.

For example, you can redirect requests to "`<Proxy IP>/epi`" and "`<Proxy IP>/eii`" to the IPs listed in the figure for both web servers. To ensure a **transparent** reverse proxy of the machines, we need to use the `ProxyPass` and `ProxyPassReverse` (https://httpd.apache.org/docs/2.4/mod/mod_proxy.html) directives on the IPs of the internal network. **Both should point to the same IP of the corresponding server**. Therefore, we need to create two `Location` entries in that file with this structure, both within the existing `VirtualHost` block:

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

Save and close the file and restart *Apache 2* (`service apache2 restart`). Now we need to check if it works by making requests to any of the URLs seen from the ***Kali* system using the reverse proxy IP.** You can also run *Firefox* from your *Ubuntu VM*, as it has communication with all the networks we created for this lab, to test that navigation goes correctly to the corresponding web pages, or simply by using `curl` from the *Kali container*. If you are curious, you can find **more complex proxy setups here:** https://httpd.apache.org/docs/2.4/howto/reverse_proxy.html

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

Block 2. *Web Application Firewalls* (WAFs)

Exercise L10B2_WAF: Installing a WAF

Description of the activity / Practical application

You need **to protect your web applications against various types of threats** without changing the source code of the application

Expected results

This activity will end when you can successfully check the following:

- That *ModSecurity* **is running**, using a custom rule, and causing it to fire. You can also find blocked requests in the appropriate Apache 2 log files (`/var/log/apache2`)
- That **the *ModSecurity* rules engine *is being used***, using a fake RCE-type attack that intentionally activates the rules engine with the CRS rules. You can also find blocked requests in the appropriate *Apache 2* log files (`/var/log/apache2`). *Does this log identify the type of attack that has been attempted?*
- **That *ModSecurity* logs and stops *Nikto's* scanning attempts**. Knowing that *Nikto* is quite fast, *do you think the WAF is hindering the scanning attempts, and the results were obtained much more slowly?*
- You can **launch some *NMap NSE* scripts that work with web servers** against the proxy and see their results. Knowing that *NMap* is fast in this scenario, *do you think the WAF is hindering scanning attempts, so the results were obtained much slower?*

Additional information required to do it

The *Apache 2* reverse proxy in **the lab 10 infrastructure** also has the ***Web Application Firewall installed*** `mod_security2` to intercept incoming requests directed to our protected systems (installed with `sudo apt install libapache2-mod-security2`). The WAF is installed, but it cannot work because it **does not have proper rules** that tell it how to examine requests and determine whether or not they are dangerous. These rules must be in place, and this is precisely what we are going to do now.

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

To install the latest *OWASP ModSecurity Core Rule Set* we must first move and rename the following ***ModSecurity*** file to have an initial default recommended configuration (`sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.conf`). Then, from the **Ubuntu *VM*** we download the latest version of ***OWASP ModSecurity CRS*** from *GitHub:* `git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git` (install `git` if needed, but *Vagrant* and pre-built VMs already have it). Copying the downloaded folder to `/volume_data/rules` in the `lab10` infrastructure will cause that folder to appear in the file system of the proxy container (`/rules`). Only then can we continue.

> 🔍 ***NOTE:** We also provide you with a sample set of rules (outdated, but enough for our purposes) in a zip file as part of the Lab 10 infrastructure content*

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

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

Now we need to modify the `/etc/apache2/mods-available/security2.conf` file to match the path of the downloaded files. This means modifying the value of the first `IncludeOptional` to match the path of the `crs-setup.conf file` (if it does not already have a correct value) and adding another `Include` directive that points to the ruleset. The configuration file should look like this (remove other content it may have). Restart *Apache 2* again for the changes to 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 keep and restore in case of errors.*

Check that the WAF is up and running

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

One of the first things we must keep in mind is that *ModSecurity* is a very complex piece of software, and sometimes **its rules interfere or give false positives on legitimate requests** in our production environment.

> 🔍 ***For example, depending on the rules that have been activated, ModSecurity could prohibit access to machines when their IPs are used directly to do so, as this is considered the beginning of an attack attempt (in real situations this is mainly done by scanners, spiders, or bots). In a real-world situation, this can be avoided by using a DNS or by modifying the `/etc/hosts` file to give a name that will resolve when accessing the IP. You should not find this problem in this lab.***

To check if *ModSecurity* on the proxy is giving protection to the servers on the *back-end*, we can perform the following tests:

- Go down to the default *Apache 2* settings and add two additional policies in the default website configuration file (`/etc/apache2/sites-enabled/000-default.conf`).
    - The first policy enables ***ModSecurity*** in **intercept mode** (`On` detects incoming threats and **blocks them**) instead of **detection mode** (`DetectionOnly` detects incoming threats but only **logs them**, which is useful when proving that the WAF is not blocking legitimate requests).
    - The second policy adds a new test rule to ***ModSecurity*** that will be incorporated into the CRS ruleset. This rule is very simple and is triggered when someone uses the `testarg` parameter in a URL with a value that contains the `ssi` string. The WAF's response is to `deny` the request serving an error page with the status HTTP 403 (`status:403`), logging the incident with the message `"SSI test rule triggered!"`. This is just a simple example of how WAF rules are created, which also verifies that the rules engine is working properly.

```
 <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
triggered!'"
  </VirtualHost>
```

Restart *Apache 2* again, and then access the page. If loaded correctly, **intentionally trigger the rule to finish this part of the exercise**.

- The second test we are going to do is **check that the CRS rules are being read**. To do this, we purposely trigger a **remote command execution** (RCE) attack warning, creating a request that matches the CRS rules that prevent this type of attack, such as passing any parameter with a value `/bin/bash`

WAF against "evil"

Using the *cheatsheet* below, run *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)

Another test we can do is to use some of the nmap NSE HTTP scripts that we tested in the previous lab against the proxy and see if the result changes or the attempts are logged by *ModSecurity*.

Exercise L10B2_WAFPROT: WAF against web attacks

Description of the activity / Practical application

You need to check if **the WAF is actually blocking malicious web requests,** like the ones mentioned in the last theory topic

Expected results

This activity will end when you can successfully verify that *ModSecurity* **logs and stops attacks like XSS and SQL injection attempts**. *Does the log identify the type of attack that has been attempted?*

Additional information required to do it

XSS

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

SQL Injection

Like the XSS tests, we perform a **SQL injection attack detection test** by passing as a `POST` parameter to users of the `DROP DATABASE chain;`. Also check that it only identifies the correct SQL syntax by entering typos in the provided string to see if it captures it or not.

Block 3: AST tools for application development

Exercise L10B3_SASTSCA: *SonarQube automated application vulnerability detection tool* (SAST, with optional SCA module)

Description of the activity / Practical application

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

Expected results

This activity will end when you are able to do these two operations in addition to answering the questions that arise throughout the exercise:

- **Successfully perform a SAST scan** of an application with *SonarQube* to look for security issues and vulnerabilities and correctly interpret the findings.
- Successfully analyze an application with *SonarQube* to **locate vulnerable dependencies** and analyze the generated report with information about them.

Additional information required to do it

SonarQube (https://www.sonarqube.org/) is an **automated code quality and security review tool** that performs **static code analysis (SAST,** as explained in the theory topics) of our application's source code.

It supports around 20 different programming languages (mainly Java or C#) and looks for bugs, vulnerabilities, security hotspots, code smells, code quality metrics, etc.

It integrates with *Maven,* Gradle*, and other app building platforms also used in other courses such as* **ASW** or **SDI**, so if your projects use them, *SonarQube* will be very easy to set up and use. It also accepts modules that allow you to do other functionalities. The next two exercises do not use the **infrastructure of Lab 10**, so it is best to stop it to save resources.

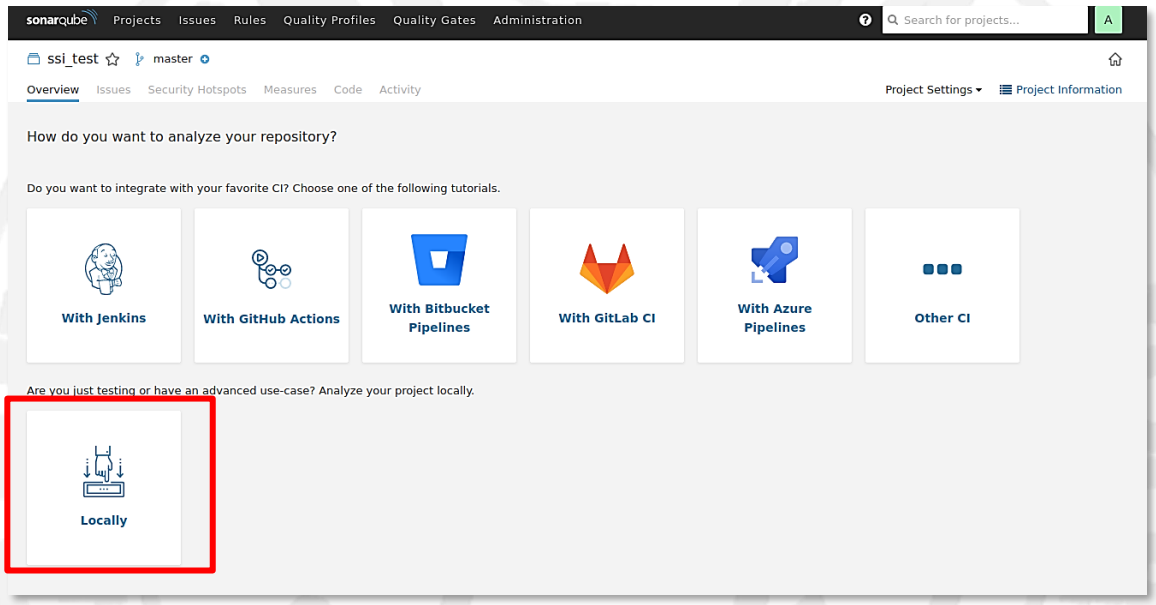
Automated SAST with *SonarQube*

This tool can identify bugs that have not been caught in our manual review and testing. This is the objective of the IPS course, so in this lab 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*** virtual machine:

- Choose a project in a supported language to parse (preferably in *Java or* C#, *using* Maven *or* Gradle). If you don't 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 **Lab 10** materials**.** Unzip the file into a folder that you can access after the virtual machine. From

now on we will assume that we are going to use this test project, if you use your own you must adapt the instructions to your case.

- Install *Maven* **on the *Ubuntu VM*** (`sudo apt install maven`)
- Run a *SonarQube container* by running the `run_sonarqube.sh` script we included in **Lab 10** materials. It downloads 0.5Gb of information and takes some time to complete.
- Log in to http://localhost:9000 with your system administrator credentials (login=`admin`, password=`admin`). Changing your password on your first login is mandatory.
- To analyze a project:
  - Click ***Create new project manually***.
  - Assign the project a ***Project key*** and a ***Display name*** and click on the ***Configure button*** (e.g., `SAST_SSI`).
  - Click ***Analyze your repository*** *locally*.



  - Look at the generated token and click continue. Then, choose ***Maven***. This is because our test project is built with *Maven* but, as you can see, there are many other options that you can use in the future (such as your TFG 😉)
  - **It will now show you a command to analyze the application associated with the *SonarQube* project you just created**. Note that the project token and key are the ones that were generated when you set up the analysis.
  - Go to the project folder you unzipped into your virtual machine and run the command that the *SonarQube interface showed you* inside it to analyze the project. The command will look something like this.

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

- Once the analysis is complete, the *SonarQube user interface* will automatically update with the results of the analysis. You will have categories of ***Bugs*** and ***Code Smells*** with elements that are more related to **IPS** or **CPM**. The categories most closely linked to our course are ***Vulnerabilities*** and the ***Security Hotspots*** tab.

- Open the ***Security Hotspots*** tab and check out the 7 potential issues it reports. Answer these three questions with the information provided by the tool:
    1. *Do you think the Authentication security issue is real or a false positive?*
    2. According to the description of SQL injection vulnerabilities given in theory **Topic 6,** *do you think the related security issues identified by the tool can be potentially dangerous?*
    3. Based on what we saw in theory **Topic 2 (** *cryptography) topic, do you think the app is using a strong hashing method?*

Automated SCA with *SonarQube*

🔍 ***NOTE:*** ***If possible, try to increase the RAM of the virtual machine to at least 4 Gb for this exercise. This can be done with the default VM (2 Gb), but it will drain the VM's RAM during the process and crash after generating the report, so you will need to restart it. The following description assumes that you cannot increase RAM.***

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

- Shut down the VM and, if possible, increase its RAM.
- Reboot the VM and log in as you normally do. Locate the `pom.xml` file from our test project (in the folder you unzipped it to earlier), open it, and make sure to add 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.Log4j2Log
gerFactory</value>
                      </systemProperty>
                  </systemProperties>
                  <scanIntervalSeconds>10</scanIntervalSeconds>
                  <webAppSourceDirectory>${basedir}/src/main/webapp/</webAppSourceDire
ctory>
                  <webAppConfig>
                      <descriptor>${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 from the previous SAST scan to start the *SonarQube* scan. Now, as part of *SonarQube*'s *automated* analysis, an SCA analysis of the test project's dependencies will also be done. This requires a substantial amount of time (you need to download all available CVE information) and can also crash due to lack of resources, especially if you were not able to increase the RAM of the virtual machine. In either case, a report will be generated in: `<the folder where you unzipped the test app>/dvja-master/target/dependency-check-report.html`
- Analyze the report: *Are there vulnerable dependencies? Can you get information about each vulnerability?* (description, rating...)
- Shut down the VM again and restore the RAM to its original value if you want (and if you changed it).

BACK
1
2
3

Exercise L10B3_HARDPROXY: Combining proxies with previous techniques

Description of the activity / Practical application

Understand how a proxy machine can be a strategic point **where you can increase the security measures of your infrastructure** over other parts of it.

Expected results

This activity will end when **you know what to do to provide greater security to the proxy machine** using techniques we saw in previous labs.

Additional information required to do it

Once we have this security-enhanced proxy created, and taking advantage of the activities we did in previous labs, think about how you would combine them so that the protection can be even greater in a real-world scenario. To do that, answer these questions:

- *What would you do with the operating system of a proxy machine?*
- If the proxy must have a public `ssh` remote connection installed for management reasons, *what additional security measures would you enable for this service?*
- Now that we have an exposed web proxy, *would you install* `fail2ban` *on it?* If the answer is yes, *what additional steps would you take from what we saw in a previous lab?*
- *Would you* allow `nmap` scans on this *machine, and what would you do to prevent them?*
- If you do not want a public `ssh` connection on the proxy, but still want to be able to connect to it via `ssh` for management purposes, what would you do?

- If you are concerned about vulnerabilities in your application code, *what countermeasures would you use to prevent them as much as possible?*
- If you can answer the questions above, *what do you think about the security of the resulting machine and the protection it provides to the underlying infrastructure?*

Exercise L10B3_DAST: Using a DAST against an application

Description of the activity / Practical application

This activity teaches you how **to DAST test a web application**.

Expected results

This activity is considered complete **when you are able to use ZAP to do a DAST scan of a testing website**, both actively and manually, also using the aggressive mode to see if problems are detected. You are also able to understand the type of errors that can be discovered against SAST or SCA solutions and interpret the reports generated. You can answer these questions:

- *Do you think it is interesting to introduce tools like this into a CI/CD pipeline to build your application?*
- *Do you think it is interesting to present a ZAP report made on the final version of your product or TFG to your investors/clients/ evaluation tribunal?*

Additional information required to do it

Using ZAP as a DAST requires a series of steps that are going to be explained below.

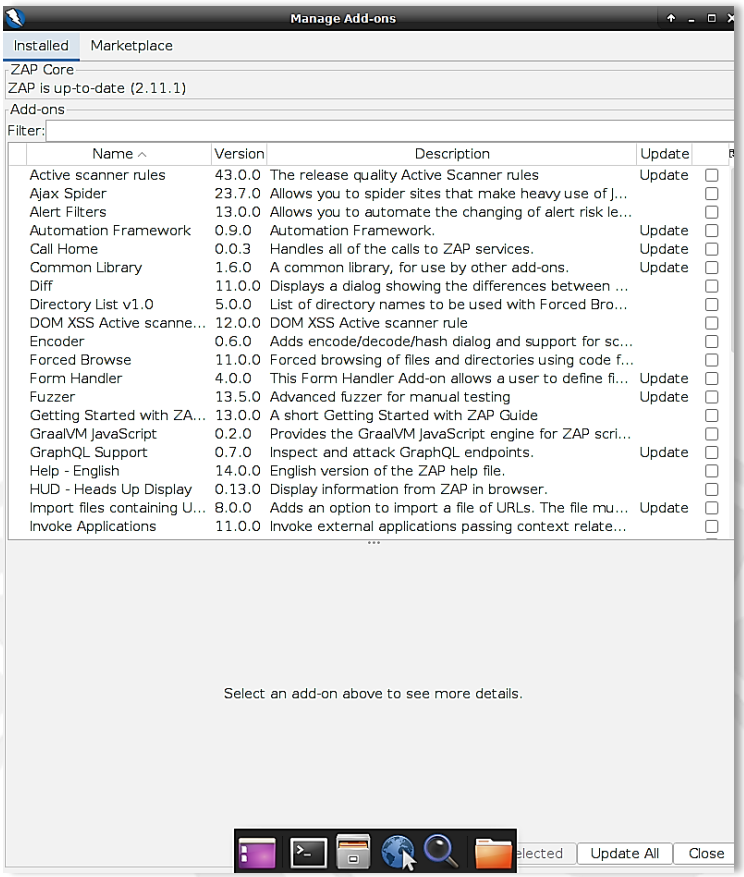
Introduction

OWASP ZAP (https://www.zaproxy.org/getting-started/) is a free tool typically used for *pentesting* web applications. However, **it is also a DAST tool** depending on how it is used.

> 🔍 ***Remember that, as we saw in the theory, DASTs are black box tools that "attack" applications from outside them, introducing possible errors to detect vulnerabilities. ZAP itself is a proxy that sits between the browser and the application being examined, intercepting requests between the two, allowing them to be modified, and then sending them to the web.***

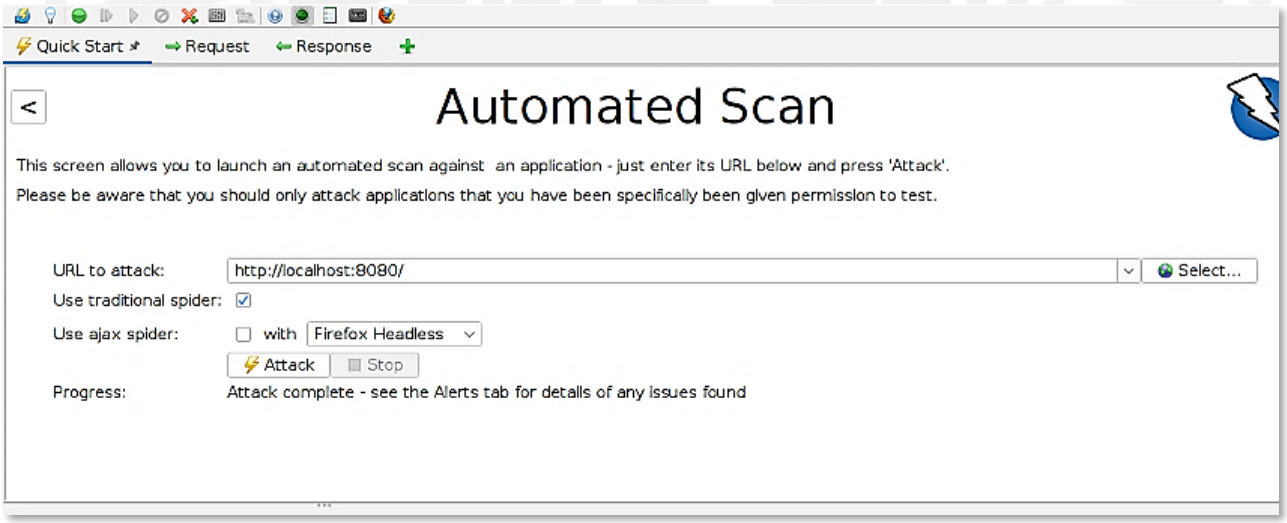
The truth is that in order to test this DAST **we will need a "victim" application**, and we can use any from other courses that we have available and of which **we are the owners or have obtained a written authorization**. If this is not the case, a good way to test it is by installing and configuring the ***Damn Vulnerable Web App*** **(DVWA)** as follows: https://hub.docker.com/r/vulnerables/web-dvwa/. We will need *Docker* for this: `docker run --rm -it -p 8080:80 vulnerable/web-dvwa` (although **it is better to put another port**, because the `8080` clashes with ZAP).

Once this is done, we can install the tool according to the OS used to do the tests by following these indications: https://software.opensuse.org/download.html?project=home%3Acabelo&package=owasp-zap. **It is recommended to do this from the operating system's repositories**. Once this is done, we run the tool with `owasp-zap` and let all the *add-ons* that come standard be updated to achieve the best possible performance (*Update All*).



Active scan

The most direct way to employ this DAST tool is through its *Quick Start-Automated Scan option*. We simply give it the URL of the website to be analyzed and after a while **it will give us a list of the security problems that it has found in it,** which we can analyze by clicking on each of them to know their details, severity, etc. **We will also see that it has *spidered* the application**, that is, that it will show us a tree of pages that make up the examined site.

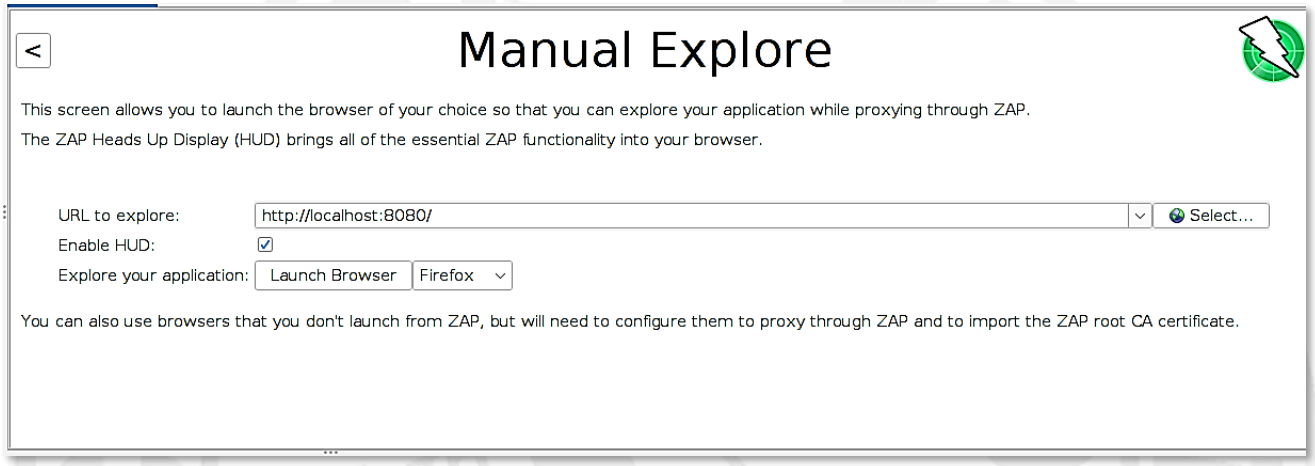


If everything has gone well, we will recognize some of the problems we have described in the theory, which we would have to find a solution to before putting the application into production.

Manual scanning

While active scanning is a very simple and straightforward way to use the DAST tool, it has a problem that you have probably already noticed: **if you have a *login* screen for a private area, the tool will not be able to browse that private area because it doesn't know the credentials**. To solve this, one of the simplest options is **to do a manual scan**: We browse the application in a browser that ZAP shows us, we authenticate normally (**we are in a security test scenario**, we are supposed to have credentials) and the proxy will do security tests (and spidering) as we browse the application normally.

This way we can browse to all the parts that interest us and access pages that for some reason the tool's spider cannot reach

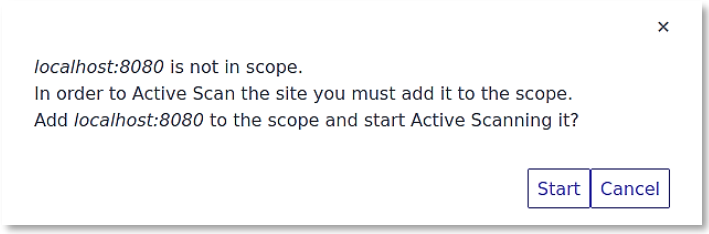


There is one thing we must keep in mind about the browser that this tool shows: it is the default browser of the system with a plugin called **HUD,** which shows us **on the left side the vulnerabilities of the web page** we are visiting at this moment (with their severity with a typical color code in flags) and on **the right those accumulated by all the websites visited on the site so far**.

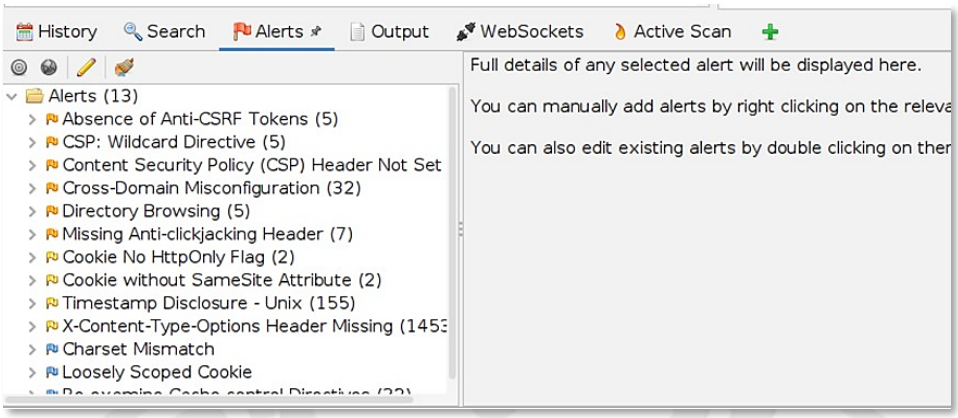
As we will see, ZAP continues to do its job "underneath", just as it did with active scanning.

While browsing the web we now see additional controls that tell us the status of the tool's findings on the web. To fully understand what is going on while browsing the testing site, we need to consider the following:

- What you find is classified by gravity on the superimposed button panel on the left side of the website.
- On the right side of the website**, you can activate the previous active scan at any time**, a sign that you can launch it once you have passed the *login* screen
- Depending on what you have done, the tool will give you this warning, which you simply accept and click "*Start*" to do the scan. You can see its progress on the HUD icons themselves.



- Vulnerabilities found from manual or hybrid scanning will appear again in the ZAP interface.



- **This working mode allows you to use a special option, the "*Attack Mode*",** represented by the sight of a weapon and activated by clicking on it. The *Attack Mode* works as you navigate through the pages, and you can do an *active scan* in parallel. This is a special way of working ZAP **where it will make intrusive tests to all the pages** that it can reach from which we are browsing, therefore increasing the options of finding more problems, but also the risk of "breaking" something in the application by entering corrupted data, *fuzzing*, etc. For that reason, before using this "*Attack Mode*" **it is strongly recommended to do two things**:
  1. **Disconnect the network from the test machine just in case, if possible**. It is very common for many pages to use third-party services for something (e.g., advertising, *trackers*...) and could be attacked by accident if we are not careful. Of course, **it could only be done if the page is able to function this way**. In any case, **never browse to the website of a third-party service in this mode, because you could commit a crime**.
  2. **Test a web clone** to avoid unwanted interactions and data corruption from ZAP's actions.
- Finally, do not forget that ZAP has the option to **obtain reports** where all the vulnerabilities found, their severity, and other information are summarized, ideal to pass on to the development team so that they can proceed to solve them.

✍ Badges & Self-Assessment

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

Badge Level	Unlocked when	Unlocked?
	You understand what a reverse proxy is. You can answer this question: Does *a reverse proxy really serve websites by itself?*	
	Do you understand what the main functionality of a WAF is	
	You know how to start web scans using `nikto`	
	You can check if *ModSecurity* also provides protection against some `nmap` scans	
	You understand the purpose of a SAST tool and the kind of results it produces	
	You understand the purpose of an SCA tool and the kind of results it produces	
	You can answer the questions: *Does a reverse proxy completely isolate a service infrastructure from its potential customers? What is the point of using two totally different network ranges?*	
	You can answer these questions: *Is the custom rule we use testing the ModSecurity engine itself or the CRS rules we download? Is this enough to prove that the entire infrastructure is working?*	
	You can answer these questions: *Is the fake RCE attack we did testing the ModSecurity engine itself or the CRS rules we downloaded? Is this enough to prove that the entire infrastructure is working?*	
	You can answer these questions: *Are ModSecurity blocked requests Logged? If so, 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 attempts (XSS, SQL injection...)? Do you identify the type of attack attempt? So, can a potential GUI that reads WAF logs show its users the types of attacks that are occurring?*	
	You understand how a NIDS and a *proxy forward* can be combined to detect compromised machines on an internal network	
	You can analyze the source code of an application using *SonarQube*. You can answer these questions: *Do you think these types of tools locate all possible errors? If not, do you think it is worth using them?*	

	You can successfully configure a *ModSecurity2* WAF that works to protect a server infrastructure	
	Bearing in mind that a WAF identifies an attack only if the syntax of the language in which it is encoded is correct (*XSS, SQL Injection*...), reason about what could be the main technique that can be used to bypass the protection of a WAF	
	You can answer these questions: *Why is a reverse proxy important from a security standpoint? Do you think this concept is like encapsulation in object-oriented programming?*	
	You can answer this question: *Do you think WAFs provide maximum protection against attacks, or should they be treated as an extra layer of protection?*	
	You know how to combine techniques we have seen in previous labs to further improve a proxy's security in more scenarios.	
	You know how to use a DAST tool like ZAP to check for vulnerabilities in a running web application and scan its pages for vulnerabilities in an automated or manual manner	
	You understand that an application delivered with a correct SCA, SAST, and DAST report always offers more guarantees than the lack of results of tools of this kind	
	You have a clear understanding of the type of vulnerabilities that can be prevented by a WAF *like ModSecurity* versus those prevented by a static analysis tool like *SonarQube*	
	**Hold the door!** You can protect a web server using a WAF-secured reverse proxy with intrusion detection capabilities, and locate vulnerabilities in the code or its dependencies by implementing automated SCA and SAST tools	