VICEPRESIDENCIA TERCERA DEL GOBIERNO
MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL

GOBIERNO DE ESPAÑA

SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

Webinar 5 "Use of OWASP ZAP"

Exercises

017 TU AYUDA EN CIBERSEGURIDAD

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD

INDEX

1. Practical Exercise ... 3
2. Research Exercise ... 7
2.1. What type of vulnerability has been identified? ... 11
2.2. What kind of implications does it have? ... 11
2.3. How would you address the vulnerability? ... 12
3. Additional exercise ... 13

FIGURE INDEX

Figure 1 – Proxy settings ... 3
Figure 2 – Web browsing of the local "test" host ... 4
Figure 3 – HTTP Headers ... 4
Figure 4 – Request log ... 5
Figure 5 – POST Request form ... 5
Figure 6 – Warning about lacks of CSRF token ... 6
Figure 7 – bWAPP deployment ... 7
Figure 8 – bWAPP login ... 7
Figure 9 – Challenge: OS Command Injection (I) ... 8
Figure 10 – Challenge: OS Command Injection (II) ... 8
Figure 11 – POST Request ... 9
Figure 12 – nslookup output seems similar to server config ... 9
Figure 13 – Breakpoint ... 10
Figure 14 – Modification of the target parameter ... 10
Figure 15 – Confirmation of injection ... 10
Figure 16 – Source code (commandi.php) ... 11
Figure 17 – Bind shell with netcat ... 12
Figure 18 – Downloading dictionaries ... 13
Figure 19 – Adding new dictionaries ... 13
Figure 20 – Adding new dictionaries ... 14
Figure 21 – Identification of the configuration file "config.inc” ... 14

1. PRACTICAL EXERCISE

Learning HTTP communications in the bWAPP application using a passive approach from a Kali Linux distribution.

The objective of this exercise is to familiarize the student with the configuration of OWASP ZAP for HTTP traffic analysis and to study the communications used with a local domain. The host "test" will serve as a support; we will study the type of parameters sent and received, the method used, the location of web forms, etc. The student will use a passive approach, without using any of the attack features available in OWASP ZAP.

Let's configure the browser proxy to use the localhost address on port 8080. Be sure to select the "Use this proxy server for all protocols" option to include possible SSL traffic in that setting.

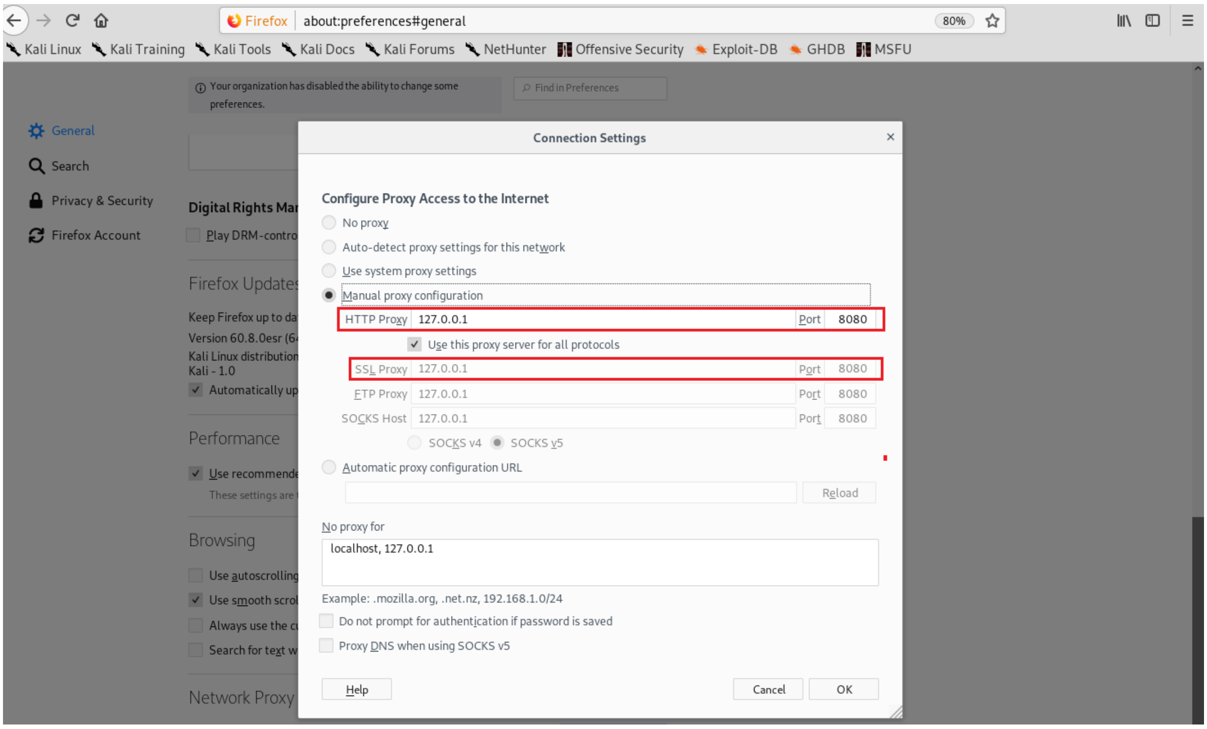


Figure 1 – Proxy settings

After setting the proxy, you will navigate to the http://test/app/bWAPP/login.php host and verify that OWASP ZAP collects all requests.

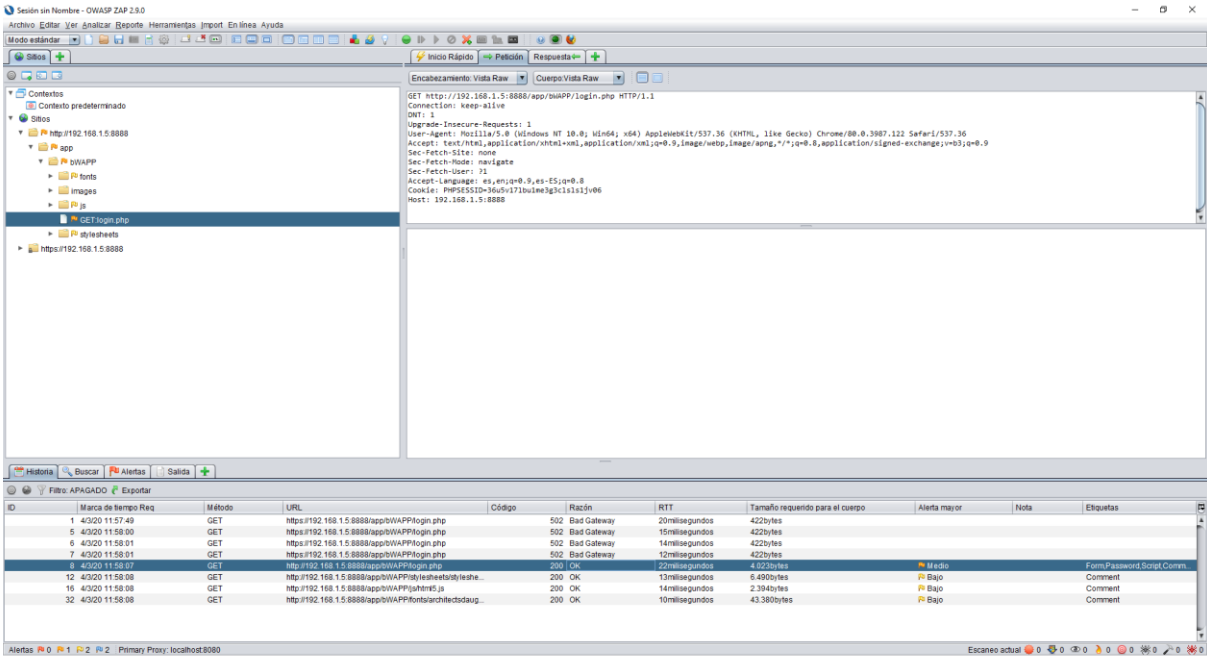


Figure 2 – Web browsing of the local "test" host

From now on we can study the type of traffic exchanged between the browser and the local host, as well as the technologies used by the same.

For example, we can immediately observe some of the HTTP security headers used by the web server.

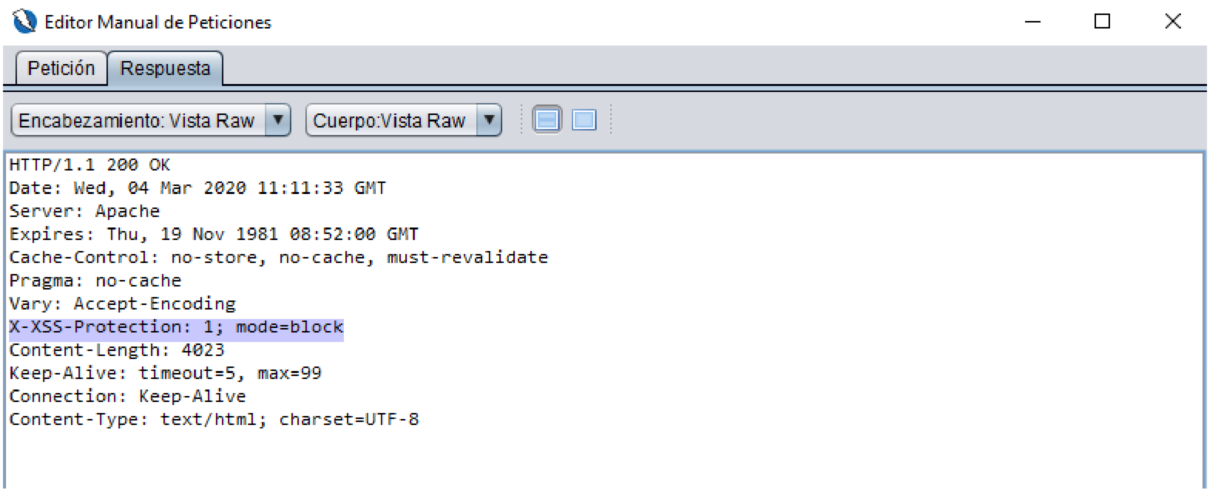
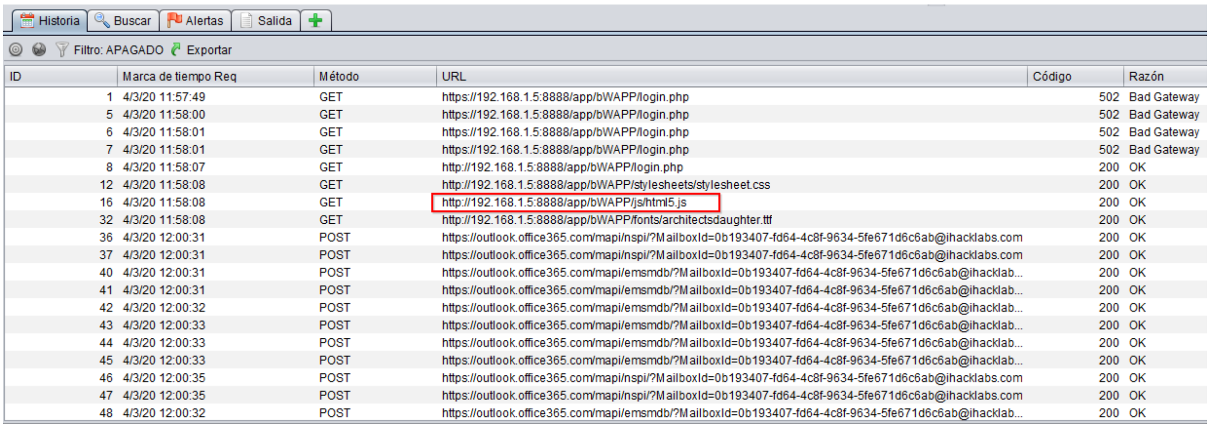


Figure 3 – HTTP Headers

Some of these headers (https://owasp.org/www-project-secure-headers/), such as the use of *X-XSS-Protection*, would make XSS-type attacks difficult by activating certain filters in the browser. Similarly, the *X-Content-Type-Options* header would make it possible to prevent certain types of attacks as a result of "content sniffing" carried out by the browser (http://webblaze.cs.berkeley.edu/papers/barth-caballero-song.pdf). It is recommended that the student researches and understands the use of these headers for a better understanding of concepts related to web security. In this aspect, the OWASP (Open Web Application Security Project) is one of the best starting points for understanding the basic pillars of web security.

Continuing with the analysis of the host "test", in the information window (History tab) you can see, the type of resources requested when accessing the main URL: CSS, JavaScript, etc.



Historia | Buscar | Alertas | Salida
Filtro: APAGADO | Exportar

ID	Marca de tiempo Req	Método	URL	Código	Razón
1	4/3/20 11:57:49	GET	https://192.168.1.5:8888/app/bWAPP/login.php	502	Bad Gateway
5	4/3/20 11:58:00	GET	https://192.168.1.5:8888/app/bWAPP/login.php	502	Bad Gateway
6	4/3/20 11:58:01	GET	https://192.168.1.5:8888/app/bWAPP/login.php	502	Bad Gateway
7	4/3/20 11:58:01	GET	https://192.168.1.5:8888/app/bWAPP/login.php	502	Bad Gateway
8	4/3/20 11:58:07	GET	http://192.168.1.5:8888/app/bWAPP/login.php	200	OK
12	4/3/20 11:58:08	GET	http://192.168.1.5:8888/app/bWAPP/stylesheets/stylesheet.css	200	OK
16	4/3/20 11:58:08	GET	http://192.168.1.5:8888/app/bWAPP/js/html5.js	200	OK
32	4/3/20 11:58:08	GET	http://192.168.1.5:8888/app/bWAPP/fonts/architectsdaughter.ttf	200	OK
36	4/3/20 12:00:31	POST	https://outlook.office365.com/mapi/nspi/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklabs.com	200	OK
37	4/3/20 12:00:31	POST	https://outlook.office365.com/mapi/nspi/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklabs.com	200	OK
40	4/3/20 12:00:31	POST	https://outlook.office365.com/mapi/emsmdb/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklab...	200	OK
41	4/3/20 12:00:31	POST	https://outlook.office365.com/mapi/emsmdb/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklab...	200	OK
42	4/3/20 12:00:32	POST	https://outlook.office365.com/mapi/emsmdb/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklab...	200	OK
43	4/3/20 12:00:33	POST	https://outlook.office365.com/mapi/emsmdb/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklab...	200	OK
44	4/3/20 12:00:33	POST	https://outlook.office365.com/mapi/emsmdb/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklab...	200	OK
45	4/3/20 12:00:33	POST	https://outlook.office365.com/mapi/emsmdb/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklab...	200	OK
46	4/3/20 12:00:35	POST	https://outlook.office365.com/mapi/nspi/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklabs.com	200	OK
47	4/3/20 12:00:35	POST	https://outlook.office365.com/mapi/nspi/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklabs.com	200	OK
48	4/3/20 12:00:32	POST	https://outlook.office365.com/mapi/emsmdb/?MailboxId=0b193407-fd64-4c8f-9634-5fe671d6c6ab@ihacklab...	200	OK

Figure 4 – Request log

Although most of the resources are requested via the web, we can also filter by POST methods to locate entry parameters of interest. For example, the following image shows the information submitted when using one of the web search engines. If we had the corresponding authorization to do a security audit, it would be interesting to test several payloads with these parameters (by means of the fuzzing functionality) to corroborate if they are susceptible to any vulnerability.

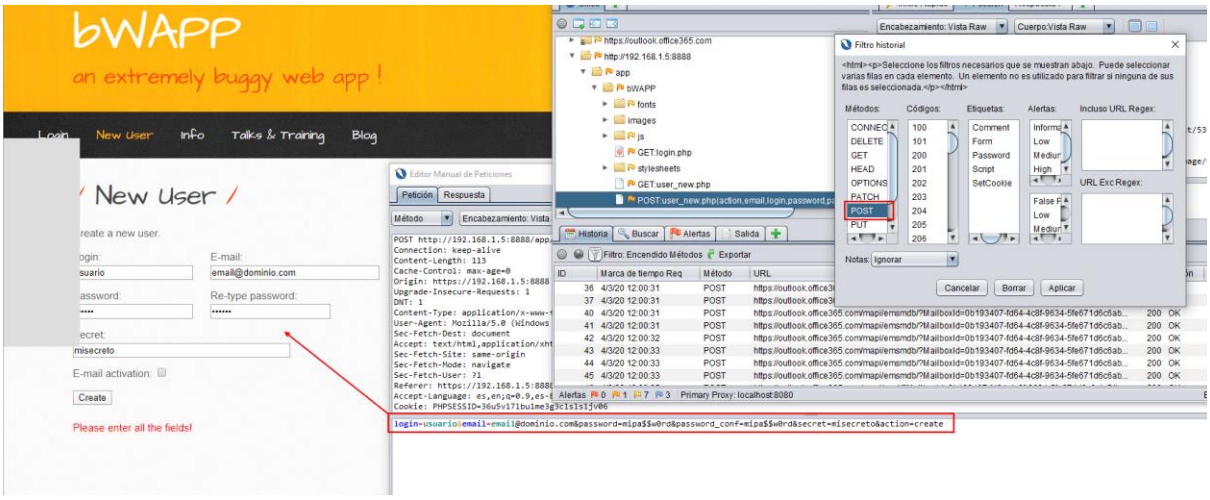


Figure 5 – POST Request form

As we browse, we will frequently check the alert window to see if ZAP has detected any security issues. In the following image, for example, it informs us that a form without a CSRF token has been detected (https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html). In this case, however, the alert does not represent any danger since the form is not related to the execution of potentially harmful actions or the sending of critical information.

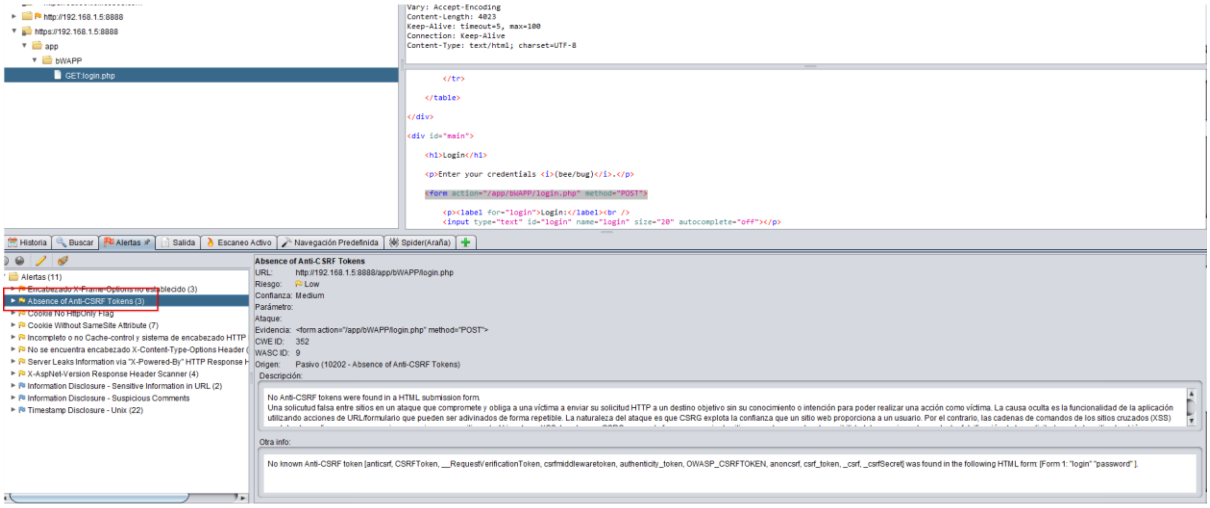


Figure 6 – Warning about lacks of CSRF token

By carefully studying the requests intercepted by ZAP we will be able to reconstruct the technologies used by the service. We will also be able to identify the entries that are likely to be vulnerable. Remember, however, that active scanning or using any of the attack features available in ZAP requires the appropriate authorization, otherwise a crime could be committed.

2. RESEARCH EXERCISE

The objective of the exercise is to investigate the HTTP traffic used by a certain web portal and try to take advantage, if possible, of a vulnerability in one of its parameters. Because this type of testing requires prior authorization from the domain owner, a web service will be installed on a Kali instead, and the testing will be performed locally.

To proceed with the installation, download the attached bWAPP.zip file and unzip its contents into the /var/www/ directory. Then follow the instructions described in the INSTALL.txt file.

```
root@kali:/var/www/html/bWAPP# ls -l
total 19652
drwxrwxrwx  2 root root       4096 ene 24 15:12 apache2
drwxrwxrwx 13 root root      12288 ene 24 15:12 bWAPP
-rwxrwxrwx  1 root root    5010042 nov  2  2014 bWAPP_intro.pdf
-rwxrwxrwx  1 root vboxsf 15058349 ene 24 15:11 bWAPP_latest.zip
-rwxrwxrwx  1 root root        325 mar  8  2014 ClientAccessPolicy.xml
-rwxrwxrwx  1 root root        200 mar 11  2014 crossdomain.xml
drwxrwxrwx  2 root root       4096 ene 24 15:12 evil
-rwxrwxrwx  1 root root       2589 may 12  2014 INSTALL.txt
-rwxrwxrwx  1 root root       2491 nov  2  2014 README.txt
-rwxrwxrwx  1 root root       8271 nov  2  2014 release_notes.txt
root@kali:/var/www/html/bWAPP# less INSTALL.txt
```

Figure 7 – bWAPP deployment

After installation verify that you have access to the web platform from your browser and that it is correctly configured to use ZAP as a web proxy.

Figure 8 – bWAPP login

Then authenticate yourself to the portal using your default credentials (if these have not been changed):

- **Login**: bee
- **Password**: bug

Once authenticated, choose the type of bug¸ "OS Command Injection" and press the Hack button.

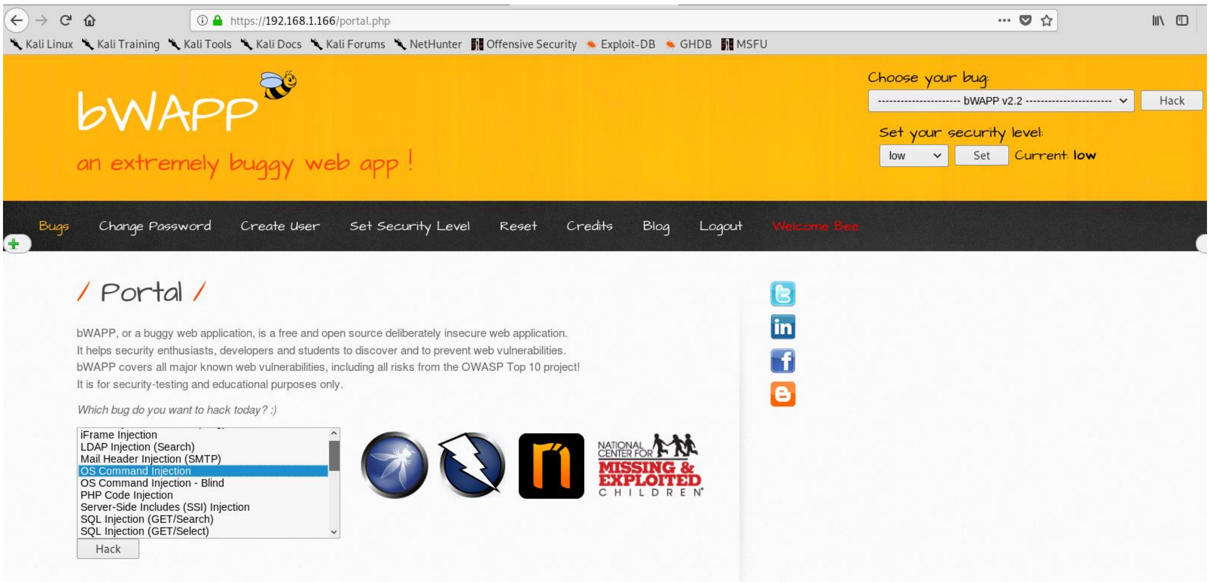


Figure 9 – Challenge: OS Command Injection (I)

The user will find the following web portal where he will have to audit, using ZAP, if any of the parameters used are vulnerable. He will also have to describe the type of vulnerability found, its implications and how it would be solved. Students will not be able to use active scanning or any of the attack functions implemented in ZAP (this includes: Spider, fuzzing, predefined navigation, etc.). Only a manual approach can be used to improve their skills in using OWASP ZAP. After correctly identifying the vulnerability, students will be able to study the source code of the vulnerable script.

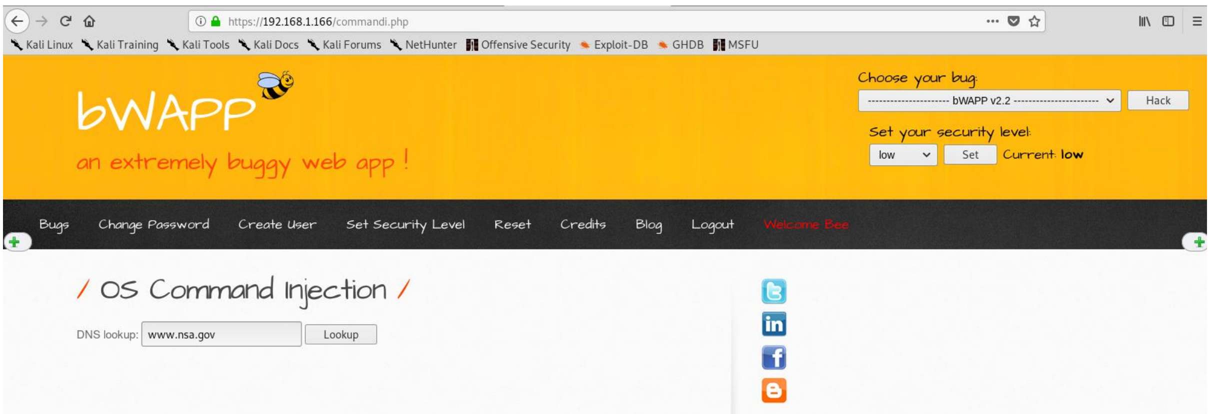


Figure 10 – Challenge: OS Command Injection (II)

Exercise solution:

If you analyze the sent request when you press the "Lookup" button, you can see that the input field (by default, the value www.nsa.gov) is sent via POST with the parameter target.

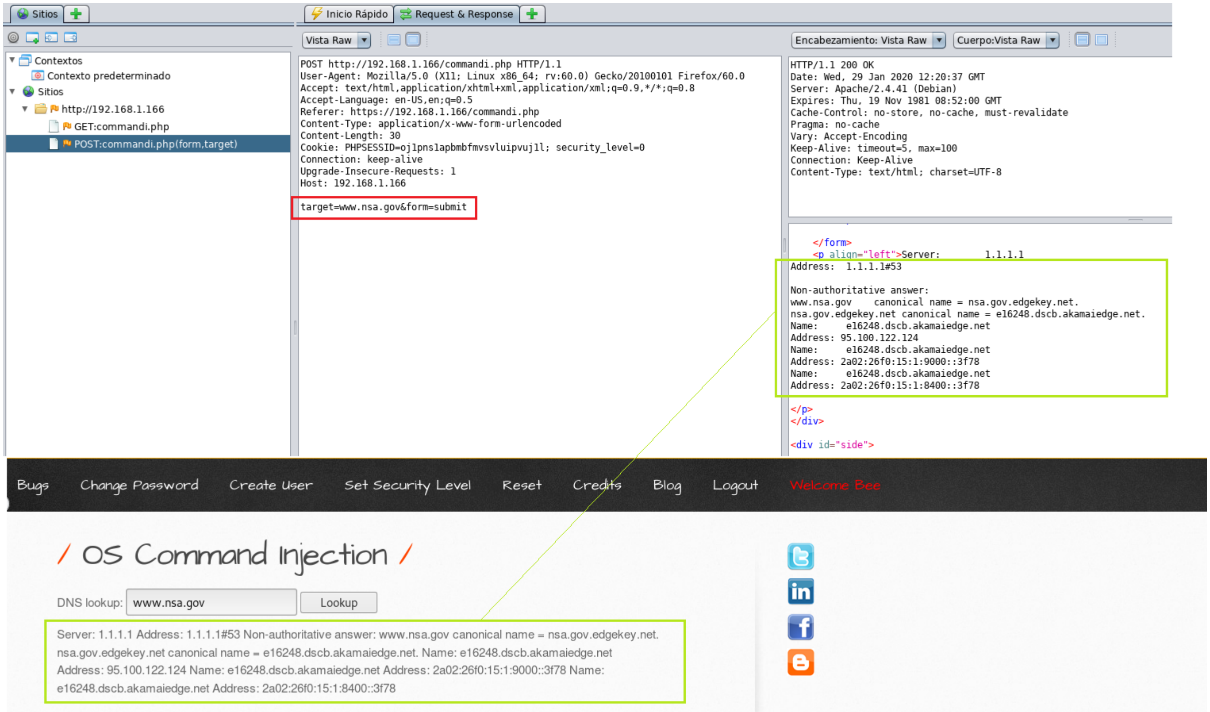


Figure 11 – POST Request

Note that the result returned by the server matches the output generated by the **nslookup** command**:**

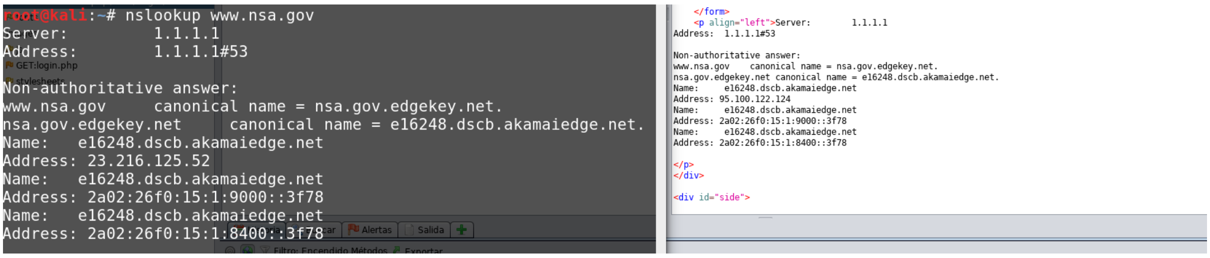


Figure 12 – nslookup output seems similar to server config

Everything seems to indicate that the web server is using the value sent by the target parameter to run it in the console and return the generated output. What would happen if instead of sending the domain we add a second command in the input? For example, the command: ;ls

To modify this argument on the fly, we will use the break points described in the course, with which we can "stop" the web request before it is sent to the web service.

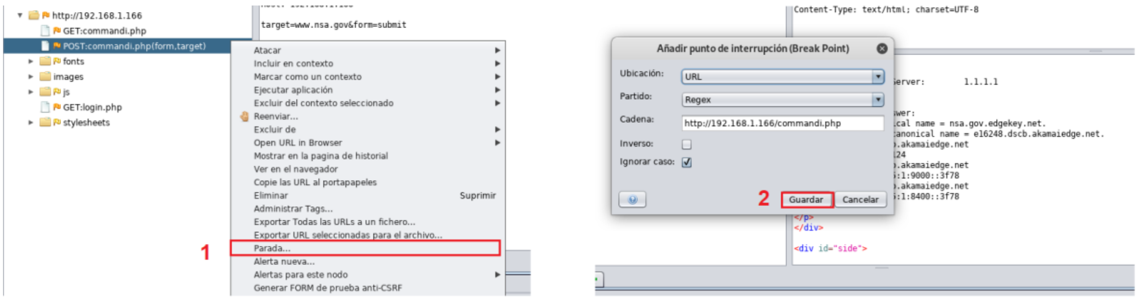


Figure 13 – Breakpoint

After creating the breakpoint we'll click on the "Lookup" button again and replace the content of the "target" parameter with the following string:

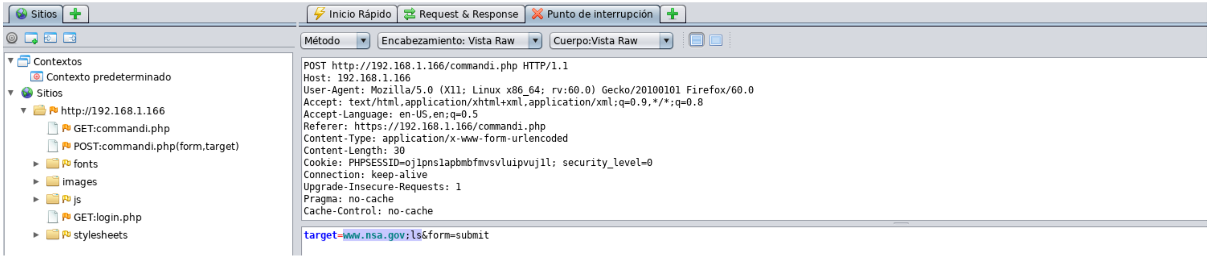


Figure 14 – Modification of the target parameter

Note that this would execute the command: nslookup www.nsa.gov;ls (in Linux the ';' character allows concatenation of several commands). After forwarding the request to the destination, we can see that we have indeed managed to recover the list of files on the server, which would confirm the type of vulnerability.

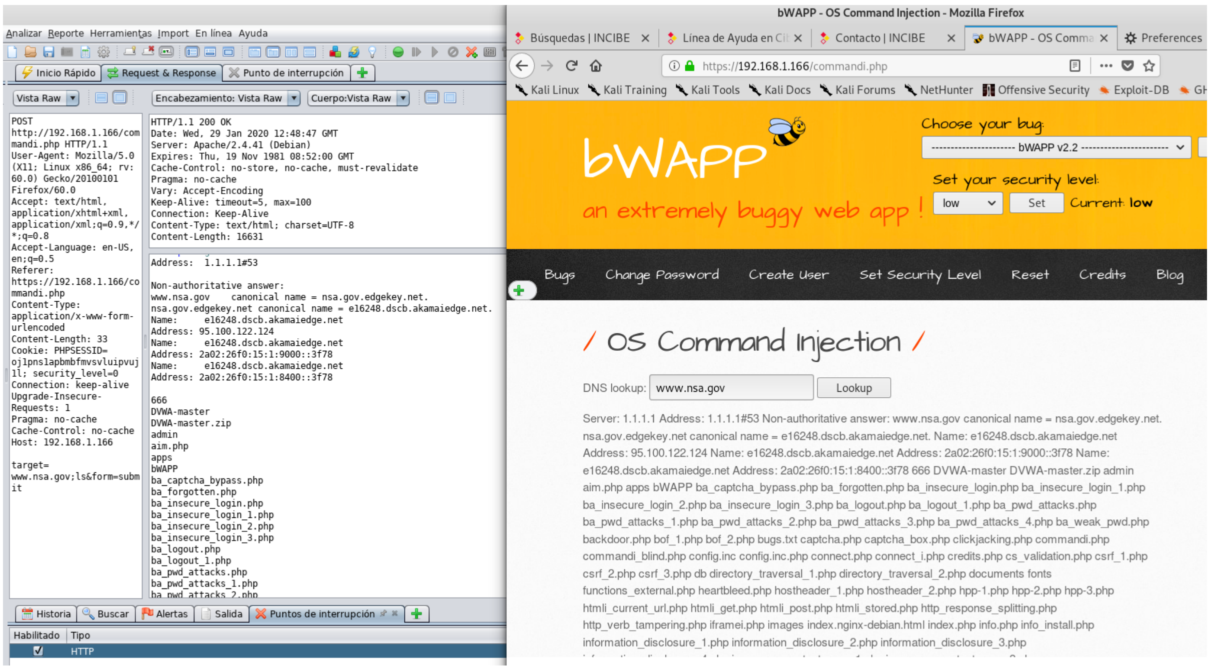


Figure 15 – Confirmation of injection

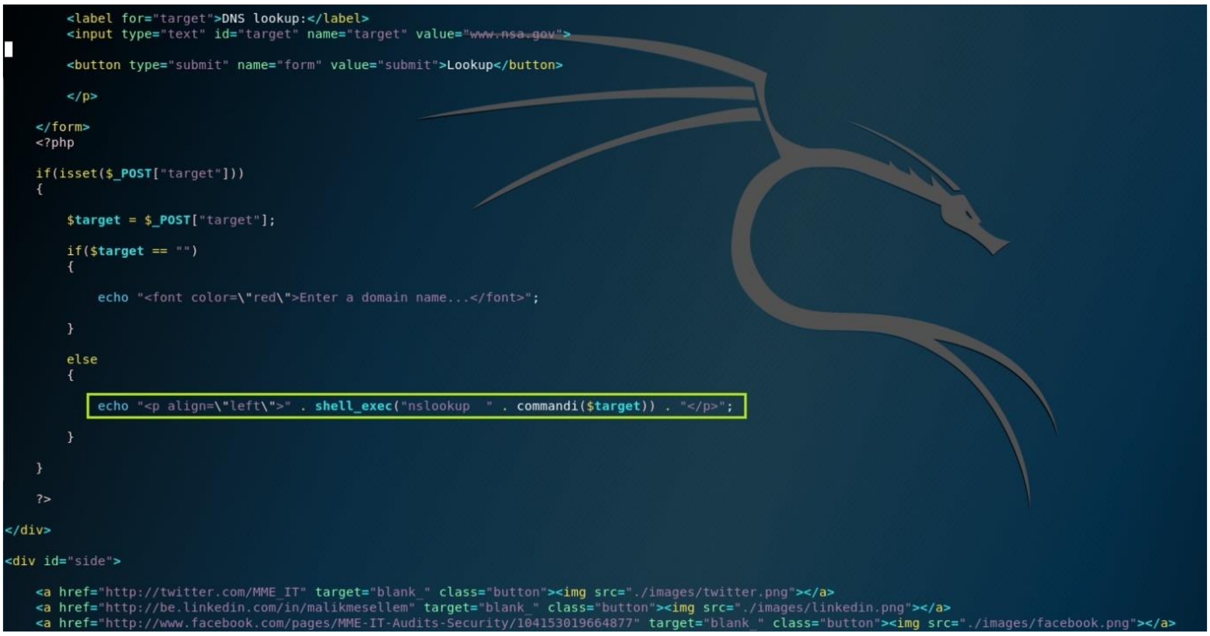
With this information we can answer the following questions:

2.1. What type of vulnerability has been identified?

The vulnerability corresponds to a command injection: as described by OWASP (https://owasp.org/www-community/attacks/Command_Injection), this type of attack:

"is possible when an application passes insecure data provided by the user (forms, cookies, HTTP headers, etc.) to a system shell. In this attack, the operating system commands provided by the attacker are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation".

If we access the web directory and open the script *commandi.php* we can corroborate that the script is invoking the *nslookup* command through the insecure function *shell_exec* (https://www.php.net/manual/es/function.shell-exec.php) without applying any kind of validation or filter to the target parameter.



```
        <label for="target">DNS lookup:</label>
        <input type="text" id="target" name="target" value="www.nsa.gov">

        <button type="submit" name="form" value="submit">Lookup</button>

        </p>

    </form>
    <?php

    if(isset($_POST["target"]))
    {

        $target = $_POST["target"];

        if($target == "")
        {

            echo "<font color=\"red\">Enter a domain name...</font>";

        }

        else
        {

            echo "<p align=\"left\">" . shell_exec("nslookup  " . commandi($target)) . "</p>";

        }

    }

    ?>

</div>

<div id="side">

    <a href="http://twitter.com/MME_IT" target="blank_" class="button"><img src="./images/twitter.png"></a>
    <a href="http://be.linkedin.com/in/malikmesellem" target="blank_" class="button"><img src="./images/linkedin.png"></a>
    <a href="http://www.facebook.com/pages/MME-IT-Audits-Security/104153019664877" target="blank_" class="button"><img src="./images/facebook.png"></a>
```

Figure 16 – Source code (commandi.php)

2.2. What kind of implications does it have?

An attacker could execute all sorts of commands and thus completely compromise the web server. For example, if instead of executing an "ls" the attacker had executed "*nc -l -p 2222 -c /bin/bash*" he would install a bind shell as a backdoor.

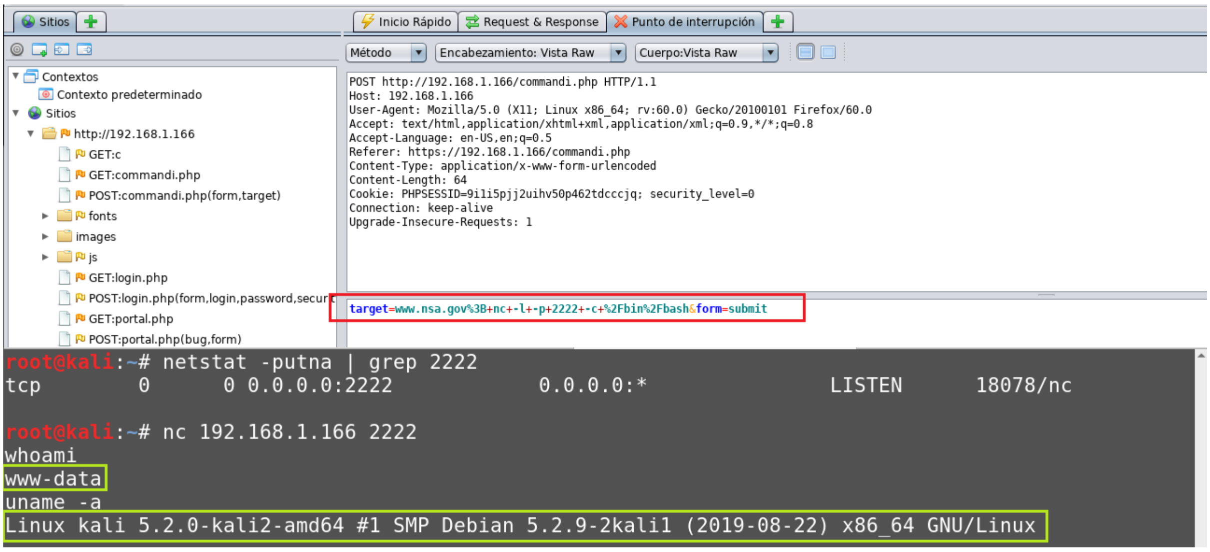


Figure 17 – Bind shell with netcat

2.3. How would you address the vulnerability?

- We recommend using the guidelines described by the OWAS project (https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html).

These countermeasures are summarized in:

- Avoid directly invoking commands.
- Escape and filter the values provided to the commands.
- Parameterization along with proper validation of input parameters.

3. ADDITIONAL EXERCISE

Taking advantage of the installation of the local web service used in the previous research section, in this exercise users will learn how to add new dictionaries to ZAP for the discovery of new web resources.

A repository of special interest for this exercise is the project at Github Fuzzdb since it gathers many dictionaries within the *predictable-filepaths* directory (https://github.com/fuzzdb-project/fuzzdb/tree/master/discovery/predictable-filepaths) for many web technologies; for example, for CMS (Drupal, Joomla, WordPress, etc.), login files commonly used for different platforms, etc.

If we want to make use of these dictionaries we can either download them manually from Github or clone the whole repository locally.

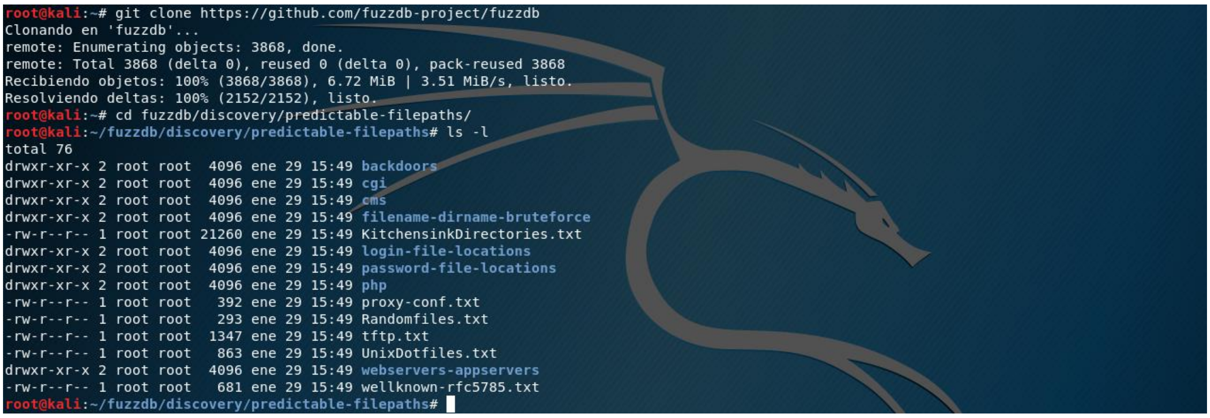


Figure 18 – Downloading dictionaries

Later, if we want to add some of these dictionaries to ZAP, to be able to be used with the functionality "Predefined navigation", we will go to the menu "Tools -> Options" and later we will select the dictionary that we want (in the following image the raft-large-directories.txt dictionary has been selected). Note that from this menu we can also configure if we want to include file navigation, the extensions we want to include and other performance related parameters, for example, the number of threads to use. If we want to integrate more dictionaries we will repeat the same process.

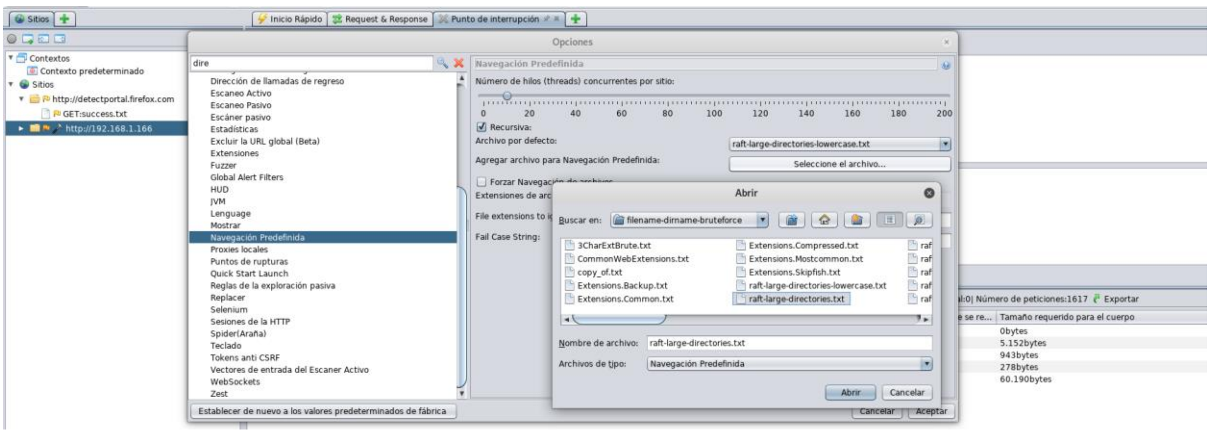


Figure 19 – Adding new dictionaries

Once the dictionaries are added, we will select the resource from which we want to discover new directories and, by right clicking, we will select the option "Defined navigation directory" within the Attack menu. Notice that in the lower window the previously added dictionaries will appear and we will be able to select any of them.

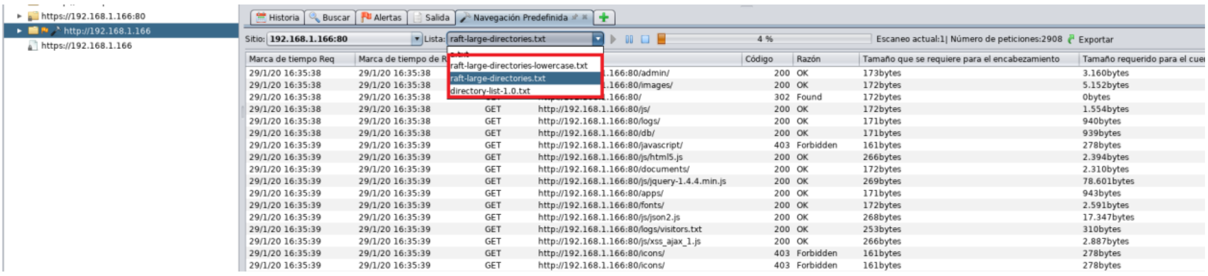


Figure 20 – Adding new dictionaries

As detailed in the webinar, the discovery of directories and files through the "Predefined Navigation" functionality is very useful for identifying unreferenced resources. Sometimes, these resources allow us to access directories that, by mistake or carelessness, have been made public and that offer information about the platform, technologies used or any other type of sensitive data about the configuration of the web service. The following image shows one of the configuration files identified thanks to one of the dictionaries; the file "config.inc" located in the root directory. Note that it includes access credentials to a certain database.

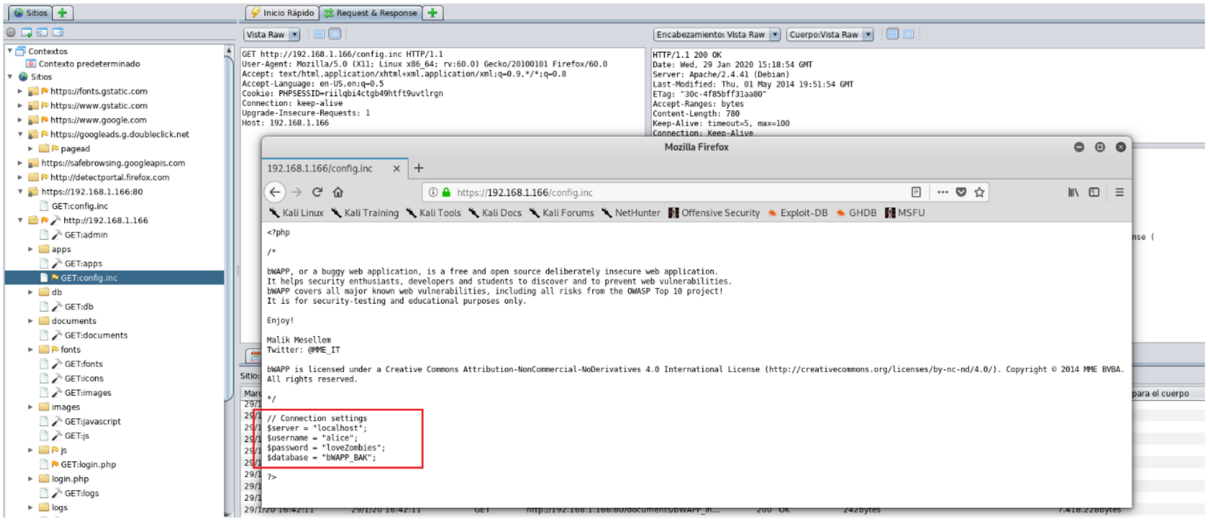


Figure 21 – Identification of the configuration file "config.inc"