Cybersecurity Test Automation: Experiences with Robot Framework and OWASP ZAP Technologies

Diogo Filipe Afonso Fernandes

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Alberto Manuel Rodrigues da Silva
Prof. Ana Cristina Ramada Paiva

Examination Committee

Chairperson: Prof. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Alberto Manuel Rodrigues da Silva
Member of the Committee: Prof. João Fernando Peixoto Ferreira

November 2022

Abstract

Web applications had a massive growth over the last decades, increasing the risk and the data sensitive exposure to attacks that are also evolving and becoming more sophisticated. Web applications shall be tested repeatedly to give their stakeholders confidence about security exposure. However, to have better coverage of applications is important to include a process that would allow reviewing the implementations from the early until the deployment phase. ITLingo initiative, with the RSL language, helps with the requirements specifications and tests and already integrates with Robot Framework, a RPA tool that produces reusable and reproducible test scripts, enabling testing automation.

The present work aims to explore the contribution of the Robot Framework with the OWASP ZAP, an open-source web vulnerability scanner. Using both technologies combined we can achieve an automated approach to an automated web vulnerability scanning approach that can be reusable and facilitates organizations to have an end-to-end solution to test their web applications against common vulnerabilities.

Keywords: RPA, Web Application, Security Testing, Vulnerability Scanners, OWASP ZAP, Robot Framework

Resumo

As aplicações Web têm tido um crescimento massivo ao longo das últimas décadas, e com isso também tem aumentado o risco e a exposição de dados sensíveis a ataques que também estão a evoluir e a tornar-se cada vez mais sofisticados. As aplicações Web devem ser testadas repetidamente a fim de se ter uma maior confiança na sua exposição e postura face à segurança. Para uma melhor cobertura de uma aplicação é importante ter um processo que permita rever o desenvolvimento desde as fases iniciais até à sua implementação. A iniciativa ITLingo, com a linguagem RSL, ajuda nas especificações de requisitos e testes, e já integra com o Robot Framework, uma ferramenta RPA que produz casos de teste reutilizáveis e reprodutíveis, permitindo a automatização de dos mesmos.

O presente trabalho visa alargar a integração da RSL e do Robot Framework com o OWASP ZAP, um scanner de vulnerabilidade web de código aberto, de modo a ter uma abordagem automatizada dos requisitos aos testes de segurança, ajudando as organizações a ter uma solução end-to-end para testar as suas aplicações web contra vulnerabilidades comuns.

Palavras-chave: RPA, Aplicações Web, Testes de Segurança, Scanners de Vulnerabilidades, OWASP ZAP, Robot Framework

Acknowledgments

First of all, I'm very grateful to my professor advisor Alberto Silva and Ana Paiva for all the patience, guidance, and support during this past year and for their overall availability to aid in all my difficulties and setbacks during the development of the RoboSecZap library.

I'm also very grateful to my friends and colleagues at Instituto Superior Técnico, for all the fruitful discussions, comments, and brainstorms that considerably improved my final project.

To my parents, I am very grateful for all the unconditional support and obviously for all the financial funds which without would have made my academic path an impossibility.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ACL - Access Control List

API - Application Programming Interface

ATDD - Acceptance Test-Driven Development

CDN - Content Delivery N etworks

CI/CD - Continuous Integration/Continuous Delivery

CVE - Common Vulnerabilities and Exposures

CVSS - Common Vulnerability Scoring System

CWE - Common Weakness Enumeration

DSR - Design Science Research

FIRST - Forum of Incident Response and Security Teams

GDPR - General Data Protection Regulation

GUI - Graphic User Interface

IS - Information Systems

NIAC - National Infrastructure Advisory Council

OS - Operating Systems

PCI DSS - PCI Data Security Standard

RF - Robot Framework

RG - Research Goals

RPA - Robot Process Automation

RSL - Requirements Specification Language

SDLC - Software development life cycle

SSDLC - Secure Software development life cycle

SSRF - Server-Side Request Forgery

VPN - Virtual Private Network

VWAD - Vulnerable Web Applications Directory

WAF - Web Application Firewall

ZAP - Zed Attack Proxy

1. Introduction

This chapter introduces the general context and motivation of this project, discusses the limitations of the existing work, and presents the key research goals, the followed methodology, and the document structure.

1.1. General Context

Web Applications are everywhere in our daily lives, from social interaction to government services and banking. Millions of users are connected through different web applications, with the purpose to exchange messages, manage a business, search for information, perform financial operations, and many others [1]. Some of these web applications stores sensitive data, that is attractive for malicious actors to steal and therefore make money with that. These malicious users, also called hackers, try to exploit some weakness of web applications, commonly known as vulnerabilities, to get this sensitive data but also to cause a disruption in the service availability [2]. Sometimes these malicious actors are well succeeding in attacks, and massive data breaches caused monetary and reputational consequences to the attacked organizations [2].

A recent report "Cost of a Data Breach Report – 2020" by IBM, shows that the average cost, per personal data record, is 146 dollars [2]. However, there is no cyber attack that aims for a single record. In that report, there's also an analysis of the average data breach total cost, where a single data breach could cause 1.52 million dollars in lost business; 1.11 million dollars on detection and escalation; 0.99 million dollars in ex-post response; and 0.24 million dollars on a notification. In sum, the average data breach can cost 3.86 million dollars. It is worth mentioning that only 52% of the analyzed breaches in the report were caused by malicious attacks, costing an average of 4.27 million dollars and the four most important root causes for these malicious attacks were cloud misconfiguration (19%), compromised credentials (19%), a vulnerability in third-party software (16%) and phishing (14%). Another relevant finding present in that report is the average time to detect and sustain a data breach which the authors concluded to be estimated for 280 days, and for data breaches caused by a malicious attack the average estimated time is 315 days [2].

To reduce the number of web vulnerabilities, organizations can use web vulnerability scanners, a computer program that scans computers, networks, and applications for known security vulnerabilities, which, can help in the process of vulnerability discovery [3]. There are several web vulnerability scanners in the market however, we will focus on OWASP ZAP, an open-source web vulnerability scanner developed in Java and distributed by the OWASP community. However, the cost of vulnerability remediation can be huge when discovered late, that is why the development of an application must consider following a software development lifecycle, and apply security best practices during all the phases of such development process.

Security testing is the process of accessing and testing software to discover security risks and vulnerabilities [1]. The process of testing software is time expensive and robotic process automation (RPA) could help to increase the accuracy and coverage of these tests, however many of the vulnerabilities are hardly found in an automated form. For the RPA purpose, we

chose Robot Framework (RF) technology, an open-source test automation framework tool, used for acceptance testing and acceptance test-driven development (ATDD).

1.2. Project Context

Our research is framed in the scope of the ITLingo initiative that intends to improve the accuracy of technical documentation such as requirements and test specifications and to promote productivity through semi-automatic techniques [4]. We intend to integrate it with OWASP ZAP, an open-source web vulnerability scanner that will be used to validate the security requirements. With this integration, we intend to develop an end-to-end approach for integrating security requirements and testing specifications with testing automation. In our approach, as Figure 1.1 suggests, we want to adapt the "end-to-end approach from requirements to automated tests" defined by Maciel et al. [4], which is composed of six tasks, and we want to change task number 5 and task number 6 to have a less manual work to map all the GUI (Graphic User Interface) elements to test.

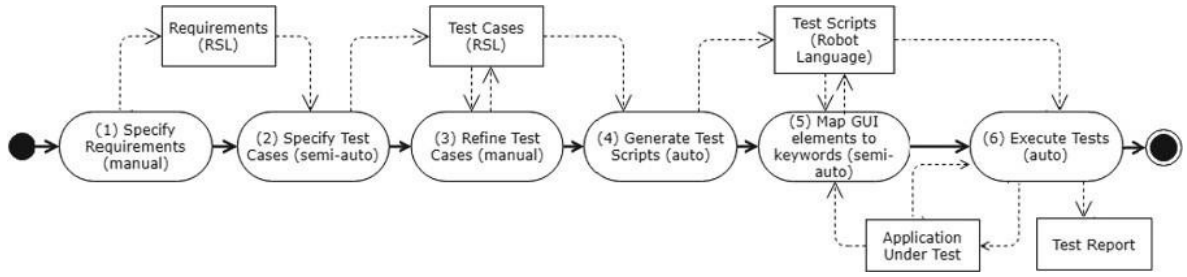

Figure 1.1 - ITLingo process approach extracted from [4]

1.3. Proposed Approach

Figure 1.2 shows the proposed process. First, we need to define the test scripts in the RF language. Second, the test scripts are executed in an automated form against the system under test. Third, the generated test reports, with the vulnerabilities found during the execution of the test, are analysed in a manual way.

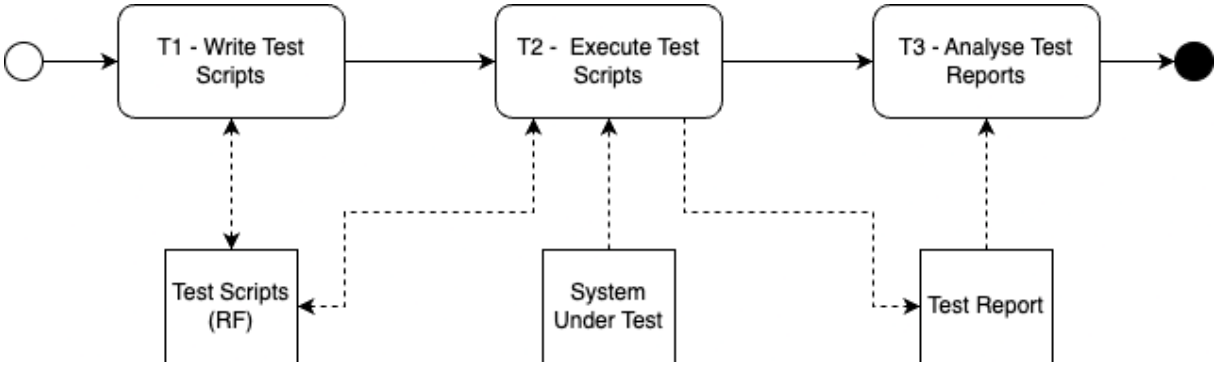

Figure 1.2 - Proposed process for automating Web Vulnerability Scanning

Our work intend to contribute to an end-to-end approach from security requirements to security tests, however the integration between the RSL (Requirements Specification Language) language and the RF test scripts was already developed in other works and therefore isn't part of our research goals.

1.4. Research Goals

This research proposes an approach to support organizations to increase security in their software projects, providing information on the active vulnerabilities and how to remediate them in their software development processes. RoboSecZapLib, a Robot Framework Library developed in Python that instrumentalizes a web vulnerability scanner called OWASP ZAP, intends to help and contribute to project members, with or without any cybersecurity knowledge, to perform a web vulnerability scanning in an automated form, using only open-source tools. This research involves analyzing and studying other initiatives to identify a solution to manage and maintain security in a software development life cycle. We shall evaluate this approach by testing this library against different types of web applications.

So, the main research goals (RG-i) of this project are:

RG-1. Define a cybersecurity testing library for the Robot Framework library, that would allow to specify and execute cybersecurity tests integrate with OWASP ZAP Vulnerability Scanner.

RG-2. Explore the integration between Robot Framework and OWASP ZAP by providing that specific library.

RG-3. Execute web vulnerability scans against different web applications to show that the approach is application-independent.

1.5. Research Methodology

This project follows the design science research (DSR) methodology, a well-established research approach in information systems (IS) and other disciplines. The DSR is an iterative methodology that combines principles, practices, and procedures [5]–[7].

Hevner et al. propose a set of guidelines for the application of DSR in the area of information systems, namely including the following aspects [5]:

Design as an Artifact: The main goal of DSR is to create viable artifacts in the form of a construct (e.g., a software application or tool), a representation (e.g., a new language or extension of a previous notation), a technique (e.g., a process or method), or an instantiation (e.g., a case study that applies such artifacts). In this research, the key artifact is the RoboSecZapLib, a Robot Framework Library that instrumentalizes OWASP ZAP.

The problem's relevance: This methodology focuses on automating web vulnerability scanning against any type of web application. As discussed, this specific project intends to automate the process of web vulnerability scanning in general. In this sense, RoboSecZapLib

contributes to accelerating and ensuring a secure software development process intuitively, according to cybersecurity guidelines.

The design evaluation: The design artifact's quality (e.g., measures in terms of utility or efficacy) is demonstrated rigorously through a well-executed evaluation method.

Research contribution: Effective DSR offers a clear and demonstrable contribution to the design artifact's application, such as design foundations and design methodologies. RoboSecZapLib contributes to guiding and accelerating the secure software development processes through analysis and management of information, supporting agile PM methodologies.

Research Rigor: The DSR depends upon rigorous methods application in the evaluation and construction of the design artifact. We evaluated our artifact based on two experiments and a user session assessment.

Design as a search process: The search for a compelling artifact depends on the available ways to reach desired outputs while the rules in the problem environment are still satisfied. We iterated several times to implement and evaluate our solution.

Communication of the results: The research presentation is effective from technology and business perspectives. The communication of the results of this research involves writing and presenting the master's thesis.

Complementary, DSR methodology recommends a process based on six tasks [5]:

Problem identification and motivation: Define the specific research problem, justify the solution's value, and motivate the researcher to investigate the answer. This task took place in May of 2021, with the study and analysis of the research's context, motivation, and background.

Define the objectives of a solution: Infer the goals of a solution for the defined problem and the knowledge of what is achievable. In this project, we propose a Robot Framework library to obtain an automated form to check for web vulnerabilities. This task took place in May of 2021.

Design and development: Create the artifacts, the core part of the project. This task includes multiple iterations with several activities. These iterations started in May 2021 and took place until June 2022.

Demonstration: Demonstrate the efficacy of the artifacts in solving the problem. In this task, we apply the solution to four experiments with different web-vulnerable applications: This task occurred between June and September of 2022.

Evaluation: Analyze and measure how well the artifacts support the problem initially defined, comparing different approaches using objective metrics. We evaluated the solution and compared it with the related work based on two experiments and a user session assessment. This task occurred between June and September of 2022.

Communication: Communicate the problem, the artifacts, and the design, considering relevance, utility, novelty, and effectiveness to researchers and other relevant audiences. This task occurred between June and October of 2022 with the writing of the MSc dissertation.

1.6. Dissertation structure

This document is structured into six chapters and one appendix:

Chapter 1 (Introduction) explains the context of this research, the motivation, the importance of a platform such as RoboSecZap, and the problems this tool solves. Then, the general research goals are established, followed by a discussion of the research methodology and document structure.

Chapter 2 (Background) presents and discusses the investigation's fundamental theoretical and technological concepts, including security testing, secure software development lifecycle, open web application secure project, mitre, common vulnerability scoring system, test automation, and robotic process automation.

Chapter 3 (Technologies Involved) presents and discusses tools related to RoboSecZap Library, namely OWASP ZAP, Robot Framework, and Robot Framework libraries.

Chapter 4 (RoboSecZap Library) describes the solution proposed, presenting its architecture, implemented requirements, and main functionalities.

Chapter 5 (Results and Discussion) discusses how the evaluation and testing of this research were conducted, namely through four experiments where we applied the RoboSecZap library against different web applications.

Chapter 6 (Conclusions) presents the main conclusion explaining the research's contributions and referring open issues and future work.

Complementary, Appendix A includes the source code of the RoboSecZap library.

2. Background

This chapter introduces and discusses general concepts and foundations underlying this research, namely security testing, secure software development lifecycle, open web application security project, MITRE, Common Vulnerability Scoring System, Test automation, and Robotic Process Automation and Vulnerability Scanners.

2.1. Security Testing

Security testing is the process that validates software system requirements related to security properties of assets that include confidentiality, integrity, availability, authentication, authorization, and non-repudiation. These security properties can be defined as follows:

Confidentiality is the assurance that information is not disclosed to unauthorized individuals, processes, or devices;

Integrity is provided when data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed;

Availability guarantees timely, reliable access to data and information services for authorized users;

Authentication is a security measure designed to establish the validity of a transmission, message, or originator, or a means of verifying an individual's authorization to receive specific categories of information;

Authorization provides access privileges granted to a user, program, or process;

Non-repudiation is the assurance that none of the partners taking part in a transaction can later deny having participated.

Security testing identifies whether the specified or intended security properties are, for a given set of assets of interest, correctly implemented. This can be done by trying to show conformance with the security properties, similar to requirements-based testing; or by trying to address known vulnerabilities, which is similar to traditional fault-based, or destructive, testing. Intuitively, conformance testing considers well-defined, expected inputs. It tests if the system satisfies the security properties concerning these well-defined expected inputs. In contrast, addressing known vulnerabilities means using malicious, non-expected input data that is likely to exploit the considered vulnerabilities.[8]

The security assessment of a system is not a single or stand-alone activity and takes place at different stages of the software development lifecycle. [1] The purpose of security testing is to find weaknesses in software implementation, configuration, or deployment. From a high-level perspective, security testing techniques can be classified as [1] :

Black Box Testing: includes the analyses of the execution of the application to search for vulnerabilities. In this approach, also known as penetration testing, the scanner does not know the internals of the web application and it uses fuzzing techniques over the web HTTP requests.[9]

White Box Testing: consists of the analysis of the source code of the web applications. This can be done manually or by using code analysis tools. The problem is that the perfect source

code analysis may be difficult and cannot find all security flaws because of the complexity of the code.[9]

Dynamic Testing: the system under test is executed and its behavior is observed;

Static Testing: the system is analyzed without executing the system under test;

Manual Testing: Manual testing includes design, business logic as well as code verification. This technique is used by hackers to hack a website or web application so that access can be gained. Some vulnerability is difficult to find using automated tools. That vulnerability can be identified by manual scan only.

Automated Testing: Automated testing is the technique that uses software that scans each page of a web application. After this scanning, a report is generated containing risks and methods to resolve them. There are a wide variety of tools that are used in vulnerability assessment and penetration testing. Web applications are scanned by various software such as Acunetix, OWASP ZAP, Nessus, etc. Many companies used these tools to scan their products. It automatically shows various weaknesses.

2.2. Secure Software Development Lifecycle

Software development life cycle or SDLC for short is a methodology for designing, building, and maintaining information from a system. It includes the stages of planning, design, build, release, maintenance, and updates as well as the replacement and retirement of the application when the need arises.[10]

The secure SDLC (SSDLC) builds on this process by incorporating security in all stages of the lifecycle. For any SDLC model that is used, information security must be integrated into the SDLC to ensure appropriate protection for the information that the system will transmit, process, and store. Applying the risk management process to system development enables organizations to balance requirements for the protection of agency information and assets with the cost of security controls and mitigation strategies throughout the SDLC. Risk management processes identify critical assets and operations, as well as systemic vulnerabilities across the organization. Risks are often shared throughout the organization and are not specific to certain system architectures.

It is a common belief that security requirements and testing inhibit the development process. However, a secure SDLC provides an effective method for breaking down security into stages during the development process. It unites stakeholders from development and security teams with a shared investment in the project, which helps to ensure that the software application is protected without being delayed. Developers may start by learning about the best secure coding frameworks and practices. They should also look into using automated tools to identify security risks within the code they write and to detect security vulnerabilities in the open-source libraries they bring into their projects. In addition, the management team may use a secure SDLC as a vehicle to implement a strategic methodology to create a secure product. For example, managers can perform a gap analysis to gain insight into which security activities or policies currently exist, and which are absent, and to see how effective they are at each stage of the SDLC. To achieve a streamlined SSDLC and ensure software shipping deadlines are not missed, it is necessary to establish and enforce security policies that help address high-level issues like compliance without requiring manual review or manual intervention. To achieve

this, some organizations choose to hire security experts to evaluate security requirements and to create a plan that will help the organization improve its security preparedness.[11]

Some of the benefits of integrating security into the system development life cycle include:

- early identification and mitigation of security vulnerabilities and problems with the configuration of systems, resulting in lower costs to implement security controls and mitigation of vulnerabilities;
- awareness of potential engineering challenges caused by mandatory security controls;
- identification of shared security services and reuse of security strategies and tools that will reduce development costs and improve the system's security posture through the application of proven methods and techniques;
- facilitation of informed executive decision-making through the application of a comprehensive risk management process promptly;
- documentation of important security decisions made during the development process to inform management about security considerations during all phases of development;
- improved organization and customer confidence to facilitate adoption and use of systems, and improved confidence in the continued investment in government systems; and
- improved systems interoperability and integration that would be difficult to achieve if security is considered separately at various system levels. [12]

2.3. Open Web Application Security Project

The OWASP[1] project is an open community dedicated to empowering organizations to develop, purchase and maintain trusted applications. All OWASP developed tools, documents, forums, and chapters are free and open to anyone interested in improving application security. The OWASP Top Ten is a list of the top security risks in web applications. The main goal is to educate those who are in charge to design, architect, and develop web applications as well as to alert organizations about the consequences and impacts of the most important weakness in application security, by identifying the most critical risks they could face.

The OWASP Top ten version was initially released in the year 2003 and had minor updates in the years 2004 and 2007. However, in the year 2010, a new version was released to redefine/prioritize the risks and to provide additional information on how to assess the risk of web applications. The most recent version was released in 2021 year and follows the same focus as the 2017 version, as we can see in figure 2.1.

In the most recent update, from OWASP Top 10 2017 to OWASP Top 10 2021, there were three new categories created, four categories of the previous version with a different naming a scoping, and also there was some consolidation in the content. The main objective of these changes was to focus on the root cause over the symptom.[13]

[1] https://owasp.org/

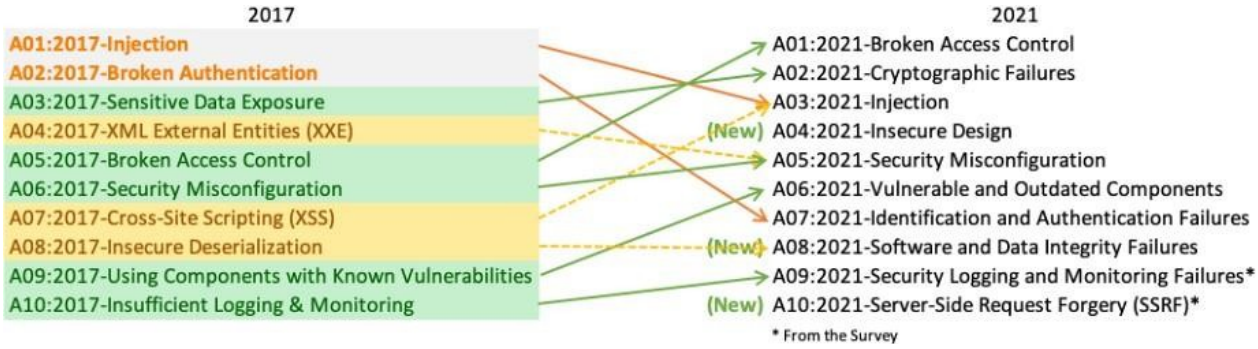

Figure 2.1 - OWASP TOP 10 Version 2017 VS Version 2021 extracted from [13]

In the next subsection, we will describe each new OWASP TOP 10 risks not only describing the risk itself but also focusing on describing the major changes that occurred. The risks described in the OWASP Top Ten (OWASP Foundation, 2021) are as follows:

A01:2021 Broken Access Control: Access control enforces a policy such that users cannot act outside of their intended permissions. This type of failure typically leads to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Thus, common access vulnerabilities include Violation of the principle of least privilege, bypassing access control checks, view or editing data from someone else's account.

A2:2021 Cryptographic Failure: Cryptographic failure can lead to the exposure of sensitive data, to prevent this type of failure, firstly we need to categorize the type of data in transit or stored to determine the protection needs. For example passwords, credit card numbers, health records, personal information, and business secrets require extra protection because they are now covered under privacy laws, such as General Data Protection Regulation (GDPR), or financial data protection such as PCI Data Security Standard (PCI DSS).

A3:2021 Injection: Injection faults, such as those that occur in SQL, operating system (OS), and LDAP. These flaws occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the parser into executing unintended commands or accessing the data without proper authorization.

A04:2021 Insecure Design: This is a new category created, with a focus on risks related to design flaws. A secure design can have implementation defects that can lead to vulnerabilities that can be exploited, an insecure design cannot be fixed by a perfect implementation by definition, thus there is a difference between design flaws and implementation defects. One of the factors that can contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed.

A05:2021 Security Misconfiguration: Good security requires having defined and implemented a secure configuration for the application, frameworks used in the development, application server, web server, database server, and operating platform. All security configurations must be defined, implemented, and maintained because they are generally not secure by default.

A06:2021 Vulnerable and Outdated Components: Some components, such as libraries, frameworks, and other software modules, are almost always running with the same privileges as the application. If a vulnerable component is exploited through an attack, it can facilitate server intrusion and serious data loss. Applications that use components with known vulnerabilities weaken the application's defenses and allow the range of possible attacks and their impacts to expand.

A07:2021 Identification and Authentication Failures: Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or exploit other implementation flaws to assume the identity of other users.

A08:2021 Software and Data Integration Failures: This is a new category that focuses on making assumptions related to software updates, critical data, and continuous integration/continuous delivery (CI/CD) pipelines without verifying integrity. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

A09:2021 Security Logging and Monitoring Failures: Without logging and monitoring, breaches cannot be detected, therefore they cannot be escalated and efficiently responded to. Insufficient logging, detection, monitoring, and active response occurs at any time: auditable events, such as logins, failed logins, and high-value transactions, are not logged; warnings and errors generate inadequate or unclear log messages.

A10:2021 Server-Side Request Forgery: This is a new category created and is related only to Server-Side Request Forgery (SSRF) flaws that occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, virtual private network (VPN), or another type of network access control list (ACL).

2.4. MITRE

The MITRE[2] corporation is a non-profit corporation that operates several research and development centers funded by the United States federal government. They provide innovative and practical solutions to some of the most critical challenges facing the U.S. nation, such as defense and intelligence, aviation, civil systems, homeland security, the judiciary, healthcare, and cybersecurity. MITRE Corporation set up three powerful public sources of cyber threat and vulnerability information, namely the Common Vulnerabilities and Exposures (CVE) list, and the Common Weakness Enumeration (CWE). [14]

The CVE list is a common naming dictionary for publicly known information security vulnerabilities, where each reference has a unique identification number, which facilitates its sharing in security databases. The list is defined and maintained by the MITRE organization with input from organizations around the world that are active in the field of information security, such as commercial tool vendors, research institutions, government agencies, and other security experts[14].

[2] https://www.mitre.org/

The CWE is a community-based list of common software and hardware weakness types. Weaknesses can be considered as flaws, faults, bugs, or other errors in the software or hardware implementation, code, design, or architecture that if left unaddressed could result in systems, networks, or hardware being vulnerable to attack. The CWE list serves as a language that can be used to identify and describe these weaknesses in terms of CWEs. The main goal of CWE is to stop vulnerabilities at the source, by educating software and hardware architects, designers, programmers, and acquirers on how to eliminate the most common mistakes before products are delivered.[15]

2.5. Common Vulnerability Scoring System

The common Vulnerability Scoring System (CVSS) was introduced by the National Infrastructure Advisory Council (NIAC) and is now managed by the Forum of Incident Response and Security Teams (FIRST). CVSS aids security managers in the prioritization of vulnerabilities by providing a metric for their relative severity.[16] The common Vulnerability Scoring System (CVSS) outputs a numerical score indicating the severity of software, hardware, and firmware vulnerabilities relative to other vulnerabilities. CVSS is composed of three metric groups as shown in figure 2.5.1 :

The base metric group represents the intrinsic characteristics of a vulnerability that are constant over time and across user environments and it's composed of two sets of metrics, the exploitability metrics that reflect the ease and technical means by which the vulnerability can be exploited, and the impact metrics that reflect the direct consequence of a successful exploit;

The temporal metric group reflects the characteristics of a vulnerability that may change over time but not across user environments;

The environmental metric group represents the characteristics of a vulnerability that are relevant and unique to a particular user's environment, these considerations include the presence of security controls that may mitigate some or all consequences of a successful attack.[17]

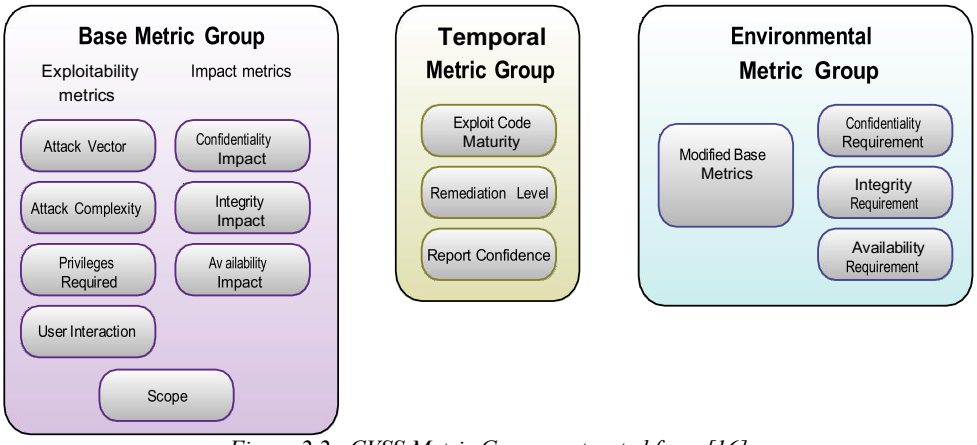

Figure 2.2 - CVSS Metric Groups extracted from [16]

2.6. Test automation and Robotic Process Automation

Robotic Process Automation (RPA) is a new approach to business process automation, allowing users to avoid doing repetitive tasks that need manual data entry and processing. RPA is a disruptive technology that enables routine activities normally performed by humans to be performed in an automated, simple, and flexible manner, making organizations more effective in business processes. [18] The main feature of a RPA is that with one application (Robot Software) we can interact with different applications through the Application Programming Interface (API) or through an existing User Interface. Simulating user actions provides some benefits, namely, there is no need to change existing IT systems to deploy the RPA infrastructure.[19] Since Robotic Process Automation can replace repetitive tasks, it significantly reduces costs and increases operational performance with a few changes in technology, particularly when open-source solutions that do not require license costs are used.

The RPA is usually applied to the following types of tasks (Figure 2.6.1) :

- regularly and frequently repeated tasks. Any automation is effective only for repetitive processes;
- tasks that have a great influence on the business. In this case, the process may not be repeated as often, however, it would be very important for the company;
- tasks that require the processing of a substantial amount of data. With increasing loads, people tend to make more and more mistakes, while the robot will continue to work stably;
- tasks that are explicitly described by the strict business rules. Robotic processes, by definition, imply the exclusion of a person from the decision-making process. Therefore, any judgment, agreement, or use of external data should be also excluded. The robot can make decisions regarding the course of processing, but the rules must be strictly defined and fixed;
- tasks that require work with at least one electronic system (external, internal).

Figure 2.3 - Business process that can be automated with RPA extracted from [18]

By automating the software testing process, technologies such as Junit, or Selenium helped in the improvement of effectiveness and efficiency. However, these technologies focus only on a specific task and system, for instance, Junit is only used for testing Java code and Selenium to

test web applications, leaving the coordination of the software testing process to humans, which can be faulty and time expensive. The application of RPA for the software testing process can enable a more efficient and effective to perform end-to-end software testing approach, from requirement analysis to test report.

2.7. Vulnerability Scanners

A Vulnerability scanner is a computer program that scans computers, networks, and applications for known vulnerabilities. To put it in another way, these scanners are utilized to find the flaws in a system. They are used to identify and discover vulnerabilities in network-based assets such as firewalls, routers, web servers, application servers, and so on that arise from misconfigurations or defective programming. [20] Following the creation of an inventory, the vulnerability scanner compares each item in the inventory to one or more databases of known vulnerabilities to see if any of the objects are vulnerable. A systems vulnerability analysis is produced as a result of such a scan, revealing any known vulnerabilities that may require threat and vulnerability management.

Vulnerability Assessment, also known as Vulnerability Testing, is a vulnerability scanning software used to assess security risks in software systems to lessen the likelihood of a security breach. It is the process of defining, identifying, classifying, and prioritizing vulnerabilities in computer systems, applications, and network infrastructures. Vulnerability assessments also provide organizations with the information, awareness, and risk backgrounds they need to recognize and respond to threats to their environment. The goal of a vulnerability assessment is to identify threats and the risks they entail. They usually entail the use of automated testing tools like network security scanners, the results of which are documented in a vulnerability assessment report.

There are several types of vulnerability scanners based on the target or system that will be tested. Bellow, we will provide a list of the types of vulnerability scanners and a brief description:

Host-Based identifies problems with the host or system, this will install a mediator program on the target machine, which will track the occurrence and alert the security analyst;

Network-Based will discover open ports and identify any unfamiliar services that are using them, it will then reveal any potential vulnerabilities linked with these services;

Database-Based will use tools and techniques to uncover security vulnerabilities in database systems and prevent SQL Injections for example;

Wireless network scans will use tools and techniques to confirm if a company's network is safely configured;

Application scans examine websites for known software flaws and inappropriate network or web application setups;

Vulnerability Scanners also provide information on Common Vulnerabilities and Exposures (CVE), which is a set of standardized names for known vulnerabilities with a risk classification system known as the Common Vulnerability Scoring System (CVSS). The National Vulnerability Database's CVSS scores include factors such as attack vector, complexity,

required privileges, user involvement, and the impact of confidentiality, integrity, and availability.

In general penetration tests is a combination of manual testing through security experts and the usage of black-box vulnerability scanners. Black-box web vulnerability scanners are a class of tools that can be used to identify security issues in applications through various techniques. The scanner queries the application's interfaces with a set of predefined attack payloads and analyses the application's responses for indicators if the attack was successful and, if this is not the case, hints on how to alter the attack in the subsequent tries.

3. Technologies Involved

RoboSecZap is developed in Python and instrumentalizes an open-source web vulnerability scanner called OWASP ZAP through a Robot Process Automation framework called Robot Framework. In this chapter, the OWASP ZAP and Robot Framework will be explained.

3.1. OWASP ZAP

The OWASP Zed Attack Proxy (ZAP) is an open-source penetration testing tool, easy to use and designed to be used by people with a wide range of security experience, ideal for developers and functional testers that don't have an experienced background with penetration testing activities. ZAP is designed for testing web applications, providing automated scanners as well as a set of tools that can help on a manual penetration test to find vulnerabilities. It is a cross-platform software, that can be deployed in windows or UNIX systems, requiring only a Java pre-installation. It has a well-developed and documented API, helping developers to run and develop automated processes.

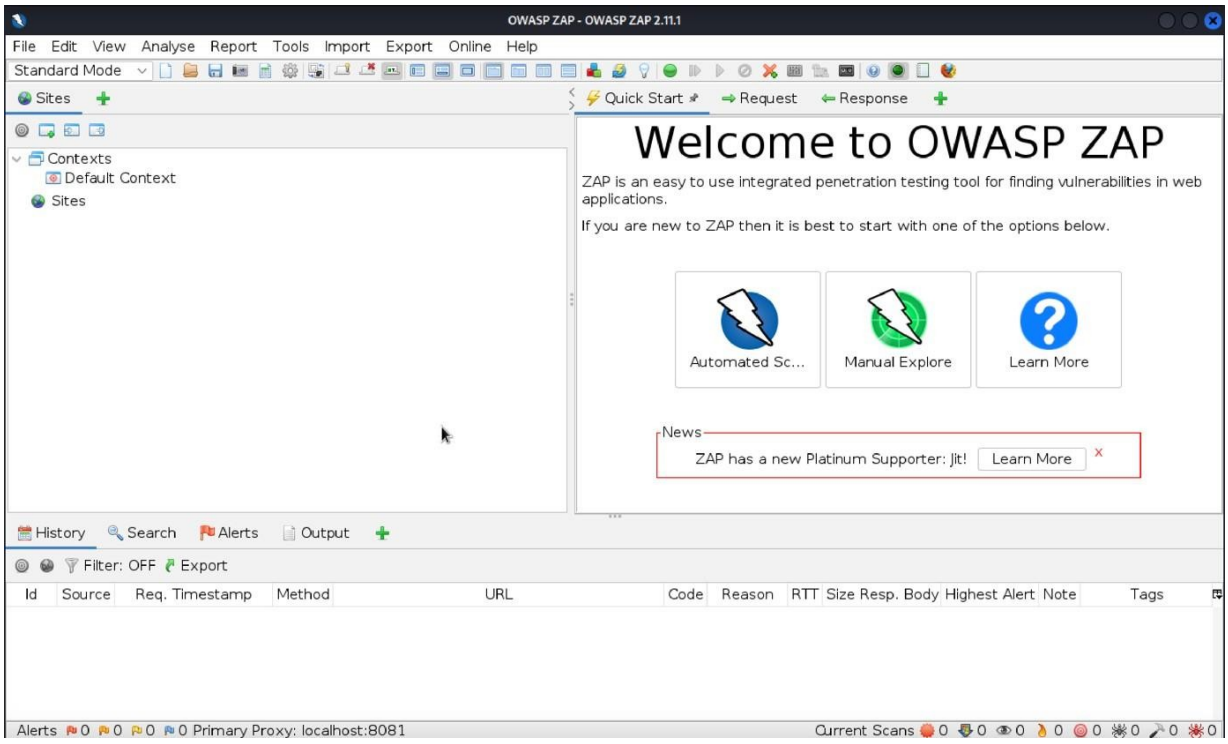

Figure 3.1 - OWASP ZAP GUI

The main OWASP ZAP functionalities are spider, ajax spider, intercepting proxy, active and passive scanner, and report generation which will be explained as follow. The spider and ajax spiders are used to crawl all the pages from a web application and are used to discover in an automated way all the resources (URLs) from a particular website. The intercepting proxy stands between the browser and the web application, so it can intercept and inspect all the messages exchanged between the browser and the web application. The passive scanner scans

all the HTTP responses and requests without modifying their content to find vulnerabilities signatures, however the active scanner attempt to find potential vulnerabilities using known attack payloads, whoever this scanner can find injection flaws, other logical vulnerabilities such as broken access control, cannot be found with this scanners, therefore a manual penetration testing should be performed in addition to a vulnerability scanning. The report generation, as the name indicates, is the mechanism to generate a report with the vulnerability found during the process of scanning, that can be exported in a JSON, PDF, or XML file[21].

3.2. Robot Framework

The Robot Framework is an open-source test automation framework tool, used for acceptance testing and acceptance test-driven development (ATDD). It has a tabular test data syntax and it uses the keyword-driven testing approach. It is developed in Python therefore it can be run on all operating systems. The tool is sponsored by Nokia Siemens Networks and released under Apache 2.0 license, meaning it is allowed to be used for free.[22], [23]

Figure 3.2 shows the modular Robot Framework architecture, that enables it to be more versatile and independent of the application to be tested. This versatility is mainly achieved by the use of several public libraries developed in Python and Java, that can be extended.

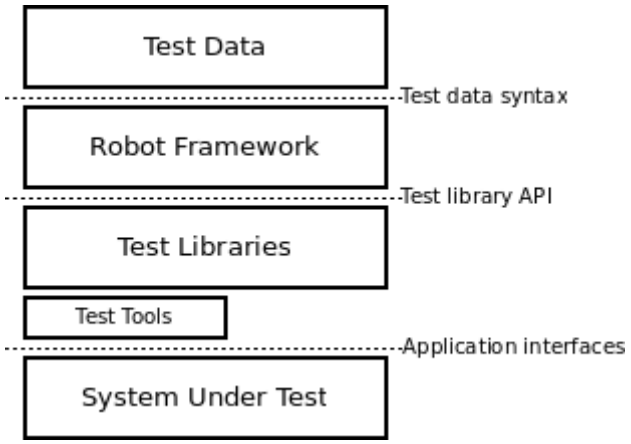

Figure 3.2 - RobotFramework Architecture extracted from [21]

Explaining its architecture in more detail: test data is a tabular form that is easy to edit, when starting the Robot Framework it will use the test data to run test cases, and generates logs and reports; the Robot Framework is the framework itself that is written in python and where we can use the commands to start the framework and then the test data file is set to execute a specific test case; test libraries are composed by two parts, a built-in library robot framework, as well as others test tools that can be extended by a extend library; system under test is the product target to be tested.[23]

3.3. Robot Framework Libraries

As mentioned Robot Framework is a generic keyword-driven test automation framework and the available keywords are defined in test libraries. There are two types of libraries: standard libraries and external libraries[24]. The standard libraries are distributed with Robot Framework and the external libraries are released in separate packages. Standard libraries include BuiltIn, Telnet, Remote, OperatingSystem, Collection, String, Dialogs, and Screenshot. External libraries include SSH, AutoIt, Database, Selenium, and Swing.

During the initial phase of our research, we identified a Robot Framework library called RoboZap[3], that already provides some preliminary integration between Robot Framework and OWASP Zap. Figure 3.3 shows a Robot Framework script[4] used with the RoboZap library and Figure 4 the command line output.

```
*** Settings ***
Documentation This is a basic test
Library RoboZap ht
tp://127.0.0.1:8080 8080


*** Variables ***
${target}         http://localhost:3000/#/

*** Test Cases ***
Start Headless Zap
     start headless zap /usr/local/zaproxy/

Open Target Url
     zap open url ${target}

Set Context for Application
     ${context_id}= zap define context vul-flask ${target}
     set suite variable ${CONTEXT} ${context_i              lozarrza
Active Scan Application
     ${scan_id}= zap start ascan ${CONTEXT} ${target}
     zap scan status ${scan_id}

ZAP Generate Report
     zap export report ${CURDIR}/report.json json Test Report

ZAP Close
     zap shutdown
```

Listing 3.3 RoboZap script code

[3] https://github.com/we45/RoboZap

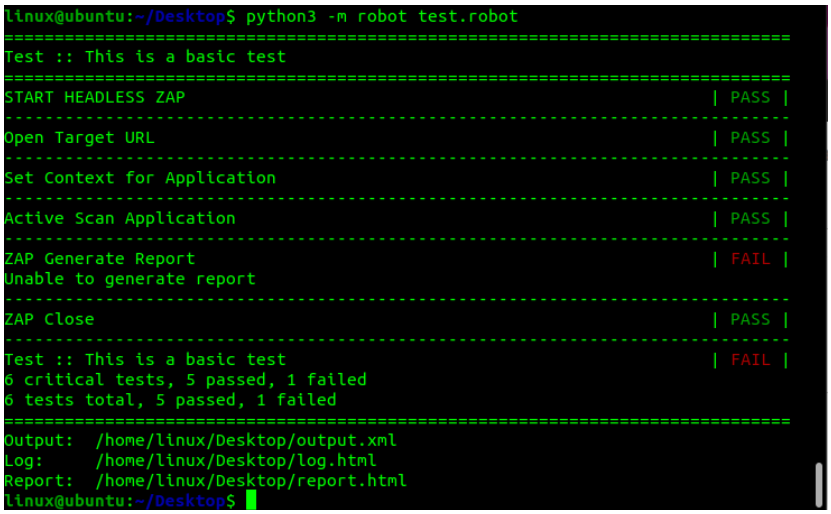

Figure 3.3 - RoboZap script command line output

The Robo Framework script, illustrated in Figure 3.3, can perform Passive and Active Scanning using the OWASP Zap Vulnerability Scanning. However, it's limited to a previous version of the OWASP Zap tool (version 2.4) and it fails to generate the report due to old API specifications, invoking the need for new development. In our work, we took this into account and update all the OWASP ZAP API calls to perform a full web vulnerability scanning.

4. RoboSecZap Architecture

This chapter presents the architecture of the proposed library. This includes the general architecture and main functionalities.

4.1. General Architecture

RoboSecZapLib is a RF library that integrates with the OWASP ZAP tool. With this library a user can make a web vulnerability scanning in an automated form without needing to have extensive knowledge within the area of cybersecurity and web vulnerability scanning. This library is prepared to scan web applications regardless of the technology they use. The outcomes of its execution are generated in report files that can be in XML, JSON, HTML, and PDF, to help developers and software testers to review and to correct several vulnerabilities that can be identified during the web vulnerability scanning process.

This library can be used in different phases of the SDLC process, whereas it can help in the early development phase or in the maintenance phase bringing confidence that after the full development of a web application, the vulnerabilities identified during the development phase are no longer active.

Figure 4.1.1 shows parallelism between the general RF architecture and the specific architecture where the RoboSecZap library and the OWASP ZAP tool are included.

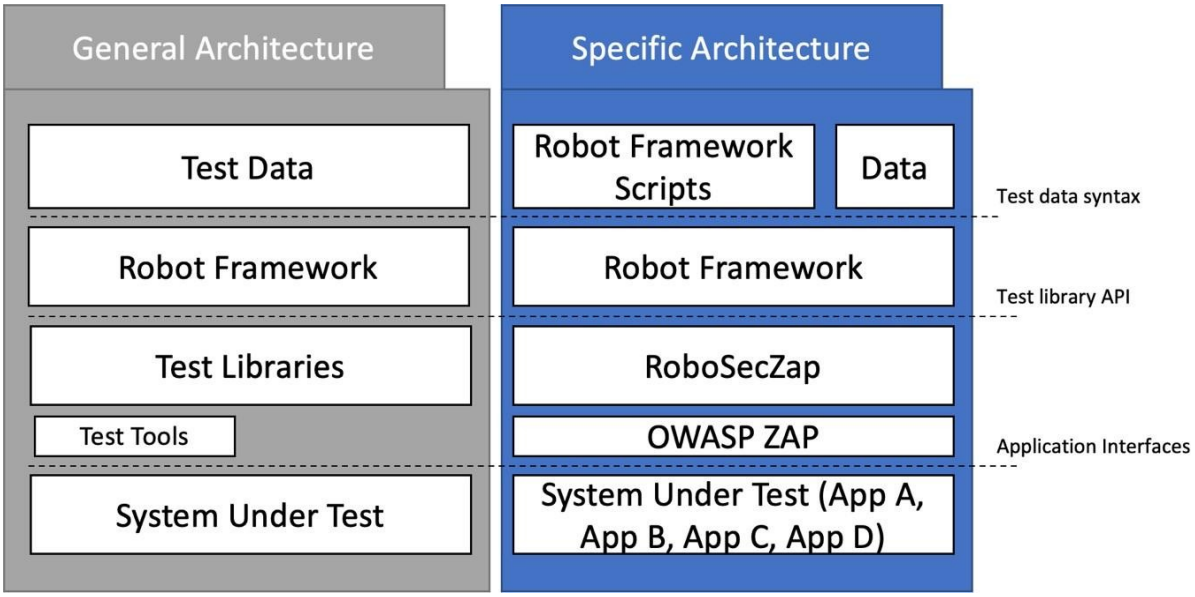

Figure 4.1 - General Vs Specific Robot Framework Architecture

This integration was possible due to OWASP ZAP API which allows to interact with ZAP programmatically. The API provides access to most of the core ZAP features such as the active scanner and spider. Not all OWASP ZAP functionalities are available throught API, however in future versions of ZAP will increase the functionality available via the API.[25]

4.2. Main Functions

The RobotSecZap library is defined as a Python class and it calls some functions available in OWASP ZAP API. Below, we introduce and explain these functions briefly, describing their inputs, outputs, and their variabilities:

Function: init_lib

Objective: This function initializes the zap object from the ZAPv2 python library, defined as global to be reused for the next functions.

Input parameters:

- apiKey: should be the same when running OWASP ZAP as a daemon (without GUI) to secure the communications between the client and the ZAP Proxy;
- appName: to define the session name for the web vulnerability scanning process.

The following listing will show how this function is defined:

```
def init_lib(self,apiKey,appName):
     global zap
     self.zap = ZAPv2(apikey=apiKey)
     self.zap.core.new_session(name=appName, overwrite=True)
```

Listing 4.1 – init_lib code

Function: spider

Objective: The spider function invokes the spider tool that is used to automatically discover the web application resources, by crawling a specificied URL. During the spidering process, the tool will identify all the hyperlinks on a specific page and adds them to a list of URLs to visit, and this process continues recursively as long as different URLs are identified. The spider scan is a async request and the time to complete the task will vary depending on the complexity of the web application. The scan ID returned via starting the spider should be used to obtain the results of the crawling. We execute the status API to get the status/percentage of work done by the Spider.

Input parameters:

- target: The web application URL to start the spidering process

The following list will show how this function is defined:

```
def spider(self,target):
     scanID = self.zap.spider.scan(target)
     while int(self.zap.spider.status(scanID)) < 100:
            continue
```

Listing 4.2 – spider code

Function: ajax_spider

Objective: The ajax spider function invokes the ajax spider tool and is supposed to use whether a web application relay on Ajax or Javascript, allowing it to crawl the web application written

in ajax in far more depth than the traditional spider. Unlike the traditional Spider, Ajax Spider does not provide a percentage for the work to be done. Use the status endpoint to identify whether the Ajax Spider is still active or finished.

Input parameters:

- target: The web application URL to start the ajax spidering process

The following listing will show how this function is defined:

```
def ajax_spider(self,target):
     scanID = self.zap.ajaxSpider.scan(target)
     while self.zap.ajaxSpider.status == 'running':
            time.sleep(2)
            continue
```

Listing 4.3 – ajax_spider code

Function: passive_scan

Objective: The Passive Scan function analyzes passively all the requests that were proxied through ZAP or initialized by the spider and ajax spider tools. We don't have to manually start the passive scan because ZAP by default passively scans all the HTTP and WebSocket messages (request and responses), however as the records are passively scanned it will take additional time to complete the full scan. After the crawling is completed use the recordsToScan API to obtain the number of records left to be scanned.

Input parameters:

- None

The following listing will show how this method is defined:

```
def passive_scan(self):
     while int(self.zap.pscan.records_to_scan) > 0:
            continue
```

Listing 4.4 – passive_scan code

Function: active_scan

Objective: Active scanning attempts to find potential vulnerabilities by using known attacks against the URLs found during a previous execution of spider and ajax spider tools, giving that this will behave like a real attack, we shouldn't use this function against an application we don't have permission to perform a vulnerability scanning. It should be noted that active scanning can only find certain types of vulnerabilities. Logical vulnerabilities, such as broken access control, will not be found by any active or automated vulnerability scanning. Manual penetration testing should always be performed in addition to active scanning to find all types of vulnerabilities.

Input parameters:

- target: The web application URL to start the active scanning process

The following listing will show how this function is defined:

```
def active_scan(self,target):
     self.zap.ascan.enable_all_scanners()
     self.zap.core.access_url(target, followredirects=True)
```

```
time.sleep(2)
scanID = self.zap.ascan.scan(target)
while int(self.zap.ascan.status(scanID)) < 100:
        time.sleep(2)
        continue
```

Listing 4.5 – active_scan code

Function: generate_report

Objective: The generate report function will invoke the ZAP API to generate different report types based on the input. In a report, each vulnerability is identified as an alert, and each alert has a description, a list of endpoints where it was identified, and a solution to correct the vulnerability, references are also included to explain better the vulnerability and how to fix it, a CWE ID, WASC ID, and Plugin ID. The report also gives the user two tables with a summary of alerts, to help for example a project owner that doesn't have any technical knowledge to review or to produce a risk analysis for the web application

Input parameters:

- type: to indicate the type of file to be outputted, for now only HTML, JSON, XML, and PDF are allowed to be indicated.
- path: to indicate in which path the report file should be stored
- filename: to indicate the filename for the report file

The following listing will show how this function is defined:

```
def generate_report(self,type,path,filename):
    type = str.lower(type)
    if type == 'html':
        report = self.zap.core.htmlreport()
    elif type == 'json':
        report = self.zap.core.jsonreport()
    elif type == 'xml':
        report = self.zap.core.xmlreport()
    elif type == 'pdf':
        report = self.zap.core.htmlreport()
        completeNameTemp = os.path.join(path, filename + '.'+"html")
        completeName = os.path.join(path, filename + '.'+type)
        file1 = open(completeNameTemp, "w")
        file1.write(report)
        file1.close()
        pdfkit.from_file(completeNameTemp,completeName)
        os.remove(completeNameTemp)
        return
    else:
        print('File extension not supported')
    completeName = os.path.join(path, filename + '.'+type)
    file1 = open(completeName, "w")
    file1.write(report)
    file1.close()
```

Listing 4.6 – generate_report code

5. Results and Discussion

During our experiment we used our developed RF library against different vulnerable web applications, from OWASP Vulnerable Web Applications Directory (VWAD), to prove that our approach can effectively identify vulnerabilities independently from the technology used in the SuT. To a better context, the OWASP VWAD project is a list maintained of known vulnerable web and mobile applications, that can be executed online or offline, and can be used by developers, security auditors, and penetration testers to practice their skills and knowledge, in the area of cyber-security. These projects can help improve multiple hacking tools and techniques to prepare for a real cybersecurity threat or incident[26].

To test our approach, expressed in Figure 1.3, we chose the following Vulnerable Web Applications as the SuT:

- In experiment 1 we choosed the Testphp Vulnweb (http://testphp.vulnweb.com/) is a e-commerce vulnerable PHP web application, provided by Acunetix, where a user can pick items and add them to a cart like an e-commerce application.
- In experiment 2 we choose the Google gruyere appsot (https://google-gruyere.appspot.com/) is a vulnerable web application developed by google engineers, with many security vulnerabilities that are organized by sections.
- In experiment 3 we choose the Aspnet testsparker (http://aspnet.testsparker.com/) is a vulnerable ASP.NET Web Application, provided by Test Sparker, and it looks like a Bitcoin Information website, where you can buy items and exchange bitcoins.
- In experiment 4 we choose Php Testsparker (http://php.testsparker.com/) is an e-commerce vulnerable PHP Web Application, also provided by Test Sparker, and on this website, you can also buy products and give reviews.

We defined the test cases following the workflow presented in Figure 5.1, where we started by initiating the library with init_lib, then we did a spidering process with spider and ajax_spider functions, after we did a scanning process using the passive and active scan, and finally we generated the reports using the generate_report function.

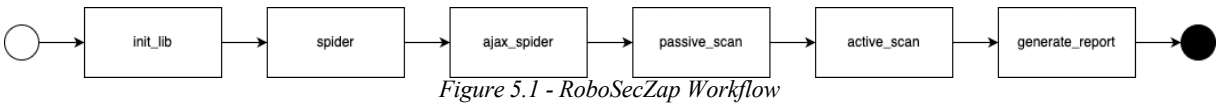

Figure 5.1 - RoboSecZap Workflow

At the end of our experiments, during the analyses of Test Reports we were able to find vulnerabilities, categorized per risk levels high, medium, low and informational. Table 1, bellow, presents the summary of the detected vulnerabilities in the tested vulnerable web applications.

Risk Level	Experiment 1	Experiment 2	Experiment 3	Experiment 4
High	2	1	4	0
Medium	4	3	4	2
Low	2	4	6	2
Informational	2	3	0	0

Table 1 - Detected Vulnerabilities Summary

The results expressed in Table 1 can prove that our approach is able to identify vulnerabilities agnostically to the technology used in the web application. Overall there were 37 vulnerabilities identified in the 4 experiments. This can help the stakeholders to have an overview of the security exposure of their applications and to prepare a vulnerabilities mitigation process, giving priority to solving high-risk vulnerabilities. The process described in figure 1.3 can also be used as many times as we want to, due to the reusability of test scripts, and it can be helpful for example to verify if a vulnerability was successfully corrected, therefore it shall not be identified again.

6. Conclusion

This dissertation proposes the usage of a RPA framework to automate web vulnerability scanning against different web applications. We chose robot framework as the RPA framework due to its versatility, extensibility, and independence from the system under test. For the web vulnerability scanning proposal we chose the OWASP ZAP tool, due to its documented API and the possibility to scan web applications independently from his technology. We could do it, developing a Robot Framework library that instrumented the OWASP ZAP tool. We could confirm that by testing against four different web applications, that were developed using different languages and technologies.

6.1. Main Contributions

The most important contributions of this research are the following:

First, we developed a RF library that integrates OWASP ZAP with the robot framework technology that gives us an automated approach to scan web applications agnostically to its technology. Second, we were able to show that our approach defined in Figure 1.3 was possible and could be repeatedly done, due to the reusability of test scripts. Third, we were able to generate reports with relevant information to verify and correct vulnerabilities found during the process. In summary, we were successful to have an approach that automates the web vulnerability scanning using Robot Framework and OWASP ZAP.

6.2. Future Work

This research identifies yet several aspects that may be addressed in future work.

Although the OWASP ZAP has a well-documented API, it yet fails to have a configuration that makes it possible to test only a specific vulnerability, however, it is expected to be possible shortly, therefore this should be a future outcome of this library.

Given that it is possible to generate JSON and XML reports, in the future these reports could be uploaded to a vulnerability management tool, to manage the correction of active vulnerabilities in a system, and of course a CI/CD process could be developed where a vulnerability mitigation process could be implemented and tested.

We tried without success to have an authenticated web vulnerability scan that could allow the spidering and the passive/active scan of authenticated pages, to find vulnerabilities in these pages, this could be achievable using a context object, thus a new research on this topic with the focus on the context object, or a different option, could led to an authenticated web vulnerability scan.

This project can also be used to test a web application firewall (WAF) configuration, where a WAF could be configured to identify if the web applications behind it could being scanned by a web vulnerability scanner and we could test if the WAF could detect and block this types of scans.

Appendix A

In this appendix we present the code for RoboSecZap, a RF library written in Python.

```
from zapv2 import ZAPv2
import pdfkit

class RobotSecZap(object):
     ROBOT_LIBRARY_SCOPE = "GLOBAL"

     def init_lib(self,apiKey,appName):
            global zap
            self.zap=ZAPv2(apikey=apiKey)
            self.zap.core.new_session(name=appName,overwrite=True)

     def spider(self,target):
            scanID = self.zap.spider.scan(target)
            while int(self.zap.spider.status(scanID)) < 100:
                   continue

     def ajax_spider(self,target):
            scanID = self.zap.ajaxSpider.scan(target)
            while self.zap.ajaxSpider.status == 'running':
                   time.sleep(2)
                   continue

     def active_scan(self,target):
            time.sleep(2)
            scanID = self.zap.ascan.scan(target)
            while int(self.zap.ascan.status(scanID)) < 100:
                   time.sleep(2)
                   continue

     def passive_scan(self,target):
            time.sleep(2)
            while int(self.zap.pscan.records_to_scan) > 0:
                   time.sleep(2)
                   continue

     def generate_report(self,type,path,filename):
            type = str.lower(type)
            if type == 'html':
                   report = self.zap.core.htmlreport()
            elif type == 'json':
                   report = self.zap.core.jsonreport()
            elif type == 'xml':
                   report = self.zap.core.xmlreport()
            elif type == 'pdf':
                   report = self.zap.core.htmlreport()
                   completeNameTemp = os.path.join(path, filename + '.'+"html")
                   completeName = os.path.join(path, filename + '.'+type)
                   file1 = open(completeNameTemp, "w")
                   file1.write(report)
                   file1.close()
```

```
        pdfkit.from_file(completeNameTemp,completeName)
        os.remove(completeNameTemp)
        return
else:
        print('File extension not supported')
completeName = os.path.join(path, filename + '.'+type)
file1 = open(completeName, "w")
file1.write(report)
file1.close()
```

References

[1] E. Saad and R. Mitchell, “Web Security Testing Guide v4.2”,*OWASP*, 2020.

[2] Ponemon Institute, “Cost of a data breach report,” *IBM Security*, p. 76, 2019, [Online]. Available: https://www.ibm.com/downloads/cas/ZBZLY7KL

[3] D. Sagar, S. Kukreja, J. Brahma, S. Tyagi, and P. Jain, “STUDYING OPEN SOURCE VULNERABILITY SCANNERS FOR VULNERABILITIES IN WEB APPLICATIONS.” [Online]. Available: www.iioab.org

[4] D. Maciel, A. C. R. Paiva, and A. R. da Silva, “From requirements to automated acceptance tests of interactive apps: An integrated model-based testing approach,” *ENASE 2019 - Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, pp. 265–272, 2019, doi: 10.5220/0007679202650272.

[5] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design Science in Information Systems Research,” *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004, doi: 10.2307/25148625.

[6] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A design science research methodology for information systems research,” *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[7] F. Gacenga, A. Cater-Steel, M. Toleman, and W.-G. Tan, “A proposal and evaluation of a design method in design science research,” *Electronic Journal of Business Research Methods*, vol. 10, no. 2, pp. pp89-100, 2012.

[8] M. Felderer, M. Büchler, M. Johns, A. D. Brucker, R. Breu, and A. Pretschner, “Chapter One – Security Testing: A Survey,” *Advances in Computers*, vol. 101, pp. 1–51, 2016, [Online]. Available: http://www.brucker.ch/bibliography/

[9] Y. Makino and V. Klyuev, “Evaluation of web vulnerability scanners,” *Proceedings of the 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2015*, vol. 1, no. September, pp. 399–402, 2015, doi: 10.1109/IDAACS.2015.7340766.

[10] J. de V. Mohino, J. B. Higuera, J. R. B. Higuera, and J. A. S. Montalvo, “The application of a new secure software development life cycle (S-SDLC) with agile methodologies,” *Electronics (Switzerland)*, vol. 8, no. 11, 2019, doi: 10.3390/electronics8111218.

[11] “What Is the Secure Software Development Lifecycle (SSDLC)?” https://www.aquasec.com/cloud-native-academy/supply-chain-security/secure-software-development-lifecycle-ssdlc/ (accessed Sep. 22, 2022).

[12] S. Radack, “THE SYSTEM DEVELOPMENT LIFE CYCLE (SDLC) NIST Special Publication (SP) 800-64, Revision 2, Security Considerations in the System Development Life Cycle.”

[13] “OWASP Top 10:2021.” https://owasp.org/Top10/ (accessed Sep. 29, 2022).

[14] O. Grigorescu, A. Nica, M. Dascalu, and R. Rughinis, “CVE2ATT&CK: BERT-Based Mapping of CVEs to MITRE ATT&CK Techniques,” *Algorithms*, vol. 15, no. 9, p. 314, Aug. 2022, doi: 10.3390/a15090314.

[15] “CWE - Common Weakness Enumeration.” https://cwe.mitre.org/ (accessed Sep. 22, 2022).

[16] P. Mell, K. Scarfone, and S. Romanosky, “A Complete Guide to the Common Vulnerability Scoring System Version 2.0,” 2007. [Online]. Available: http://www.first.org/cvss/team/.

[17] “CVSS v3.1 Specification Document.” https://www.first.org/cvss/v3.1/specification-document (accessed Sep. 22, 2022).

[18] W. M. P. van der Aalst, M. Bichler, and A. Heinzl, “Robotic Process Automation,” *Business and Information Systems Engineering*, vol. 60, no. 4. Gabler Verlag, pp. 269–272, Aug. 01, 2018. doi: 10.1007/s12599-018-0542-4.

[19] Ternopil′s′kyĭ na ́ ts ̀ ional′nyĭ ekonomichnyĭ universytet and Institute of Electrical and Electronics Engineers, *2019 9th International Conference on Advanced Computer Information Technologies : ACIT’2019 : conference proceedings : Ceske Budejovice, Czech Republic, June 5-7, 2019*.

[20] “Vulnerability Scanning”, doi: 10.36227/techrxiv.20317194.v1.

[21] “OWASP ZAP 2.9 Getting Started Guide.” [Online]. Available: https://www.owasp.org/index.php/ZAP.

[22] L. Jian-Ping, L. Juan-Juan, and W. Dong-Long, “Application analysis of automated testing framework based on robot,” *Proceedings of the International Conference on Networking and Distributed Computing, ICNDC*, pp. 194–197, 2012, doi: 10.1109/ICNDC.2012.53.

[23] S. Stresnjak and Z. Hocenski, “Usage of Robot Framework in Automation of Functional Test Regression,” *The Sixth International Conference on Software Engineering Advances*, no. The Sixth International Conference on Software Engineering Advances, pp. 30–34, 2011, [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=icsea_2011_2_10_10057

[24] “Robot Framework User Guide.” https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html (accessed Sep. 22, 2022).

[25] “OWASP ZAP – API.” https://www.zaproxy.org/docs/desktop/start/features/api/ (accessed Oct. 13, 2022).

[26] “OWASP Vulnerable Web Applications Directory | OWASP Foundation.” https://owasp.org/www-project-vulnerable-web-applications-directory/ (accessed Oct. 03, 2022).