



**UNIVERSIDAD DE CASTILLA - LA MANCHA**  
**ESCUELA DE INGENIERÍA INDUSTRIAL DE**  
**TOLEDO**

**TRABAJO FIN DE GRADO Nº 17- B-225017**

*WEB HAZARD & INFRASTRUCTURE SCANNING THROUGH  
AUTOMATED PENTESTING AND REPORTING*



Autor:

SAMUEL VÁZQUEZ MORÓN

Director:

FRANCISCO MOYA FERNÁNDEZ

JULIO de 2017



# Contents

---

Abstract .....	1
1 Introduction .....	3
1.1 Summary .....	3
1.2 Motivation.....	3
1.3 Objectives.....	4
2 Antecedents .....	5
2.1 Background.....	5
2.2 Web Application Attacks .....	7
2.3 Attack Prevention.....	12
2.4 Manual Testing.....	14
3 Implementation.....	21
3.1 Information Gathering .....	21
3.2 Automated Testing.....	23
3.3 Report Generation .....	27
4 Integration.....	29
4.1 Device Integration .....	29
4.2 Dependences.....	29
4.3 Remote Control .....	31
4.4 Graphical User Interface .....	34
5 Conclusions .....	37
5.1 Recommendations .....	37
6 Appendix .....	39
6.1 Introducción (Español) .....	39
6.2 Conclusiones (Español).....	40
6.3 Sample Scan Report .....	42
7 Bibliography .....	53

# Figure Index

---

1 Global Web Application Attack Source Countries, Q1 2017 .....	6
2 Top 10 Target Countries for Web Application Attacks, Q1 2017 .....	7
3 Web Application Attack Frequency, Q1 2017 .....	8
4 Standard Login Form .....	15
5 Bypassing a Login Form .....	15
6 Successful Login Form Bypass.....	15
7 User Lookup Bypass.....	16
8 Cookie Manipulation .....	17
9 Login Error Example .....	17
10 Reflected XSS Example .....	18
11 Successful Reflected XSS.....	18
12 Stored XSS Example.....	19
13 Successful Stored XSS .....	19
14 CSRF Example .....	20
15 API key configuration in OWASP ZAP .....	31
16 Configuration to enable VNC in Raspberry Pi .....	32
17 Locating the Raspberry Pi's IP address via "ip addr" .....	32
18 Security configuration for the VNC server .....	33
19 Configuration of the dhcpd.conf file .....	34
20 GUI main screen .....	35
21 Sample scan shown on the GUI.....	35

# Abstract

The aim of this project was creating a web application scanner that would, in an automated fashion and with limited user input, apply a series of attacks or payloads on a web application to find how vulnerable it is, and then generate a report to inform the user about what payloads were successful, and thus what attacks the web application is vulnerable to.

Computer security is an extensive field with multiple branches. The reason this project focuses on web applications in particular is because web applications are constantly facing the Internet, which makes them extremely vulnerable and relatively easy to attack compared to other types of targets. Adding to this, it's common for web applications to store sensitive data or otherwise important information, which makes them attractive targets for potential cyber-attackers. For these reasons, this project has focused exclusively in web application vulnerabilities, without disregarding the importance of other aspects of computer security.

Penetration testing, or pen-testing, is the practice of attacking a computer system the way an actual attacker would in order to find vulnerabilities, always with prior authorization from the system administrator. While it is an effective method to find security breaches, it is also expensive. And while statistics show that economic losses caused by security breaches far exceed the costs of pen-testing, the percentage of enterprises that regularly perform pen-tests is relatively small, and the percentage that bothers with web application testing is smaller still. This might denote an insufficient awareness about the risks associated with insecure web applications.

Thus, the main objective of this project will be to highlight the importance of web application security and computer security in general. This will be achieved through the development of an inexpensive and easy to use web application scanner. The argument is that, if this cheap and relatively simple tool can find vulnerabilities in a web, then an actual human attacker should also be able to abuse those vulnerabilities, or find other vulnerabilities that the scanner has overlooked or is unable to find.

The scanner can then be seen as a "marketing tool" of sorts. It's not an extensive and far-reaching scan capable of detecting and solving all issues, but rather a simple and quick scan to inform the system administrator that the system is vulnerable at all, and how easy would it be to jeopardize it. A salesperson could make use of this scanner as an argument to convince a company to enlist the services of a professional pen-tester.

The scanner is written entirely in Python. The Python programming language was chosen due to its accessibility and emphasis on readability. A variety of third-party Python modules are used to increase the capabilities of the standard Python library.

The code will make use of Nmap and OWASP ZAP, free third-party tools with widespread use in the world of computer security. Nmap is a network mapper that will be used for network discovery in this project. OWASP ZAP is a free security tool designed for beginners and professionals alike, and it includes many useful tools such as a spider bot and automated

vulnerability scanning, both of which will be used for this project. Both Nmap and OWASP ZAP have Python modules that allow them to easily interact with Python.

The scanning process is divided into three distinct but consecutive stages. First, the network is scanned and analyzed in order to determine the location and identification of the web applications to be scanned. The second stage is the vulnerability scan, which will dig through directories and then apply the attacks or payloads. The third and last stage will recover all information gathered during the previous step and send it to a pdf file.

The network scan will begin by obtaining the CIDR of the network the device containing the code is connected to. CIDR is a human-readable notation that holds information pertaining to the IP address and netmask of a given network (enough to fully characterize it and differentiate it from other networks). Next, the code will scan the network for any live hosts, and finally scan those hosts for any ports that might contain web applications, using Nmap.

The second stage, the vulnerability scan, will also be divided in sequential steps. The previous network scan will be the starting point for this scan. However, the network scan only returns the root or parent directory for the web application to be scanned. Before the scan can continue, it becomes necessary to find a way to browse through the application directories to fetch all children pages. A crawler or spider is a bot designed for this very purpose, and one such bot is included within OWASP ZAP.

While this spider bot is browsing through directories, it performs a passive scan in parallel. This passive scan acts merely as a checklist, and it does not attempt to attack the target or perform any kind of disruptive action. It simply compares certain patterns and detects trivial vulnerabilities. This does not interrupt or significantly slow down the “spidering”.

Once an acceptable number of children pages has been found, the active scan begins. The active scan is the proper “vulnerability scan” per se. Unlike the passive scan, this one will actually try to attack the web application with attacks targeting specific vulnerabilities (payloads), and then determine whether the attack was successful or not. It could be seen as the automated version of pen-testing, although far simpler and not as exhaustive. The payloads include common attacks such as injection or cross-site scripting, among many others.

Once the active scan is over, the program will hold all the information it needs. The next and last step is to send these results to a file that can later be read by a human. The pdf format is chosen for its portability and system interoperability. This report file will be generated with the help of GitBook, more specifically the Jinja2 templating language included in GitBook.

While the program code can theoretically be ran from any device that satisfies all dependences (such as having Nmap and OWASP ZAP installed), the project focuses on using a Raspberry Pi as the containing device, chosen for its accessible price and its small, portable size. Since the Raspberry Pi lacks any peripherals by default, the possibility of controlling the Raspberry Pi through a mobile device is commented in the project. This is achieved via VNC (Virtual Network Computing) and a USB connection.

Additionally, a very simple graphical user interface is included, created using the Pygame library for Python. It allows text to be more easily read in the small screen of a mobile device.

# **1 Introduction**

## **1.1 Summary**

The aim of this project is to develop a vulnerability scanner for web applications that can, in an automated fashion and with minimal user input, scan a web application and generate a report detailing any vulnerabilities found along with general recommendations to mitigate them.

The program will be as minimalistic and easy to use as possible, so that a person without any technical know-how might be able to run it.

The source code can be found in the following repository:

<https://github.com/svm92/tfg>

The template used for the reports can be found in the following repository:

[https://github.com/svm92/test\\_report](https://github.com/svm92/test_report)

## **1.2 Motivation**

Penetration testing, or pen-testing, is a controlled attack on a system with the aim of finding vulnerabilities the way an attacker would, in order to neutralize them. While it is an effective method to find breaches, many companies forgo its use due to its high cost, which might range from anything from \$4,000 up to \$20,000 (roughly from 3,500€ to 18,000€)[1].

Studies state that nearly half of breaches exceeded \$500,000 (450,000€) in losses in 2015. A survey sent to security professionals reveals that 24% of the respondents had experienced a data or system breach in the last year, and yet 8% of them never perform penetration testing[2].

Thus, while penetration testing is certainly expensive, the losses incurred because of insecure web applications are far steeper. The relatively low budget spent in web application testing compared to other types of security practices might denote an insufficient awareness of the severity of breaches related to web applications.

The purpose of this project is not so much to build an extensive high-end vulnerability scanner, but to raise awareness about the importance of proper security testing. If such a simple and inexpensive tool is able to detect vulnerabilities in a target web application, a skilled attacker with sophisticated tools might put it in jeopardy with much more ease.

## 1.3 Objectives

This project will automate a simple security audit and generate a report, with the assistance of the OWASP ZAP vulnerability scanner. The device will try different attacks or payloads in an automated fashion to search for vulnerabilities in a target web application. Web applications are susceptible to diverse kinds of attacks, such as XSS (cross-site scripting) or SQL injection, and this makes them common targets for attackers. The findings will be saved into a portable document format (pdf) for later analysis.

While the results might not be comparable to those obtained by a manual pen-testing from a professional, it should be able to find at least common vulnerabilities, and for a very low price. If such a simple, inexpensive tool is capable of compromising a system, then it should become immediately evident that any malicious attacker would also be able exploit those vulnerabilities, and such a system should be deemed insecure and unreliable.

Along with the report, the program will offer a set of general recommendations to secure a typical SME infrastructure against the sort of attacks performed by the program, based on the kinds of vulnerabilities found in the target.

This sort of device is not intended to actually solve the issues found, but merely inform about them. As such, it is designed in such a way that any salesperson should be able to use it without requiring previous technical knowledge.

This sort of simple vulnerability scan could be used to encourage companies to enlist the services of a professional pen-tester that could remediate the found issues, as well as any other problems that an automated scan is unable to detect, in order to avoid financial damage or theft of sensitive data.



## **2 Antecedents**

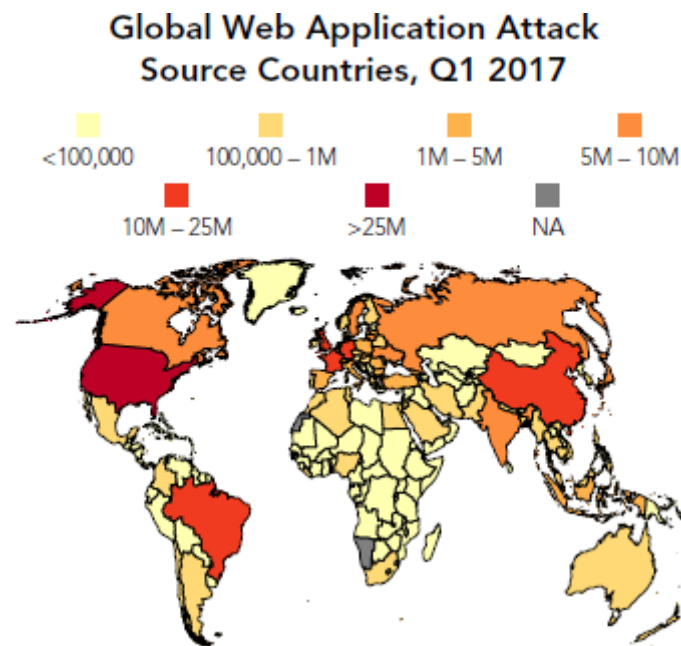
### **2.1 Background**

Computer security has always been a major concern in the field of technology. Humans first tried to purposely exploit vulnerabilities in networks back in the 60s, and then through the 70s, people would develop methods to make free phone calls by replicating the tone required to unlock the AT&T's phone network. Worms and viruses would first appear years later, created by modifying programs intended to help computers. In the coming decades, self-modifying viruses would be developed, and new methods for massive world-wide distribution would be conceived. Attacks would extend even to government and military computer systems[3][4].

In the mid-2000s, the quick growth of the Internet would only help increase the reach of virus infections thanks to emails and social media, and credit and debit cards would become some of the new targets for attacks[5].

Nowadays, with the advent of the Internet of Things and the growing globalization, the issue of computer security has only grown in importance[6]. Viruses, malware and many other varied threats and attacks can exploit vulnerabilities in a system, requiring appropriate defenses from users and companies in order to avoid information theft or financial harm. Web applications have become a prominent target for attacks due to the multiple vectors of attack that exist and the fact that they tend to hold sensitive or valuable information.

The world has seen a 35% increase in total web application attacks in the first quarter of 2017 compared to 2016. As reference, the United States is both the most targeted country for web application attacks as well as their greater source. The United States accounted for roughly 34% of all web application attacks worldwide in the first quarter of 2017, up from 28% in the previous quarter. The list of most common sources for attacks after the United States consists of Netherlands, Brazil, China, Germany, France, United Kingdom, Russia, Ukraine, and Lithuania[7], as can be seen in the following figure:



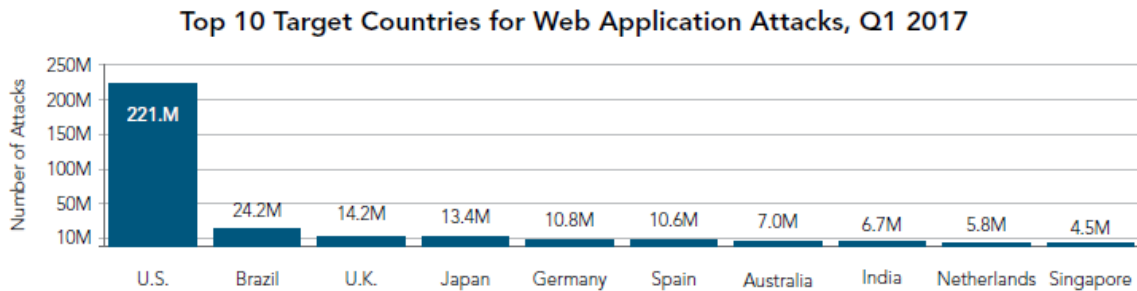
Country	Attacks Sourced	Percentage
U.S.	117,978,342	34.0%
Netherlands	43,925,118	12.7%
Brazil	28,027,292	8.1%
China	18,963,654	5.5%
Germany	16,035,037	4.6%
France	15,236,624	4.4%
U.K.	10,253,814	3.0%
Russia	8,262,292	2.4%
Ukraine	7,123,491	2.1%
Lithuania	6,566,703	1.9%

1 Global Web Application Attack Source Countries, Q1 2017

[Source: Akamai's State of the Internet Security Report]

The Netherlands is the top source for web application attacks in Eurasia, as well as the 2<sup>nd</sup> worldwide, which is noteworthy given its relatively small population density. In comparison, the United States has nearly twenty times the population but is responsible only for roughly three times the number of attacks than the Netherlands.

As for the targets of the attacks, the United States leads the list as commented above, followed by Brazil, United Kingdom, Japan, Germany, Spain, Australia, India, Netherlands, and Singapore:



2 Top 10 Target Countries for Web Application Attacks, Q1 2017

[Source: Akamai's State of the Internet Security Report]

As can be seen in the graphic, the attacks targeting the United States account for a vast majority of all attacks (nearly 70%), with the United States receiving almost ten times the number of attacks of the country in second place (Brazil). Even then, the number of attacks targeting the United States is actually down 9% compared to the last quarter. Conversely, Brazil and the United Kingdom have seen an increase in attacks of a 46% and a 30% respectively compared to the last quarter.

China and Canada, which appeared in this list in the last quarter, have been replaced by Spain and Singapore, both of which have been in this list in the past. These sudden swings may seem significant, but they are within the norms usually expected from such traffic.

## 2.2 Web Application Attacks

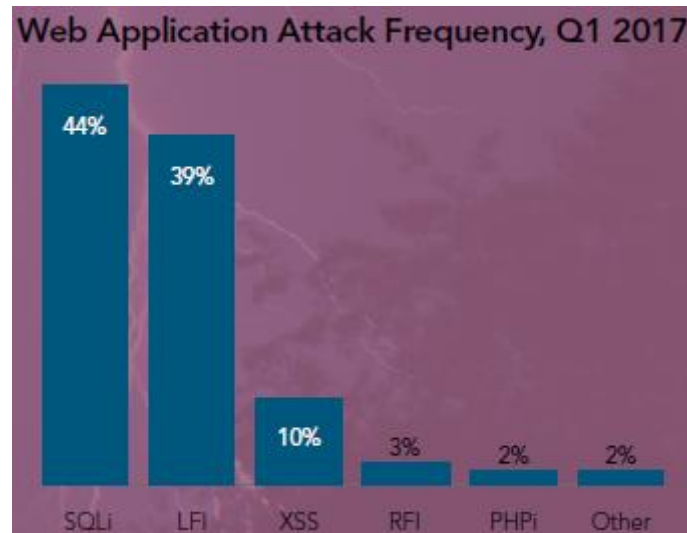
Attacks are techniques used to exploit vulnerabilities in web applications.<sup>1</sup> There exist a wide number of different possible attacks, with varying risks and difficulties in execution. Certain attacks are more susceptible to being automated than others.

According to the OWASP Top Ten Project<sup>2</sup>, the greatest security risks as of 2013 are injection, broken authentication and session management, cross-site scripting, insecure direct object references, security misconfiguration, sensitive data exposure, missing function level access control, cross-site request forgery, using components with known vulnerabilities and unvalidated redirects and forwards. A provisional list for late 2017 would remain mostly unchanged, adding insufficient attack protection and underprotected APIs to the lower end of the list while the top three positions remain unchanged.

As reference, the web application attacks most frequently used during the first quarter of 2017 were SQL injection, local file inclusion, and cross-site scripting (XSS), followed by remote file inclusion and PHP injection[7]:

<sup>1</sup> <https://www.owasp.org/index.php/Category:Attack>

<sup>2</sup> [https://www.owasp.org/index.php/Top10#OWASP\\_Top\\_10\\_for\\_2013](https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013)



3 Web Application Attack Frequency, Q1 2017

[Source: Akamai's State of the Internet Security Report]

The top three attack vectors have not changed from the last quarter. Their widespread use can be explained by the relative simplicity of their method (they are not as complex as other attack vectors) and their high efficiency. If a website does not actively protect against these vectors, more often than not it will be vulnerable to them.

Some of the most common attacks will be briefly described in the following subsections for clarity, along with some other widespread attacks.

### 2.2.1 Injection

Code injection consists on sending untrusted data to an interpreter (SQL, OS, LDAP) as part of a command or query. This might trick the interpreter into executing an arbitrary command in order to access or alter data without proper authorization.

This method is easy to exploit and can make use of diverse interpreter queries (SQL, LDAP, XPath, NoSQL), OS commands, XML parsers, or SMTP headers among others, which makes it one of the most prevalent web application attacks. Some of the most common types of injection are:

#### -SQL Injection:

This injection attack abuses the input data entry process from the client to the application in order to insert or 'inject' SQL commands to dynamically construct a SQL query that will be executed by the target system. By executing SQL arbitrary commands, the attacker might tamper with the target database in multiple ways, such as reading, altering or deleting sensitive data.

#### -Blind SQL Injection:

This attack is a specific type of SQL injection where the attacker asks true or false questions (Boolean queries) to the database, and then takes advantage of the application's generic error messages to determine the answer. This method is used by attackers to retrieve information from web applications that don't output data to the web page.

#### -XPath Injection / Blind XPath Injection:

The procedure for this attack is identical to that of an SQL injection, except that it's used for web applications that request user-supplied information to construct an XPath query for XML data. A blind variant that uses Boolean queries exists.

#### -Code Injection:

This attack injects code to be directly interpreted by the application, such as PHP code. It exploits a lack of proper input/output data validation.

#### -Command Injection:

This attack is used to execute arbitrary commands on the host operating system, when a vulnerable application passes user-supplied data such as forms or cookies to a system shell.

### **2.2.2 Broken Authentication and Session Management**

Authentication and session management functions are often not implemented correctly, presenting flaws in areas such as logout, password management or timeouts. An attacker might abuse these leaks to access passwords, keys or session tokens, or impersonate users.

### **2.2.3 Cross-Site Scripting (XSS)**

This attack takes advantage of web applications where the output makes use of the user input without validating or encoding it. The attacker might then inject malicious scripts into a trusted web site to create a malicious URL, and use social engineering to trick a victim into visiting such an URL. The user's browser will assume the script comes from a trusted source and will execute it. This might compromise any sensitive information retained by the browser, such as cookies or session tokens.

#### -Stored XSS Attacks

Also known as Persistent or Type-I XSS. The script is injected directly into a database, message forum or similar, in a way that it is permanently stored on the target server. When a user requests the stored information, that user will also inadvertently retrieve the malicious script.

#### -Reflected XSS Attacks

Also known as Non-Persistent or Type-II XSS. The script is sent to the server as part of the input of a request. A vulnerable server might then include the script in the output, such as in an error message or search result. Social engineering is then used to lure the victims into clicking a link or submitting a specific form in such a way that the injected script is reflected back to them as an output.

#### **2.2.4 Insecure Direct Object References**

A direct object reference refers to the process of exposing an internal implementation object, such as a file or database key. If incorrectly implemented, any attacker might access unauthorized data by manipulating those references. For instance, without proper access control check, an attacker might access any account by modifying the 'account' parameter in their browser's address.

#### **2.2.5 Cross-Site Request Forgery (CSRF)**

This attack requires the user to be authenticated. Then, making use of social engineering such as sending a link or abusing image tags, the attacker might trick the user into executing an unintended action. Once authenticated, the web application has no way to differentiate between a genuine request and a forged one. While this might not be used to steal data, since the attacker doesn't see the response to the forged request, it might force the victim to execute actions such as transferring funds or deleting an account.

#### **2.2.6 Clickjacking**

Multiple transparent or opaque layers are used to trick the user into unintentionally clicking on a button or link, which might redirect them to a malicious site or force them to execute an undesired action.

#### **2.2.7 Denial of Service (DoS)**

The attacker floods the target system with a large number of requests, making it unavailable for legitimate users. Alternatively, the attacker may instead exploit a programming vulnerability to achieve the same effect.

### 2.2.8 Man-in-the-middle

The attacker intercepts a communication between two systems. For instance, http transactions normally involve a TCP connection between client and server, but an attacker could split that connection into two (client-attacker and attacker-server), allowing the attacker to read or modify the exchanged data.

### 2.2.9 Man-in-the-browser

This attack follows the same approach as the man-in-the-middle attack, but it makes use of a Trojan Horse as an interceptor between the browser and its security mechanisms. It bypasses authentication factors.

### 2.2.10 Brute Force Attack

The attacker makes requests to a server testing every possible value of a parameter, analyzing the responses until a successful value is found. While the attacker might try every possible combination, it's common to instead try for specific sets of values such as a dictionary attack or rainbow tables.

### 2.2.11 Social Engineering

Social engineering refers to the practice of taking advantage of human psychology in order to access confidential information, instead of exploiting flaws in software[8]. It could be divided into multiple categories:

#### -Baiting:

The attacker leaves a malware-infected physical device such as a USB flash drive in a place where the victim is likely to find it, usually with a curiosity-inciting label. The unsuspecting victim might pick up the device and plug it into their computer, installing the malware.

#### -Phishing:

The attacker impersonates a reputed source and sends an email to the victim, trying to trick the recipient into sharing personal or confidential information, or clicking a malicious link. When the target is a specific individual or organization, it receives the name of 'spear phishing'.

#### -Pretexting:

The attacker will make use of a pretext (an invented scenario or elaborate lie) to manipulate the victim into divulging confidential information, such as a scammer pretending to need certain information pertaining to their target in order to confirm their identity.

-Scareware:

Scareware is a type of malware that attempts to trick victims into downloading malicious software. Scareware might disguise itself as system messages coming from an antivirus or firewall application, and falsely inform about a number of inexistent problems, suggesting purchasing or downloading actual malicious software to fix the supposed problems.

## 2.3 Attack Prevention

### 2.3.1 Injection

Preventing injection attacks requires the interpreter to be able to make a distinction between untrusted data and actual commands and queries, which can be achieved by one of many ways.

The most obvious solution would be to forgo use of the interpreter altogether and instead use a safe API (Application Programming Interface), or at least an API that provides a parameterized interface.

If APIs can't be used for this purpose, injection can be prevented using an interpreter and making careful use of escaping routines. Injection attacks will usually involve injecting special characters that are normally not found in legitimate queries (such as '<' or '>'), so checking for these special characters might help differentiate between an actual user query and a potential script from an attacker. By escaping these special characters (using the escape syntax for that particular interpreter) or sanitizing the input, it should be possible to block those kinds of attacks.

Another possibility is white list validation, which only takes in data structured in a particular way, such as properly formatted dates or e-mail addresses, and disregards the rest of the input.

### 2.3.2 Broken Authentication and Session Management

The most important advice for an organization is to make sure that their developers have a single set of strong authentication and session management controls. It's also important to prevent XSS flaws that could be abused to steal session IDs.

Authentication credentials should always be hashed or encrypted when stored, and should not be sent over unencrypted connections. Special care must be taken with account management functions such as password recovery, for they could be abused to guess or overwrite credentials.



Session IDs are especially vulnerable. They shouldn't appear in the URL, they should timeout, and they should be rotated after a successful login. Also, it's important that any used authentication tokens are properly invalidated during logout.

### 2.3.3 Cross-Site Scripting (XSS)

Preventing cross-site scripting requires separating legitimate browser content from untrusted data. The simplest method, effective against both reflected and stored XSS, is to filter data according to its HTML context (body, attribute, URL) and escape any data deemed untrusted.

Similar to injection, white list server-side input validation can also be used, allowing only data that follows a specific format, such as length ranges or valid characters, and disregards everything else.

Content Security Policy (CSP) can be used to restrict from which location and what type of resources is the client browser allowed to load, so it could be used to defend against XSS across an entire webpage.

### 2.3.4 Insecure Direct Object References

The simplest defense against insecure direct object references is to establish an appropriate check access control. Whenever an object or service is requested, the application should perform an access control check to confirm whether the user is authorized for the requested object.

Another method is to use an indirect reference map, which would be used for mapping from a set of internal object references to a set of indirect references. This way, the user would only have access to these per-user specific indirect references, which would be sent to the database to retrieve the actual request (the internal object reference), such as a filename or database key.

### 2.3.5 Cross-Site Request Forgery (CSRF)

The most effective way to protect against CSRF is to include an unpredictable token in each HTTP request, so that the attacker would be unable to execute a CSRF without knowledge of this token. This token should be unique per session and should preferably be included in a hidden field, so that it is sent in the body of the HTTP request and not in the URL.

Another solution is to require the user to confirm all their transactions. This can be achieved by asking them to reauthenticate or requesting proof that they are an actual user (such as by a CAPTCHA).

## 2.4 Manual Testing

### 2.4.1 Introduction

Before the testing process can be automated, it is necessary to have an understanding of how to manually uncover vulnerabilities. In order to achieve this, a few basic testing examples will be described. They will all be carried out without the assistance of pen-testing tools, using a purposely vulnerable web server.

For the purposes of these examples, a virtual machine hosting an Ubuntu 16.04 system will be used, but the basic procedure is applicable to other systems. The used vulnerable web server will be OWASP Mutillidae 2.6.43.<sup>3</sup> It should be noted that the vulnerabilities found within this server are genuine, and should never be deployed outside a virtual machine.

### 2.4.2 Injection

An injection attack requires a field that accepts input data, and manipulating it in order to create an arbitrary query or command. The syntax will change depending on the interpreter. Error messages can help find what type of database the application uses, since many developers allow their applications to display error messages by default.

For example, let's assume the server stores credentials in a MySQL table with this format:

	Username	Password
1	User1	Pass1
2	User2	Pass2
3	User3	Pass3

The table might contain any number of additional columns, and the passwords might or might not be encrypted or hashed. In order to retrieve information, the server will use MySQL queries to access the required information. For instance, inputting the username "admin" and the password "adminpass" might construct a query such as:

```
SELECT * FROM accounts WHERE username='admin' AND password='adminpass'
```

This will search through the 'accounts' table in the database for any row where both the username and the password correspond to the inputted data. The query is thus dynamically constructed, taking user input and sending it to the function as arguments. If the input is not sanitized, it's possible to abuse this to force the server to execute an arbitrary query. An example of this might be a login page.

---

<sup>3</sup> [https://www.owasp.org/index.php/OWASP\\_Mutillidae\\_2\\_Project](https://www.owasp.org/index.php/OWASP_Mutillidae_2_Project)

**Please sign-in**

**Username**

**Password**

Login

*Dont have an account? [Please register here](#)*

#### 4 Standard Login Form

For instance, let's assume that an attacker wants to log in as 'admin' without knowing the password. Since the input is passed on as arguments to the MySQL query, this would allow the attacker to modify the query. Knowing that comments in MySQL are inserted with a double hyphen (--) followed by a space, the attacker might input this in the username field:

**Please sign-in**

**Username**

**Password**

Login

*Dont have an account? [Please register here](#)*

#### 5 Bypassing a Login Form

This will construct the following query:

```
SELECT * FROM accounts WHERE username='admin' -- ' AND password=''
```

Since everything behind the double hyphen is commented out, this query is equivalent to:

```
SELECT * FROM accounts WHERE username='admin'
```

This query will return the row with the username 'admin', regardless of the password, allowing an attacker to bypass authentication.

You are logged in as admin

Logout

#### 6 Succesful Login Form Bypass

Expanding on this, it would be possible to force the server to dump the whole database by constructing a query that returns true for every row. For example, in a user lookup page, this input:

**Please enter username and password to view account details**

**Name**

**Password**

**View Account Details**

*Dont have an account? [Please register here](#)*

#### 7 User Lookup Bypass

Would construct this query:

```
SELECT * FROM accounts WHERE username='' OR 1=1 --  
' AND password='adminpass'
```

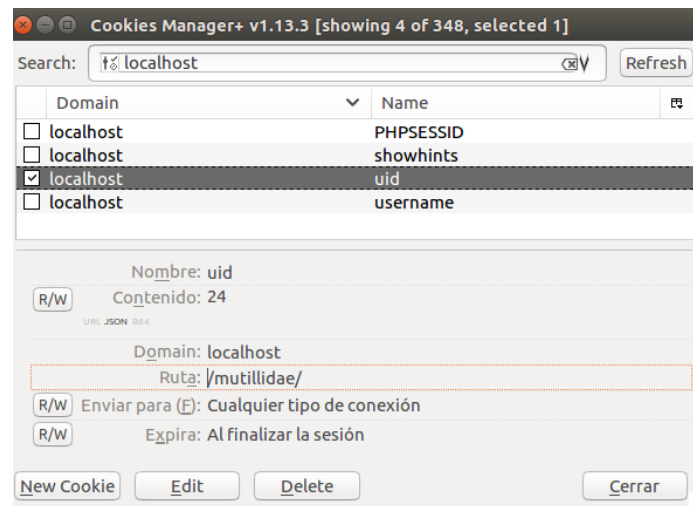
Ignoring the commented out part, this query will return every row where either of the two conditions evaluate to true. By inputting a trivial second condition that will always be true (such as 1=1), this query will return every row in the database, dumping the credentials of every user.

While these examples have been done in MySQL, the methodology is identical for other different interpreters, varying only in the syntax used to construct queries.

### 2.4.3 Broken Authentication and Session Management

There are multiple ways to hijack sessions depending on how the particular target web application is configured. As mentioned before, in an insecure page injection can be used to extract password lists or even bypass authentication altogether, but there are other ways to achieve this.

Depending on how a page works with cookies, they could be exploited to obtain administrative privileges or impersonate another user. There are many ways to edit cookies. For the purposes of this example, the 'Cookies Manager+' add-on for Mozilla Firefox will be used. First, it is necessary to understand how the target web application manages cookies. An easy way to begin is to actually register as a normal user, in order to make the web application generate a genuine session cookie.



### 8 Cookie Manipulation

Analyzing the newly generated cookies can give insight on how to proceed. For this particular example, registering as a new user has created two new cookies, 'uid' and 'username', corresponding to the user's id and the username. In most cases the relationship between the cookies and the session will not be as obvious, so it might be necessary to create more than one account to compare the cookies and see how they relate to their respective sessions. For this example, trial and error shows that changing 'uid' to 1 and 'username' to admin is equivalent to logging in as the web's administrator.

#### 2.4.4 Cross-Site Scripting (XSS)

Much like injection, cross-site scripting attacks make use of fields that allow the user to input data and then display it back in the output without proper encoding. Two variations exist, reflected XSS and stored XSS.

Reflected cross-site scripting is transient and is not stored within the server. Any page that directly outputs the user input without sanitizing it is vulnerable. For example, let's assume a user lookup page takes in a username and a password and then displays the username back without sanitizing it. For instance, inputting 'qwerty' would return:

Authentication Error: Bad user name or password

**Please enter username and password to view account details**

**Name**

**Password**

[View Account Details](#)

*Don't have an account? [Please register here](#)*

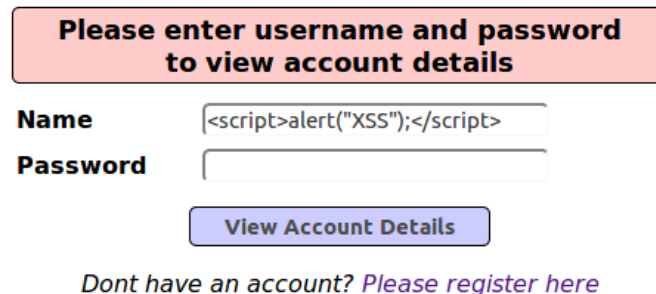
**Results for "qwerty".0 records found.**

### 9 Login Error Example

If the developer is careless and passes on the parameters directly through the URL, the URL might look like this:

`.../mutillidae/index.php?page=user-info.php&username=qwerty&password=123&user-info-php-submit-button=View+Account+Details`

If that is the case, it would be possible to input a malicious script in the username field in order to generate a malicious link that will execute that script when accessed. The attacker could then make use of social engineering to trick a victim into clicking such a link and executing the script. A very simple example of this could be a script that displays a warning message:



**Please enter username and password to view account details**

**Name**

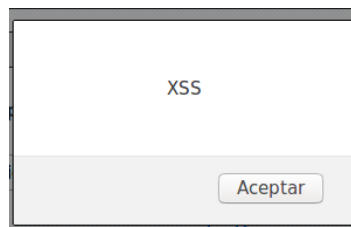
**Password**

[View Account Details](#)

[Dont have an account? Please register here](#)

#### 10 Reflected XSS Example

This would generate the URL: `.../mutillidae/index.php?page=user-info.php&username=%3Cscript%3Ealert%28%22XSS%22%29%3B%3C%2Fscript%3E&password=&user-info-php-submit-button=View+Account+Details`. The apparent garbling in the URL happens because all non-alphanumeric characters are substituted by their equivalent ASCII codes (such as '`<`' becoming `%3C`). Upon accessing this URL, the script is executed and the warning message is displayed.



#### 11 Succesful Reflected XSS

While this particular example is benign, an attacker could force the execution of any arbitrary script of their choice using this method.

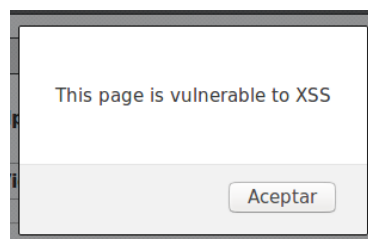
Stored cross-site scripting differs from the reflected variant in that it doesn't need to target a specific victim. Instead, as its name implies, the script is directly stored in the server and executed whenever any user accesses that page or performs a specific action. Pages vulnerable to stored XSS include all those that allow users to input data that is stored in the server without sanitizing it first. Common examples are forums, blogs or guestbooks.

An example could be a blog that allows users to leave comments that are then displayed to every other visitor. If the inputted comment was a malicious script, every visitor would accidentally execute it upon visiting the page.

**Add blog for anonymous****Note: <b>, <i> and <u> are now allowed in blog entries**

This seems like a normal comment.

```
<script>
alert("This page is vulnerable to XSS");
</script>
```

**Save Blog Entry****12 Stored XSS Example** **View Blogs**

2 Current Blog Entries			
	Name	Date	Comment
1	anonymous	2017-02-24 16:39:49	This seems like a normal comment.
2	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?

**13 Succesful Stored XSS**

This entry would be added as a permanent comment, and in most cases it would look no different than a normal comment, but it would execute the script whenever a user visits the page. Subtler scripts could execute without the user noticing anything wrong. The only way to solve this would be for the webmaster to delete the malicious entry from the server. In particularly pernicious cases with self-replicating scripts, the only solution might be to restore the database to a previous backup or reset it altogether.

**2.4.5 Insecure Direct Object References**

Insecure direct object references will take place whenever an application tries to access a resource using the resource itself as the reference to look up for. Through trial and error, it could then be possible to gain access to a confidential file without having the required privileges.

If the web application has a folder called 'passwords' that contains a text file of the name 'accounts.txt', it could be possible to access it directly through the URL `/passwords/accounts.txt`, even if the text file itself isn't accessible through normal means in the webpage. Another example would be a page that shows the user's account directly

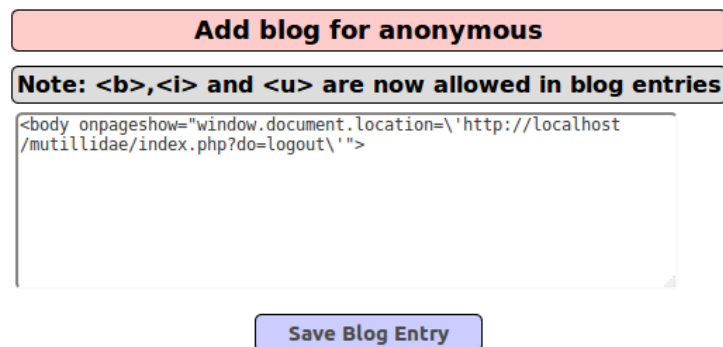
through the URL as a parameter, allowing attackers to just modify that parameter to access any account, such as `/userInfo?account=admin`.

#### 2.4.6 Cross-Site Request Forgery (CSRF)

Before this kind of attack can take place, the target must be authenticated. Then, the attacker runs a script into the target's browser to forge a request. The website has no way to know that the request isn't genuine, since it comes from the user's browser.

Cross-site request forgery can happen in multiple ways, and can be combined with many of the previous attacks. The final objective is to make the target's browser execute a script, and this can be achieved by many different ways such as sending an URL or XSS.

As an example, in a forum or comment page that doesn't sanitize input an attacker might use cross-site scripting to post an entry with the following HTML code that forces any visitor to logout:



**Add blog for anonymous**

**Note: `<b>`, `<i>` and `<u>` are now allowed in blog entries**

`<body onpageshow="window.document.location=\\'http://localhost/mutillidae/index.php?do=logout\\'\">`

**Save Blog Entry**

##### 14 CSRF Example

These kinds of attacks are not effective if the user isn't authenticated beforehand. Other possible approaches for attacks could include forcing a user to make a purchase or sending money to the attacker's account.



## 3 Implementation

The code will be written in Python. Python is a high-level programming language that puts emphasis on code readability. Unlike many other programming languages that break up code blocks with curly braces or specific keywords, Python instead uses whitespace indentation to mark the indentation level of each line of code.

The program will execute in three distinct stages. It will first begin by gathering all information it will need to work, such as what IPs and ports to scan. The second stage will be the actual vulnerability scan per se, taking in the information from the previous step as the input and sending the output to the next and last stage, the automated generation of the report.

The code will make use of diverse third-party tools such as Nmap or OWASP ZAP, which will be described in the following sections as they are introduced. Most of these tools have specific modules that allow them to easily interact with Python, and grant almost full control of their functions through scripting.

### 3.1 Information Gathering

Before the scan can start it is necessary to know what web applications will be scanned, the IP address of their hosts and which port do they reside in. The information gathering phase will be divided in three sequential stages: a first stage of interface scanning to detect what interfaces the device hosting the script is connected to and convert them to CIDR format, a second stage of network scanning to discover live hosts, and a third and final stage of port scanning. Nmap will be used in the two later stages.

Nmap<sup>4</sup> (“Network Mapper”) is a free, open source utility used for diverse tasks such as network discovery and security auditing. It was conceived as a tool to assist in the scanning of large networks, but also works against single hosts. Some of the multiple techniques it implements include port scanning, OS and version detection or ping sweeps, among others. A module for Python exists<sup>5</sup>, which makes it easy to implement Nmap functions directly through Python.

#### 3.1.1 Interface Scan

Before a network can be scanned, it is imperative to first determine *which* network is to actually be scanned. So, the first step will be to determine what network the device is connected to.

The device, by virtue of being connected to a network, will be granted an IP address. These IP addresses are used to identify any particular device inside a given network. They usually take the form of a set of four numbers, each ranging from 0 to 255 (such as 127.0.0.1). Since the

---

<sup>4</sup> <https://nmap.org/>

<sup>5</sup> <https://pypi.python.org/pypi/python-nmap>

device will know its own IP address at any point, it can be used as a starting point to determine the network.

Other than an IP address, a network is identified by its netmask, which determines what other IP addresses belong to the same subnet and should also be scanned in the following stages. It can be represented as a set of four numbers much like an IP address, or as a single number denoting how many bits in an address are part of the network, out of a total 32 bits. For instance, a 20 would denote that the first 20 bits are the network part of the address, and so the other 12 bits are used for specific host addresses under the same subnet.

Thus, a network can be fully characterized by an IP address and a netmask. Knowing this, it's common to represent a network in a human-readable format that includes this information. The most widely accepted representation is CIDR<sup>6</sup> (Classless Inter-Domain Routing) notation. Under this notation, networks are written as an address followed by a slash and then the number of bits of the netmask (such as 192.168.0.0/16).

The *netifaces* module for Python makes it trivial to fetch information about all interfaces the device is connected to. Then, the *socket* module included by default in Python can help find information related to their IPs, such as their address and netmask. Finally, the *netaddr* module can be used to turn this information into a usable CIDR form.

### 3.1.2 Network Scan

Trying to scan every single port of every potential IP address would be so time-consuming as to be impractical. It is a slow procedure that is rarely necessary. More often than not, the only IP addresses of interest are those which hold live hosts.

Host discovery works by sending specific request packets ("pings") to a wide range of IP addresses and then detecting what particular IP addresses answer back<sup>7</sup>. Nmap offers different ways to scan a network for live hosts, but the "no port scan" ("-sn" switch) will be chosen for being relatively less intrusive and faster than other alternatives, while still detailing what hosts are up in a given network.

The way it works is by sending an ICMP echo request ("Internet Control Message Protocol") to each target and then waiting for a response. By default, Nmap will send a TCP ACK packet to port 80 and a TCP SYN packet to port 443. Ports 80 and 443 are considered the default ports for http and https, respectively.

The result of the network scan will be a list of IP addresses that responded to the requests sent by Nmap. This list will be saved so that Nmap can later scan the ports on each of these IP addresses one by one.

---

<sup>6</sup> <http://searchnetworking.techtarget.com/definition/CIDR>

<sup>7</sup> <https://nmap.org/book/man-host-discovery.html>

### 3.1.3 Port Scan

Any given IP address will be associated with 65536 TCP ports and another 65536 UDP ports, where TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are two of the protocols that make up the TCP/IP protocol suite, universally used for data communication across the Internet[9]. The first 1024 TCP ports are commonly associated with specific services such as HTTP (80), HTTPS (443), DNS (53) or FTP (20). Note that this is merely a convention, and is not enforced (so there could be HTTP services in any port other than 80).

Port scanning consists on sending out connection requests to every port corresponding to an IP address sequentially, noting which ports respond. The scan can then follow up with a more in-depth scan of those ports that responded in order to determine what kind of service they are running. Scanning the whole range of 65536 ports is slow and rarely necessary, so it's common for scanners to limit their reach to a smaller array of commonly used ports. By default, Nmap will only scan the 1000 more frequently used ports.

By default Nmap will run a SYN scan, capable of scanning thousands of ports per second in absence of restrictive firewalls<sup>8</sup>. This scan sends a SYN packet to every target port and waits to receive a SYN/ACK packet in response, which would indicate that the target port is open. Since this scan doesn't open a full TCP connection, it is sometimes referred to as half-open scanning.

The scan will result in a list of open ports associated to each live host found during the previous step. Nmap can also perform service detection to determine with reasonable accuracy what kind of service is associated with each port. This allows the program to detect HTTP or HTTPS services outside of their most commonly use ports.

Once the program has composed a list of hosts and ports that might potentially have web applications, this list is sent to the next stage for an automated scan.

## 3.2 Automated Testing

The fastest, most efficient way to perform an automated test is to make use of one of the many pen-test assisting tools that exist. For the purposes of this project, OWASP ZAP (Zed Attack Proxy) will be used to automatically scan a target web application.

ZAP is an integrated penetration testing tool that can be used to find a wide array of vulnerabilities in a web application. The tool is meant both for unexperienced beginners as well as professionals. ZAP provides tools for manual testing as well as automated scanners, but this project will only make use of the second. ZAP has a Python client API that allows it to work directly through Python by use of the ZAPv2 module<sup>9</sup>.

In order to perform an automated scan on the target web application, the first step is to open the OWASP ZAP program. This can be achieved with the help of the 'OS' and 'Subprocess' modules for Python[10]. The program will be started in headless mode (without a graphical

---

<sup>8</sup> <https://nmap.org/book/man-port-scanning-techniques.html>

<sup>9</sup> <https://github.com/zaproxy/zaproxy/wiki/ApiPython>

interface) in order to minimize loading times. The script will continuously try to connect to the target webpage until it can detect the program. Once the program is open, the actual scan can be divided in two stages: the spidering stage and the active scan stage.

The program is only supplied with the root URL (parent page) of the web application, so it needs some way to find its subdirectories (children pages). A spider or crawler is a bot that does exactly that, fetching any children pages under the parent URL to later scan them. OWASP ZAP features a spider that can be directly called from Python. If the scan takes too long, it is possible to limit the number of children to scan in the code. This would reduce the total scan time, but would also lower the number of vulnerabilities found.

While the program is running the spider, it will passively scan every received response, searching for easy to find vulnerabilities (similar to a checklist). This is a passive scan done in the background without interrupting the exploration of the web application.

According to the official OWASP ZAP documentation, the passive scan searches for any of the following vulnerabilities<sup>10</sup>:

#### -Application Errors

Checks server responses in order to find responses that contain known error strings.

#### -Cache Control

Checks the cache-control header field to ensure it follows the industry's best practices.

#### -Content Type Missing

Ensures that the Content-Type header exists and its value isn't empty.

#### -Cookie HttpOnly

If a cookie is set with the "HttpOnly" flag, browsers will know that only client side scripts should interact with that cookie. This check ensures that all cookies set are flagged as "HttpOnly".

#### -Cookie Secure Flag

If a cookie is set with the secure flag, it will not be sent during a plain HTTP session. This check analyzes all cookies generated during HTTPS sessions to make sure they include this flag.

#### -Cross Domain Script Inclusion

Checks whether there are scripts that don't come from the domain hosting the content.

#### -Header XSS Protection

The X-XSS-Protection header will keep a page from loading if it detects a potential XSS attack. This check confirms the existence of this header and whether it is correctly configured.

---

<sup>10</sup> <https://github.com/zaproxy/zap-core-help/wiki/HelpAddonsPscanrulesPscanrules>

### -Mixed Content

A HTTPS connection is considered more secure than a HTTP one, but not so if the content is sourced via HTTP, which would defeat the purpose of HTTPS in the first place. This check analyzes content served via HTTPS to ensure none of it is sourced via plain HTTP.

### -Password Autocomplete

This is considered an unsafe practice. This check examines all password fields to ensure their 'autocomplete' setting is disabled.

### -Private Address Disclosure

Checks whether the response contains private IP addresses (RFC 1918).

### -Session Id in URL Rewrite

Ensures that the session token is not written to the URL.

### -X-Content-Type-Options

MIME sniffing<sup>11</sup> may occur when a document has no MIME type. In this situation, some browsers will try to guess the correct MIME type by looking at the resource. This represents a potential vulnerability, so the X-Content-Type-Options header exists to prevent MIME-sniffing. This check ensures that this header exists and is correctly set to 'nosniff'.

### -X-Frame-Options Header Scanner

The X-Frame-Options header<sup>12</sup> defines how a page will be rendered. It can forbid the page's content from being embedded into other sites, which can prevent click-jacking attempts. This check ensures that this header exists and is valid.

Once an appropriate number of URLs has been fetched and the passive checks have been applied, the proper scanning stage begins. Unlike the passive scan performed during the spidering process, this is an active scan that applies attacks or payloads against the target application in order to find vulnerabilities.

According to the official OWASP ZAP documentation, the attempted attacks are as follows<sup>13</sup>:

### -Buffer Overflow

In the case of a buffer overflow, the extra data might get written into a sensitive memory location. To check for this flaw, the program sends large strings of input text and checks for anomalous behavior.

### -Client Browser Cache

---

<sup>11</sup> <https://www.keycdn.com/support/what-is-mime-sniffing/>

<sup>12</sup> <https://www.keycdn.com/blog/x-frame-options/>

<sup>13</sup> <https://github.com/zaproxy/zap-core-help/wiki/HelpAddonsAscanrulesAscanrules>

Checks the headers of secure pages to ensure they don't allow browsers to cache the page.

#### -CRLF Injection

CRLF refers to the Carriage Return and Line Feed special characters used to denote the end of a line. It might be abused by an attacker to split a HTTP response, which can lead to cross-site scripting or cache poisoning<sup>14</sup>. The program tests for this vulnerability by trying to inject a new header into the response by sending a parameter preceded by CRLF characters.

#### -Cross Site Scripting (Reflected)

First, it submits a 'safe' value and checks the response for any instances of this value. Then, for any locations where this value was found in, the program attempts a series of attacks aimed at that location to confirm whether a XSS attack is possible.

#### -Cross Site Scripted (Persistent)

It begins by submitting a 'safe' value much like the reflected variant, but it then spiders the whole application to find any locations where the value might be found and tries to attack them.

#### -Code Injection

Submits PHP and ASP attack strings as URL parameter values and then examines the response.

#### -Command Injection

Submits \*NIX and Windows OS commands as URL parameter values that would produce a specific output should the application be vulnerable. If this output isn't found in the response, then it tries a more complex approach that involves blind injection by submitting sleep instructions and timing the responses.

#### -Directory Browsing

Tries to gain access to a directory listing by examining the response for patterns commonly used with web server software such as Apache or IIS.

#### -External Redirect

Tries to force a redirect by sending specific parameter values to the URL, and then checks the headers and response to check whether a redirect occurred.

#### -Format String Error

Sends strings of input text that compiled C code would interpret as formatting parameters, and then looks for any anomalies.

#### -Parameter Tampering

---

<sup>14</sup> <https://www.acunetix.com/websitesecurity/crlf-injection/>

Submits requests with parameter values known to cause errors, and compares the response against patterns found in common errors in Java, Microsoft VBScript, PHP and others.

#### -Path Traversal

Tries to access files and directories outside of the web document root using combinations of pathname prefixes and names of local files typically found in Windows or \*NIX systems.

#### -Remote File Include

Passes an external URL as a parameter value for the current URL and checks whether this causes the title of the page to change.

#### -Server Side Include

Detects the OS the server is running on, and then sends a HTML SSI directive. The response is then analyzed to check whether it matches with a pattern indicating the SSI (Server Side Include) was successful.

#### -SQL Injection

Attacks URL parameters and form parameters with valid and invalid SQL syntax, using diverse SQL injection techniques such as error, Boolean or Union based injection.

Once the scan is done, OWASP ZAP saves all the collected information pertaining to the found vulnerabilities inside a variable that will later be accessed and analyzed in order to get the needed data to generate the report.

The headless instance of OWASP ZAP will be closed via Python as soon as the report generation ends, or at any point that an error occurs and the execution of the program is interrupted.

## **3.3 Report Generation**

The next step is to make use of the information gathered during the previous step to generate a human-readable report that summarizes the vulnerabilities found during the scan and their severity. The report will be formatted according to the 800-115 NIST guideline[11], using a penetration test report from Offensive Security as a template[12]. The resulting output will be a pdf file generated using GitBook.

First, it's necessary to create a template for the report in GitBook. GitBook uses Jinja2, a templating language for Python modelled after Django's templates. Its syntax makes it possible to not only create templates for variables, but also to iterate through lists and use conditionals[13].

GitBook will take into account any variables found within the *book.json* file when generating the pdf file<sup>15</sup>. The purpose of the code will then be to dynamically create this *book.json* file using the appropriate variables obtained during the previous scan. This is achieved by collecting the variables of interest inside a list that is later dumped to the json file.

The report will begin with a short introduction mentioning what URLs were scanned and what version of OWASP ZAP was used for the scan. It will follow with a summary of results, listing every vulnerability that was detected at least once, along with a brief description of each vulnerability and a list of all URLs where they were found. The next section is an attack narrative, describing in some level of detail the payloads that were applied during the active scan, followed by a brief risk rating section assigning risk levels to the vulnerabilities. The report will end with a set of recommendations for each vulnerability and a brief description of the methodology used to automate the scan and generate the report.

---

<sup>15</sup> <https://gitbookio.gitbooks.io/documentation/content/format/templating.html>



## 4 Integration

### 4.1 Device Integration

The program code will be contained within a Raspberry Pi. Raspberry Pi is a low-cost computer of small size developed by the Raspberry Pi Foundation in the United Kingdom with the purpose of helping promote the study of computer science and bringing programming to developing countries, but it also sees extended use in other fields such as robotics<sup>16</sup>.

Part of the success of the Raspberry Pi is due to the wide variety of models that exist, as well as their accessible price. By default the Raspberry Pi doesn't include peripherals such as mice, keyboards or even a screen, but most models include several ports that allow such devices to be connected externally. Some models include an Ethernet port that allows the Raspberry Pi to access the Internet.

Raspberry Pi can work with many different operating systems, but this project will use Raspbian, a Debian-based Linux distribution and the primary operating system officially provided by the Raspberry Pi Foundation.

Note that this program can be executed from any laptop, and does not strictly require a Raspberry Pi to run. A Raspberry Pi is chosen for the purposes of these examples because of its small size and portability, as well as its low cost.

### 4.2 Dependencies

In order for the program code to run, a series of requirements or dependencies must be installed beforehand in the device that will be used to run the scan, whether it is a Raspberry Pi, a laptop or any other compatible device.

First, the appropriate repositories must be cloned to the device. This can be easily achieved through the command line in any operating system that accepts orders via bash shell such as GNU/Linux or OS X. If prompted for a username and password while cloning from GitBook, those fields might be left blank.

```
$ sudo apt-get update
$ sudo apt-get install git-all # Only needed if git is not installed
$ git clone https://github.com/svm92/tfg.git
$ git clone https://git.gitbook.com/svm92/test-report-skeleton.git
```

Installing GitBook is also required, assuming it's not installed already:

```
$ sudo apt-get install npm # Needed to install Gitbook
$ sudo npm install gitbook-cli -g && gitbook update
$ sudo apt-get install calibre # Needed for pdf functionality
```

---

<sup>16</sup> [http://elinux.org/RPi\\_Hub](http://elinux.org/RPi_Hub)

The Python code requires the following network manipulation libraries:

```
$ pip3 install netifaces
$ pip3 install netaddr
```

Nmap and its Python API can be installed with the following commands:

```
$ sudo apt-get install nmap
$ pip3 install python-nmap
```

Finally, it's necessary to install OWASP ZAP and its Python API. It might be downloaded and installed from the official page, or directly via shell like this:

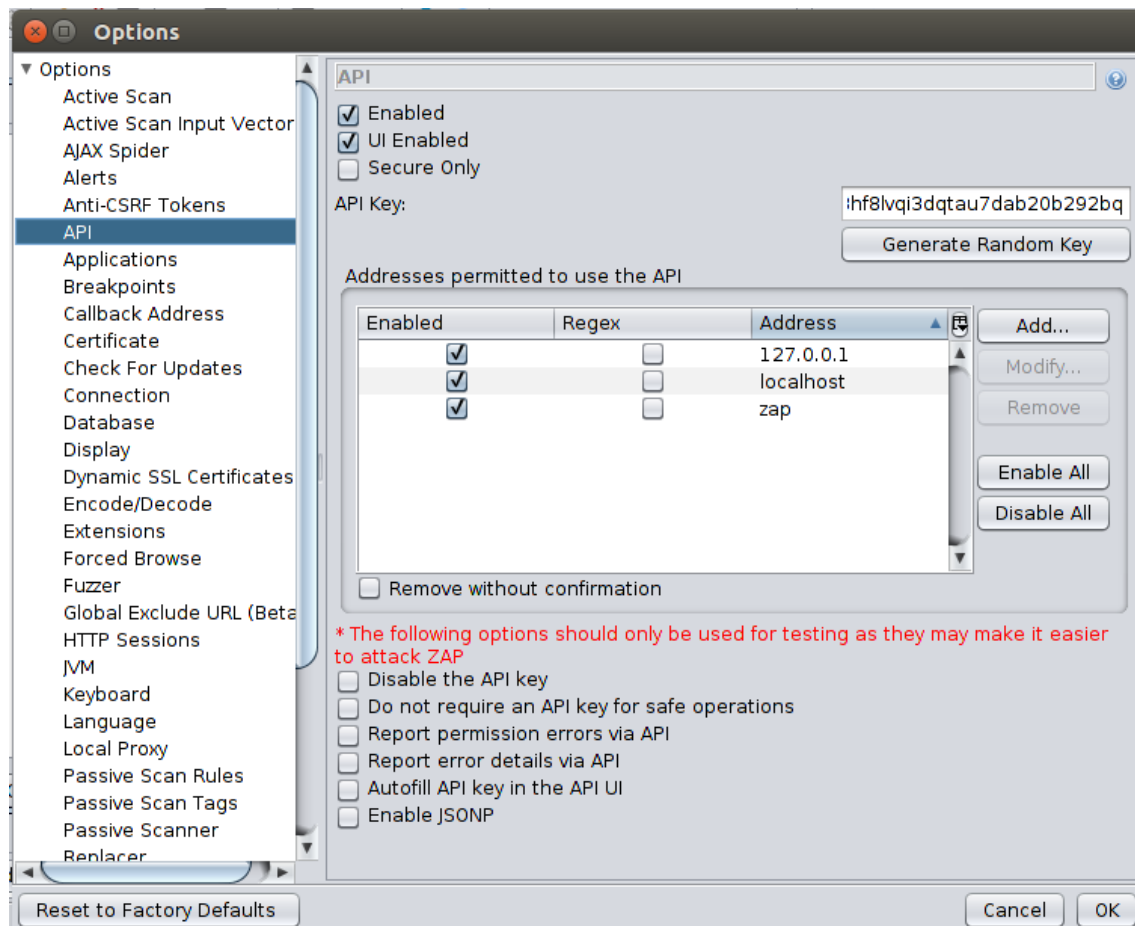
```
$ cd /home/pi/
$ wget -O "/home/pi/OWASP ZAP/"
"https://github.com/zaproxy/zaproxy/releases/download/2.6.0/ZAP_2.6.0_
Linux.tar.gz"
$ tar -xzf "OWASP ZAP" -C "/home/pi/OWASP ZAP/"
$ rm "OWASP ZAP"
$ pip3 install python-owasp-zap-v2.4
```

The target containing directory might be changed at will, remembering to change the appropriate *owasp\_location* variable inside the code to reflect that change. It is recommended to always install the newest version available.

In order to solve possible conflicts between different versions of urllib3, running the following command is recommended:

```
$ pip3 install requests --upgrade
```

As a last step before the code can be executed, OWASP ZAP must be opened once (zap.sh file) in order to configure the API key. The API key is an optional security measure that isn't strictly required to run OWASP ZAP, but is nonetheless recommended since it helps prevent malicious sites from accessing the ZAP API. Its configuration can be found under Tools → Options → API:



15 API key configuration in OWASP ZAP

The “Generate Random Key” button will, as its name implies, generate a new API key that can be copied to the `api_key` variable used in the code. Alternatively, the “Disable the API key” option may be ticked and the `api_key` variable left blank, but this is again not recommended.

## 4.3 Remote Control

In order to avoid the need for a keyboard and a screen, it’s convenient to be able to control the Raspberry Pi remotely. This can be achieved via Virtual Network Computing (VNC).

VNC is a graphical desktop sharing system consisting of a server and a viewer<sup>17</sup>. The VNC server (Raspberry Pi in this case) will receive any keyboard or mouse input coming from the VNC viewer, which will also act as a screen. The VNC viewer can be anything from a mobile phone to a tablet, or even a laptop.

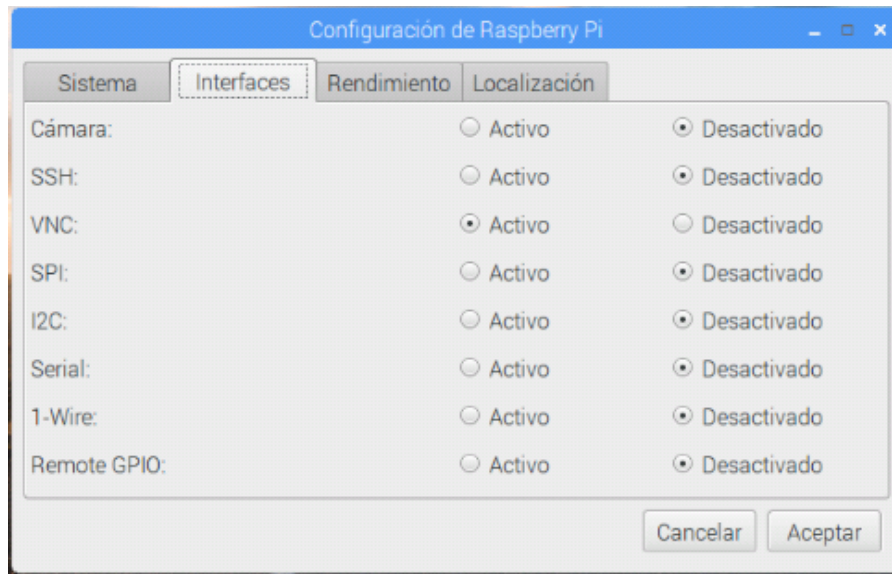
By default, Raspbian includes VNC Connect from RealVNC, which includes both a VNC server and a viewer. It will be used as the VNC server for the Raspberry Pi, although any other VNC server could be used in its place. There is a wide variety of VNC viewers available for different platforms.

<sup>17</sup> <https://www.raspberrypi.org/documentation/remote-access/vnc/>

First, while VNC Connect is included by default in Raspbian installations, it's still a good idea to run the following commands to ensure the latest version is installed:

```
sudo apt-get update
sudo apt-get install realvnc-vnc-server realvnc-vnc-viewer
```

Next, it is necessary to enable the VNC server to allow external connections to the Raspberry Pi. This can be done graphically by selecting Menu > Preferences > Raspberry Pi Configuration > Interfaces and then setting the VNC option to Enabled. Alternatively, the VNC server can be enabled via command line using `raspi-config` and navigating to the Interfacing Options section.



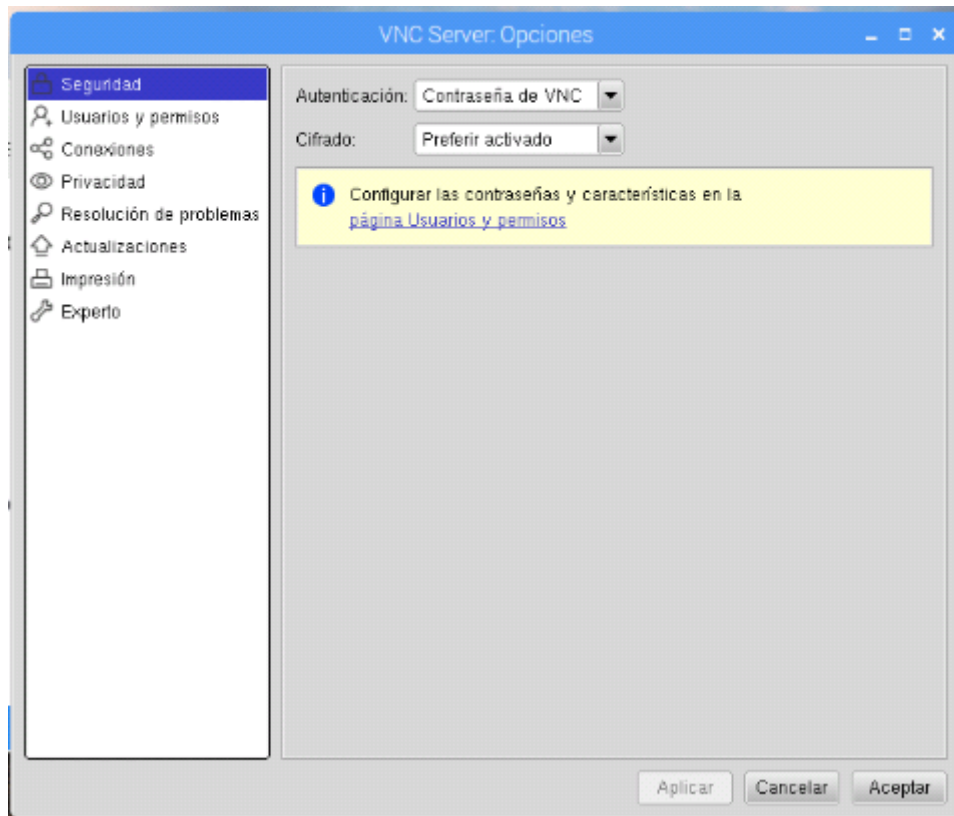
16 Configuration to enable VNC in Raspberry Pi

In order for the VNC viewer to connect to the enabled VNC server in the Raspberry Pi, it first needs to know its IP address. The IP address can be located by running `ip addr` on the Raspberry Pi and then checking the `inet` parameter in the appropriate section (eth0 for Ethernet connections, wlan0 for LAN connections, usb0 for USB).

```
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP g
roup default qlen 1000
    link/ether b8:27:eb:4d:87:8c brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.131/24 brd 192.168.1.255 scope global wlan0
        valid_lft forever preferred_lft forever
    inet6 fe80::7ce5:9e93:80ec:5442/64 scope link
        valid_lft forever preferred_lft forever
```

17 Locating the Raspberry Pi's IP address via "ip addr"

Other than the IP address, the VNC viewer will require a password to successfully connect to the Raspberry Pi. For VNC Connect, a password can be specified under the Menu > Options > Security section, choosing "VNC password" from the Authentication dropdown menu. Depending on the VNC viewer, it might also be necessary to change the encryption option to a lower security setting to avoid conflicts.



#### 18 Security configuration for the VNC server

This method requires both the Raspberry Pi and the VNC viewer to be connected to the Internet. Connecting a Raspberry Pi to the Internet without a keyboard or a mouse can be tricky, so instead, it is advised to have the phone or tablet act as a Wi-Fi Hotspot for the Raspberry Pi to connect to. This only requires a conventional USB cable to connect the Raspberry Pi to the device that will act as the VNC viewer. The IP address to be supplied to the VNC viewer in this case is the one that corresponds to *usb0*, which can also be found by running *ip addr*.

By default, the Raspberry Pi gets assigned a dynamic (random) IP address. This can be an issue, as the VNC viewer cannot connect if it doesn't know the IP address of the VNC server, and finding the Raspberry Pi's IP address requires a screen, which would defeat the purpose of using VNC in the first place. A simple solution is to assign a static IP address to the Raspberry Pi. This requires editing the *dhcpcd.conf* file<sup>18</sup>:

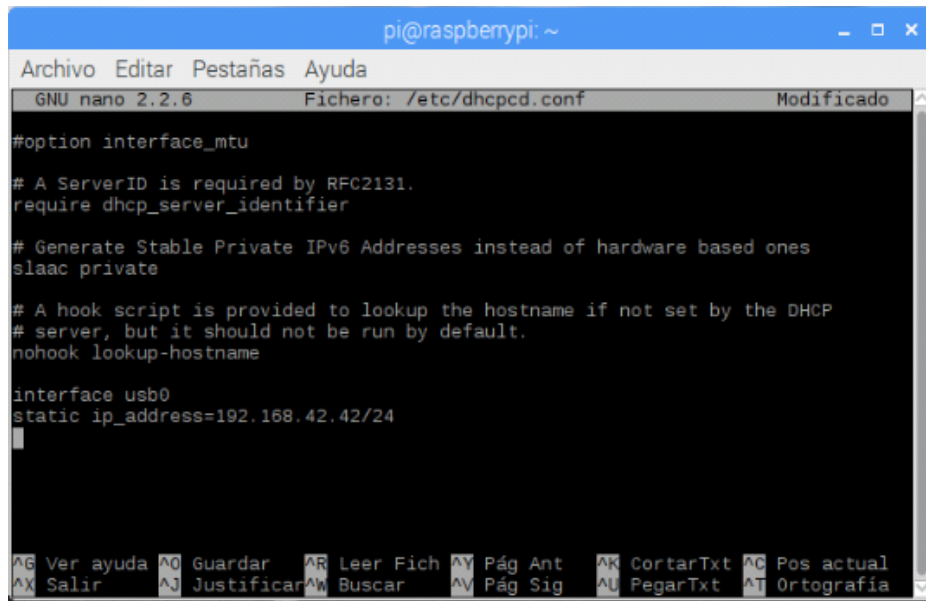
```
sudo nano /etc/dhcpcd.conf
```

The following lines should be added to the bottom of the file:

```
interface usb0
static ip_address=192.168.42.42/24
```

---

<sup>18</sup> <https://www.modmypi.com/blog/how-to-give-your-raspberry-pi-a-static-ip-address-update>



```
pi@raspberrypi: ~  
Archivo Editar Pestañas Ayuda  
GNU nano 2.2.6 Fichero: /etc/dhcpd.conf Modificado  
#option interface_mtu  
# A ServerID is required by RFC2131.  
require dhcp_server_identifier  
# Generate Stable Private IPv6 Addresses instead of hardware based ones  
slaac private  
# A hook script is provided to lookup the hostname if not set by the DHCP  
# server, but it should not be run by default.  
nohook lookup-hostname  
interface usb0  
static ip_address=192.168.42.42/24  
^G Ver ayuda ^O Guardar ^R Leer Fich ^Y Pág Ant ^K CortarTxt ^C Pos actual  
^X Salir ^J Justificar ^B Buscar ^V Pág Sig ^U PegarTxt ^T Ortografía
```

#### 19 Configuration of the dhcpd.conf file

*Interface* refers to the type of connection (such as usb0 for USB or wlan0 for LAN). The *ip\_address* will be the new static IP that will be assigned to the Raspberry Pi (the /24 at the end is necessary to specify a netmask of 255.255.255.0).

Note that it is necessary to run “*sudo service dhcpd restart*” or reboot the Raspberry Pi for these changes to take effect. After this, it should be possible to control the Raspberry Pi from a phone or tablet even in absence of a keyboard or screen. This configuration requires only a power supply for the Raspberry Pi, a USB cable to connect it to the viewer device, and permission for this device to connect to the network.

From here, it should be easy to control the Raspberry Pi via phone or tablet to connect it to the enterprise network, and then run the appropriate scans.

## 4.4 Graphical User Interface

By default, the code will send its output to the terminal in real time, informing the user about the progress of the scan. However, if the Raspberry Pi is going to be controlled via VNC as suggested in the previous section, the small size of the text on the terminal might become an issue. In order to solve this, a simple Graphical User Interface (GUI) is included, created using Pygame.

Pygame<sup>19</sup> is a Python library conceived for making multimedia applications like games, art or music. Since this library was developed with visual applications in mind, it can be used to easily make graphical interfaces.

The GUI itself will be simple and intuitive to use. Due to the lack of customization of the scanning program, the main menu will consist simply of two buttons to start the scan or quit

<sup>19</sup> <https://www.pygame.org/wiki/about>

the GUI. The colors of the buttons will brighten as the cursor hovers over them, to signal they are clickable.

Web Application Scanner

# Web Application Scanner

Start scan

Quit

## 20 GUI main screen

Hitting the “Start scan” button will call the scan code. While the scan is running, any output from the scan will be redirected to the GUI to be shown on the screen. Logging will still work as intended. The size of the text is dynamically created, so that if the text is too big to fit the screen, it will be shrunk first.

Web Application Scanner

```
Spider progress: 96%  
Spider completed  
Scanning http://127.0.0.1/  
Scan progress: 0%  
Scan progress: 3%  
Scan progress: 4%  
Scan progress: 10%  
Scan progress: 25%  
> Scan progress: 28% <
```

## 21 Sample scan shown on the GUI

Once the scan is over and the report has been generated, the GUI will return to the main menu.





## 5 Conclusions

Information security has been an important topic of debate since the inception of technology, and will remain an important topic in the foreseeable future. The risks associated with improperly secured systems range from economic harm to information theft, and the advent of the Internet has only increased this risk and its potential effects to unprecedented levels. Now more than ever, the importance of information security cannot be overstated.

In the particular case of enterprises, vulnerable systems might result in heavy financial damage and loss of reputation associated with the leaking of private information, which in some cases might even have legal repercussions. For blooming small and medium-sized enterprises this kind of mistake could prove fatal, and would severely hinder their possibilities of growing and becoming competitive in an ever-changing economy.

While this project has focused on vulnerabilities related to web applications in particular, that's not to say other aspects of information security aren't as important or should be disregarded. However, the fact that web applications are constantly exposed to the Internet make them an attractive target for attackers.

As mentioned in previous sections, statistics show that enterprises spend less in web application testing compared to other forms of computer security. This might denote an insufficient awareness of the importance of secure web applications, and information security in general.

### 5.1 Recommendations

The web application scanner developed for this project should not be seen as a substitute for a proper penetration test from a professional. This scanner will fail to detect many kinds of vulnerabilities, some because the program is fairly simple, and some because certain types of vulnerabilities cannot be found by automated means at all. It can be seen as an inexpensive way to test the waters, but the only way to ensure a web application is secure is to actually contact a professional pen-tester.

It should also be noted that full invulnerability is unachievable, as there will always be new vectors of attack and unforeseeable circumstances. Still, good practices can be adopted to minimize risks, such as:

- Having a security plan detailing how to spend the security budget and what individuals are responsible for what is highly recommended. Vulnerabilities should be prioritized according to their potential for harm and ease of exploitation. If an enterprise has multiple web applications, they should also be prioritized accordingly.
- Cookies should be used safely. They should never store sensitive information, and should be encrypted if possible.

- URL encryption can help prevent a variety of different attacks, such as Cross-Site Request Forgery or path traversal.
- Data validation can help prevent some of the most widespread forms of attack such as XSS or injection.
- Usage of HTTPS over HTTP is recommended.

While these steps might be costly and might result in a more complex web infrastructure, the resulting web application will be less liable to being affected by an attack. All in all, while there is no easy solution to the issue of web application security, regular audits can prove to be a cost-effective method to keep possible risks under control.

## 6 Appendix

### 6.1 Introducción (Español)

#### 6.1.1 Resumen

El objetivo de este proyecto es desarrollar un escáner de vulnerabilidades para aplicaciones web que pueda, de manera automática y con mínima intervención del usuario, escanear una aplicación web y generar un informe detallando las vulnerabilidades encontradas así como recomendaciones generales para mitigarlas.

El programa será tan minimalista y sencillo de utilizar como sea posible, de tal forma que una persona que carezca de conocimientos técnicos pudiera utilizarlo.

El código fuente puede encontrarse en el siguiente repositorio:

<https://github.com/svm92/tfg>

La plantilla utilizada para los informes puede encontrarse en el siguiente repositorio:

[https://github.com/svm92/test\\_report](https://github.com/svm92/test_report)

#### 6.1.2 Motivación

Una prueba de penetración, o pen-test, es un ataque controlado a un sistema con el propósito de encontrar vulnerabilidades de la manera que lo haría un atacante, para así poder neutralizarlas. Aunque es un método efectivo para encontrar fallos de seguridad, muchas compañías evitan utilizarlo debido a su alto coste, que oscila aproximadamente entre los 3500€ y 18000€[1].

Estudios exponen que casi la mitad de los fallos de seguridad encontrados en 2015 superaban los 450000€ en pérdidas. Una encuesta enviada a profesionales de la seguridad revela que el 24% de los encuestados había sufrido un filtrado de datos en el último año, y sin embargo el 8% de los mismos nunca realiza pruebas de penetración[2].

Por tanto, si bien las pruebas de penetración son costosas, las pérdidas provocadas debido a aplicaciones web inseguras son mucho más graves. El relativamente escaso presupuesto que se invierte en testeo de aplicaciones web comparado con otros tipos de prácticas de seguridad podría denotar una insuficiente concienciación acerca de la severidad de los fallos de seguridad relacionados con aplicaciones web.

El propósito de este proyecto no es tanto crear un escáner de vulnerabilidades exhaustivo de alta gama, sino concienciar acerca de la importancia de las pruebas de seguridad. Si una herramienta tan sencilla y barata es capaz de detectar vulnerabilidades en una aplicación web, un atacante habilidoso con herramientas sofisticadas podría ponerlo en peligro con mucha mayor facilidad.

### 6.1.3 Objetivos

Este proyecto automatizará una auditoría de seguridad sencilla y generará un informe, con la ayuda del escáner de vulnerabilidades OWASP ZAP. El dispositivo probará diferentes ataques o “payloads” de manera automatizada para buscar vulnerabilidades en una aplicación web objetivo. Las aplicaciones web son susceptibles a diversos tipos de ataques, como XSS (cross-site scripting) o inyección SQL, y esto las convierte en objetivos comunes para los atacantes. Los resultados se guardarán en un documento de formato portátil (pdf) para su posterior análisis.

Si bien los resultados no serán comparables a aquellos obtenidos mediante un pen-test manual de un profesional, debería al menos ser capaz de encontrar vulnerabilidades comunes, y por un bajo precio. Si una herramienta tan simple y económica es capaz de poner en peligro un sistema, entonces debería resultar inmediatamente evidente que cualquier atacante malicioso también sería capaz de explotar esas vulnerabilidades, y tal sistema debería considerarse inseguro y poco fiable.

Además del informe, el programa ofrecerá una serie de recomendaciones generales para proteger una infraestructura pyme típica contra el tipo de ataques que realiza el programa, basándose en los tipos de vulnerabilidades encontradas en el objetivo.

Este tipo de dispositivo no está ideado para resolver los problemas encontrados, sino meramente informar sobre los mismos. De esta forma, está diseñado de tal manera que cualquier comercial debería poder utilizarlo sin requerir conocimientos técnicos previos.

Este tipo de escaneo de vulnerabilidades sencillo podría usarse para alentar a las compañías a contratar los servicios de un pen-tester profesional que pudiera remediar los problemas encontrados, así como otros problemas que un escaneo automatizado sea incapaz de encontrar, para así evitar daños financieros o el robo de datos confidenciales.

## 6.2 Conclusiones (Español)

La seguridad de la información ha sido un importante tema de debate desde el nacimiento de la tecnología, y seguirá siendo un tema importante en un futuro previsible. Los riesgos asociados con sistemas impropriadamente mantenidos incluyen daños económicos y robos de información, y la llegada de Internet no ha hecho sino aumentar este riesgo y sus efectos potenciales a niveles nunca antes vistos. Ahora más que nunca, no debe desestimarse la importancia de la seguridad de la información.

En el caso particular de las empresas, un sistema vulnerable podría resultar en graves daños financieros y la pérdida de reputación asociada con el filtrado de información privada, que en algunos casos podría incluso tener repercusiones legales. Para pequeñas y medianas empresas en auge, este tipo de error podría resultar fatal, y podría herir gravemente sus posibilidades de crecer y volverse competitivas en una economía cambiante.

Si bien este proyecto se ha centrado en vulnerabilidades relacionadas con aplicaciones web en particular, esto no quiere decir que otros aspectos de la seguridad de la información no sean igual de importantes o deban ser desestimados. Sin embargo, el hecho de que las aplicaciones web estén constantemente expuestas a Internet las convierte en un objetivo atractivo para atacantes.

Como se mencionó en secciones anteriores, las estadísticas muestran que las empresas gastan menos en pruebas de aplicaciones web en comparación con otras formas de seguridad informática. Esto podría indicar una insuficiente concienciación acerca de la importancia de las aplicaciones web seguras, y de la seguridad de la información en general.

### 6.2.1 Recomendaciones

El escáner de aplicaciones web desarrollado para este proyecto no debería verse como un sustituto de una prueba de penetración realizada por un profesional. Este escáner no puede detectar muchos tipos de vulnerabilidades, algunas debido a que el programa es bastante simple, y otras debido a que ciertos tipos de vulnerabilidades sencillamente no pueden ser detectadas mediante métodos automáticos. Puede verse como una forma económica de tantear el terreno, pero la única forma de certificar que una aplicación web es segura es contactando a un pen-tester profesional.

También cabría destacar que la invulnerabilidad completa es inalcanzable, pues siempre existirán nuevos vectores de ataque y circunstancias imprevistas. Aun así, pueden adoptarse buenas prácticas para minimizar riesgos, tales como:

- Es altamente recomendado tener un plan de seguridad detallando como emplear el presupuesto de seguridad y que individuos deberían ser responsables de que. Deberían fijarse prioridades respecto a las vulnerabilidades en función de su daño potencial y la facilidad de su explotación. Si una empresa cuenta con múltiples aplicaciones web, deberían ordenarse según su prioridad de una manera similar.
- Las cookies deberían usarse de manera segura. Nunca deberían guardar información confidencial, y deberían encriptarse siempre que fuera posible.
- Encriptar la URL puede ayudar a prevenir una variedad de diferentes ataques, como CSRF o escalado de directorios.
- La validación de datos puede ayudar a prevenir algunas de las formas de ataque más extendidas como XSS o inyección.
- Se recomienda el uso de HTTPS en lugar de HTTP.

Aunque estos pasos pueden ser costosos y pueden resultar en una infraestructura web más complicada, la aplicación web resultante será menos propensa a ser afectada por un ataque. Todo considerado, si bien no existe una solución sencilla al problema de la seguridad de las aplicaciones web, las auditorías periódicas pueden demostrar ser un método rentable para mantener los posibles riesgos bajo control.

## 6.3 Sample Scan Report

# AUTOMATED TEST REPORT

Table of Contents

Introduction	1.1
Executive Summary	1.2
Summary of Results	1.2.1
Attack Narrative	1.3
Risk Rating	1.4
Recommendations	1.5
Appendix: Methodology	1.6



## Test Report

This report is the result of an automated scan performed with OWASP ZAP on <http://127.0.0.1/>, with the purpose of finding out whether the web application was vulnerable. The scan was performed with the help of OWASP ZAP 2.6.0. The report is formatted according to the NIST 800-115 guideline.

## Executive Summary

The purpose of this automated test was to confirm whether the target web application meets the most basic security requirements. The test was meant to simulate a malicious attacker exploiting common vulnerabilities and breaches in a web application. The attacks were conducted under the level of access and privilege that a general Internet user would have.

## Summary of Results

The automated scan found a total of 23 distinct vulnerabilities. The following types of vulnerabilities were found:

### Cookie No HttpOnly Flag

A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.

This vulnerability was found in the following URL(s):

<http://127.0.0.1/>

### Web Browser XSS Protection Not Enabled

Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server

This vulnerability was found in the following URL(s):

<http://127.0.0.1/sitemap.xml>

<http://127.0.0.1/passwords/>

<http://127.0.0.1/config.inc>

<http://127.0.0.1/>

<http://127.0.0.1/javascript/>

<http://127.0.0.1/classes/>

### Application Error Disclosure

This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.

This vulnerability was found in the following URL(s):

<http://127.0.0.1/passwords/>

<http://127.0.0.1/javascript/>

<http://127.0.0.1/classes/>

### X-Content-Type-Options Header Missing

The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

This vulnerability was found in the following URL(s):

<http://127.0.0.1/robots.txt>

<http://127.0.0.1/passwords/>

<http://127.0.0.1/javascript/>

<http://127.0.0.1/classes/>

<http://127.0.0.1/>

## X-Frame-Options Header Not Set

X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks.

This vulnerability was found in the following URL(s):

<http://127.0.0.1/passwords/><http://127.0.0.1/javascript/><http://127.0.0.1/classes/><http://127.0.0.1/>

## Directory Browsing

It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files , backup source files etc which can be accessed to read sensitive information.

This vulnerability was found in the following URL(s):

<http://127.0.0.1/passwords/><http://127.0.0.1/javascript/><http://127.0.0.1/classes/>

## Attack Narrative

The scan was automatically executed with the help of OWASP ZAP 2.6.0. The scan consisted of a first stage of directory browsing (spidering) for a maximum number of 5 children directories. The second stage was an active scan that attempted to execute the following attacks in the target web application:

Attack	Description
<b>Buffer Overflow</b>	In the case of a buffer overflow, the extra data might get written into a sensitive memory location. To check for this flaw, the program sends large strings of input text and checks for anomalous behavior.
<b>Client Browser Cache</b>	Checks the headers of secure pages to ensure they don't allow browsers to cache the page.
<b>CRLF Injection</b>	CRLF refers to the Carriage Return and Line Feed special characters used to denote the end of a line. It might be abused by an attacker to split a HTTP response, which can lead to cross-site scripting or cache poisoning. The program tests for this vulnerability by trying to inject a new header into the response by sending a parameter preceded by CRLF characters.
<b>Cross Site Scripting (Reflected)</b>	First, it submits a 'safe' value and checks the response for any instances of this value. Then, for any locations where this value was found in, the program attempts a series of attacks aimed at that location to confirm whether a XSS attack is possible.
<b>Cross Site Scripting (Persistent)</b>	It begins by submitting a 'safe' value much like the reflected variant, but it then spiders the whole application to find any locations where the value might be found and tries to attack them.
<b>Directory Browsing</b>	Tries to gain access to a directory listing by examining the response for patterns commonly used with web server software such as Apache or IIS.
<b>External Redirect</b>	Tries to force a redirect by sending specific parameter values to the URL, and then checks the headers and response to check whether a redirect occurred.
<b>Format String Error</b>	Sends strings of input text that compiled C code would interpret as formatting parameters, and then looks for any anomalies.
<b>Parameter Tempering</b>	Submits requests with parameter values known to cause errors, and compares the response against patterns found in common errors in Java, Microsoft VBScript, PHP and others.
<b>Path Traversal</b>	Tries to access files and directories outside of the web document root using combinations of pathname prefixes and names of local files typically found in Windows or *NIX systems.
<b>Remote File Inclusion</b>	Passes an external URL as a parameter value for the current URL and checks whether this causes the title of the page to change.
<b>Remote OS Command Injection</b>	Submits *NIX and Windows OS commands as URL parameter values that would produce a specific output should the application be vulnerable. If this output isn't found in the response, then it tries a more complex approach that involves blind injection by submitting sleep instructions and timing the responses.
<b>Server Side Code Injection</b>	Submits PHP and ASP attack strings as URL parameter values and then examines the response.
<b>Server Side Include</b>	Detects the OS the server is running on, and then sends a HTML SSI directive. The response is then analyzed to check whether it matches with a pattern indicating the SSI (Server Side Include) was successful.
<b>SQL Injection</b>	Attacks URL parameters and form parameters with valid and invalid SQL syntax, using diverse SQL injection techniques such as error, Boolean or Union based injection.

## Risk Rating

Vulnerabilities are rated according to their prevalence and their potential impact, as well as the difficulty of the exploit. The total number of vulnerabilities found for each risk level is:

- Low risk: 13
- Medium risk: 10
- Hight risk: 0

## Recommendations

Given the vulnerabilities found, some possible recommendations to remediate each of them include:

### Cookie No HttpOnly Flag

Ensure that the HttpOnly flag is set for all cookies.

### Web Browser XSS Protection Not Enabled

Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.

### Application Error Disclosure

Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.

### X-Content-Type-Options Header Missing

Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

### X-Frame-Options Header Not Set

Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).

### Directory Browsing

Disable directory browsing. If this is required, make sure the listed files does not induce risks.

## Appendix: Methodology

It should be noted that this test was performed in an automated manner using tools such as OWASP ZAP, with no direct human intervention. This makes the test non-exhaustive, and it should be assumed that the target web application has many more vulnerabilities and security breaches than the ones mentioned in this report. This test should be taken as a 'lower floor' for possible vulnerabilities. If this automated test is able to find vulnerabilities in a web application, a resourceful human should be able to find many more, most of which would escape the scope of this test. If the tested web application is found to be vulnerable, it is recommended to contact a professional penetration tester to find and mitigate these and other security breaches.



# 7 Bibliography

- [1] G. Glover, 'How Much Does a Pentest Cost?', *SecurityMetrics*, Apr-2015. [Online]. Available: <http://blog.securitymetrics.com/2015/04/penetration-test-cost.html>. [Accessed: 13-Jun-2017].
- [2] WhiteHat Security, 'Web Applications Security Statistics Report 2015', *WhiteHat Security, Inc*, 2015. [Online]. Available: <https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf>.
- [3] R. Brown, 'Computer security threats: A brief history', *Dell*, Aug-2014. [Online]. Available: <https://blog.dell.com/en-us/computer-security-threats-a-brief-history/>. [Accessed: 13-Jun-2017].
- [4] R. Delgado, 'Digital Dangers: A Brief History of Computer Security Threats', *Information Security Buzz*, Sep-2014. [Online]. Available: <http://www.informationsecuritybuzz.com/articles/digital-dangers-brief-history-computer-security-threats/>. [Accessed: 13-Jun-2017].
- [5] T. Julian, 'Defining Moments in the History of Cyber-Security', *Infosecurity Magazine*, Dec-2014. [Online]. Available: <https://www.infosecurity-magazine.com/opinions/the-history-of-cybersecurity/>. [Accessed: 13-Jun-2017].
- [6] G. Robbins, 'The Internet of Things: How Vulnerable Is It?', *The San Diego Union-Tribune*, Oct-2016. [Online]. Available: <http://www.govtech.com/fs/The-Internet-of-Things-How-Vulnerable-Is-It.html>. [Accessed: 13-Jun-2017].
- [7] M. McKeay and W. Nather, 'Akamai's State of the Internet Security Report (Q1 2017)', 2017.
- [8] M. Rouse, 'What is social engineering?', *SearchSecurity*, Feb-2016. [Online]. Available: <http://searchsecurity.techtarget.com/definition/social-engineering>. [Accessed: 05-Apr-2017].
- [9] T. Bradley, 'Introduction to Port Scanning in Network Security', *Lifewire*, Feb-2016. [Online]. Available: <https://www.lifewire.com/introduction-to-port-scanning-2486802>. [Accessed: 25-May-2017].
- [10] B. Kelebek, 'Automating security tests using OWASP ZAP and Jenkins', *Securify*, Mar-2015. [Online]. Available: [https://www.securify.nl/blog/SFY20150303/automating\\_security\\_tests\\_using\\_owasp\\_zap\\_and\\_jenkins.html](https://www.securify.nl/blog/SFY20150303/automating_security_tests_using_owasp_zap_and_jenkins.html). [Accessed: 05-Apr-2017].
- [11] K. Scarfone, M. Souppaya, A. Cody, and A. Orebaugh, 'Technical Guide to Information Security Testing and Assessment', *National Institute of Standards and Technology*, Sep-2008. [Online]. Available: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf>.
- [12] Offensive Security, 'Penetration Test Report', Aug-2013. [Online]. Available: <https://www.offensive-security.com/reports/sample-penetration-testing-report.pdf>.
- [13] B. Hill, 'Quickstart Guide to Using the Jinja2 Template Engine', 2012. [Online]. Available: <http://kagerato.net/articles/software/libraries/jinja-quickstart.html>. [Accessed: 05-Apr-2017].