INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Plataforma Inteligente para a Automação da Deteção de Vulnerabilidades em Aplicações Web

Diogo da Silva Moreira

Mestrado em Informática e Gestão

Orientador:
Professor Doutor Carlos José Corredoura Serrão, Professor Associado, ISCTE-Instituto Universitário de Lisboa

Co-Orientador:
Professor Doutor João Pedro Calado Barradas Branco Pavia, Professor Auxiliar, ISCTE-Instituto Universitário de Lisboa

Setembro, 2024

TECNOLOGIAS
E ARQUITETURA

Departamento de Ciências e Tecnologias da Informação (DCTI)

Plataforma Inteligente para a Automação da Deteção de Vulnerabilidades em Aplicações Web

Diogo da Silva Moreira

Mestrado em Informática e Gestão

Orientador:
Professor Doutor Carlos José Corredoura Serrão, Professor Associado, ISCTE-Instituto Universitário de Lisboa

Co-Orientador:
Professor Doutor João Pedro Calado Barradas Branco Pavia, Professor Auxiliar, ISCTE-Instituto Universitário de Lisboa

Setembro, 2024

Agradecimentos

A elaboração da presente dissertação de mestrado não teria sido possível sem a ajuda e incentivo de diversas pessoas que se tornaram um pilar muito importante no meu percurso académico.

Em primeiro lugar, gostava de agradecer aos meus orientadores e professores Carlos Serrão e João Pedro Pavia, por terem aceite realizar este projeto e por todo o empenho que dedicaram. Guiaram-me nos momentos mais difíceis e estavam sempre prontos a ajudar quando assim era necessário.

Um agradecimento especial para toda a minha família e namorada, pela motivação, compreensão, estabilidade e alegria que me transmitiram durante este percurso. Sem eles nada disto seria possível.

Por último, quero agradecer a todas as pessoas que fizeram parte do meu percurso académico, incluindo no desenvolvimento desta dissertação, certamente algumas levarei para a vida.

Muito obrigado!

Resumo

Num mundo cada vez mais dependente da tecnologia e numa era em que a conectividade é omnipresente, as aplicações *web* tornaram-se indispensáveis no quotidiano. A evolução destas aplicações, aliada ao aumento exponencial de utilizadores, trouxe consigo não apenas comodidade, mas também desafios significativos em termos de segurança. Garantir a segurança das aplicações *web* e dos seus dados é cada vez mais uma prioridade para as empresas, embora muitas empresas não disponham de *know-how*, tempo e dinheiro para o realizarem.

Na presente investigação foi estudado e desenvolvido um sistema com o objetivo de automatizar o processo de deteção de vulnerabilidades em aplicações *web*. O sistema desenvolvido explora os benefícios da interoperabilidade de duas soluções de automação da ferramenta de análise de vulnerabilidades em aplicações *web* utilizada. A escolha da ferramenta está sustentada na análise ao estado da arte realizada sobre o tema da segurança nas aplicações *web*, que identificou os principais riscos e *scanners* eficazes na deteção de vulnerabilidades associadas a esses riscos. A solução desenvolvida é de baixo custo e requer muito pouca intervenção do utilizador. Com o intuito de validar e avaliar a plataforma desenvolvida, foram realizadas experiências em aplicações com diferentes tipos de vulnerabilidades conhecidas à priori e numa aplicação real. É fundamental garantir a segurança das aplicações *web*, e o sistema desenvolvido mostrou-se capaz de automatizar a deteção dos riscos de vulnerabilidade e devolver os resultados de uma maneira relativamente simples para o utilizador.

Palavras-chave: Aplicação *Web*, Vulnerabilidade, Segurança, *Scanner*, Automatização, Deteção

Abstract

In a world increasingly dependent on technology and in an era where connectivity is omnipresent, web applications have become an essential part of our everyday life. The evolution of these applications, combined with the exponential increase in users, has brought with it not only convenience, but also significant challenges in terms of security. Ensuring the security of web applications and their data is increasingly a priority for companies, although many companies lack the know-how, time and money to do so.

In this research, a system was studied and developed with the aim of automating the process of detecting vulnerabilities in web applications. The system developed exploits the benefits of the interoperability of two automation solutions for the web application vulnerability analysis tool used. The choice of tool is based on the state-of-the-art analysis carried out on the subject of web application security, which identified the main risks and effective scanners for detecting vulnerabilities associated with these risks. The solution developed is low-cost and requires very little user intervention. In order to validate and evaluate the platform developed, experiments were carried out on applications with different types of vulnerabilities known in advance and on a real application. It is essential to guarantee the security of web applications, and the system developed proved capable of automating the detection of vulnerability risks and returning the results in a relatively simple way for the user.

Keywords: Web Application, Vulnerability, Security, Scanner, Automation, Detection

Índice

Índice de Tabelas

Índice de Figuras

Acrónimos

API - Application Programming Interface

ASVS - Application Security Verification Standard

ASP - Active Server Pages

CA – Certification Authority

CGI - Common Gateway Interface

CIS - Center for Internet Security

CLI - Command-Line Interface

CPU - Central Processing Unit

CVE - Common Vulnerabilities and Exposures

CWE - Common Weakness Enumeration

DAST - Dynamic Application Security Testing

DVWA - Damn Vulnerable Web Application

DSR - Data Security Requirements

EL - Expression Language

HTML - HyperText Markup Language

HTTP - HyperText Transfer Protocol

HTTPS - HyperText Transfer Protocol Secure

ID - Identifier

JSON - JavaScript Object Notation

LDAP - Lightweight Directory Access Protocol

MFA - Multi-Factor Authentication

NTLM - NT LAN Manager

ORM - Object-Relational Mapping

OS - Operating System

OWASP - Open Worldwide Application Security Project

SAST - Static Application Security Testing

SIAAS - Sistema Inteligente para Automação de Auditorias de Segurança (Portuguese translation of "Intelligent System for Automation of Security Audits")

SQL - Structured Query Language

SSL - Secure Sockets Layer

TLS - Transport Layer Security

URL - Uniform Resource Locator

VM - Virtual Machine

WASC - Web Application Security Consortium

XSS - Cross-Site Scripting

ZAP - Zed Attack Proxy

CAPÍTULO 1

Introdução

A dependência pela tecnologia num mundo cada vez mais conectado é inegável e nas últimas décadas tem transformado por completo o nosso quotidiano. O número de utilizadores da *Internet* sofreu um aumento muito significativo neste mesmo período, sendo de aproximadamente 4700 milhões por todo o mundo (Althunayyan et al., 2022). Como consequência deste fenómeno, a quantidade de aplicações *web* sofreu um acentuado crescimento, estimando-se a existência de 1800 milhões de aplicações *web* (Althunayyan et al., 2022).

1.1. Problema e Contexto

Para além do aumento mencionado, as aplicações *web* também têm vindo a evoluir, desde simples páginas estáticas na *internet*, para grandes e complexos sistemas de informação dinâmicos. Inicialmente, eram desenvolvidas através de modelos e linguagens mais simples que abrangiam poucos problemas de segurança, sendo que tal facto facilitava muito as tarefas de analisar e proteger as mesmas (Shahid et al., 2022). Atualmente, os seres humanos conseguem interagir uns com os outros, recorrendo a aplicações personalizadas, *user-friendly*, desenvolvidas em diferentes linguagens e plataformas, que possibilitam não só a utilização de vários utilizadores em simultâneo, como o acesso a partir de diversos navegadores *web*. Com a intrusão digital no nosso dia-a-dia, as aplicações *web* evoluíram e temos a possibilidade de realizar praticamente tudo através das mesmas. Temos aplicações *web* que nos permitem executar serviços em diversos setores incluindo a banca, a saúde, o comércio, a educação, entre outros (Althunayyan et al., 2022). A natureza heterogénea destas aplicações e a abundante dependência de sistemas de informação estão a originar inúmeros problemas de segurança e a dificultar o trabalho aos profissionais do setor relativamente a atualizações e proteção face às ameaças e ataques emergentes (Shahid et al., 2022). O aumento da superfície de ataque e do valor associado às aplicações *web*, devido a fatores mencionados anteriormente, contribui para a descoberta de uma maior diversidade de vulnerabilidades associadas às mesmas. Muitos estudos têm vindo a explorar o tema das vulnerabilidades em relação a estas aplicações, abordando também o crescente número de ataques associado. Tendo como base estas investigações, é possível afirmar que em média são identificadas 33 vulnerabilidades em cada

aplicação, sendo que 19% destas dão ao atacante controlo sobre a aplicação e o sistema operacional (Muralidharan et al., 2023). Nota-se também que o número de ataques a este tipo de aplicações aumentou cerca de 88%, correspondendo a mais do dobro da taxa de crescimento anual prevista (Shahid et al., 2022).

A segurança de aplicações *web* é complexa e apresenta muitos desafios devido a estas serem acessíveis pela rede, tornando-as suscetíveis a inúmeros potenciais ataques. Quando um atacante deteta uma vulnerabilidade numa aplicação *web*, este pode causar diversos prejuízos, entre danos financeiros, de reputação ou físicos. Por outras palavras, todos os dados armazenados na base de dados da aplicação podem ser roubados e partilhados com outros atacantes, podendo originar novos pontos de ataque (Lavens et al., 2022).

Com um intuito de se classificar e organizar as principais vulnerabilidades em grupos, optou-se por recorrer a uma organização internacional, OWASP (*Open Worldwide Application Security Project*), que apresenta um ranking dos 10 riscos mais críticos: "OWASP Top 10" (OWASP, 2021).

Inúmeras empresas utilizam *scanners* de vulnerabilidades de modo a facilitar a análise do *software* das suas aplicações *web*. Estas ferramentas permitem monitorizar de uma forma contínua o código da aplicação durante o seu desenvolvimento e após a sua implementação, tornando-se fundamental para um desenvolvimento de *software* seguro, permitindo automatizar o processo da deteção de vulnerabilidades (Lavens et al., 2022). Positivamente, observa-se uma evolução nas análises de segurança assim como nas técnicas de proteção e deteção, uma vez que a realização de apenas análises manuais e testes tradicionais não é adequada nem escalável (Sonmez & Kilic, 2021).

A principal motivação para a elaboração desta dissertação é investigar e desenvolver um sistema que permita automatizar a identificação de vulnerabilidades em aplicações *web*, minimizando a intervenção do utilizador. O objetivo é criar uma solução eficiente, acessível e escalável, que consiga detetar falhas de segurança e reportá-las ao utilizador de uma forma simples e organizada. A plataforma proposta integrará duas abordagens de automação distintas da ferramenta de análise de vulnerabilidades selecionada, permitindo automatizar a deteção do máximo de vulnerabilidades e alertar para a necessidade da correção das mesmas. Esta proposta visa contribuir para a resolução deste problema da falta de segurança em aplicações *web*, oferecendo uma solução mais automatizada que garanta uma análise da segurança e das vulnerabilidades das aplicações, reduzindo o tempo, custos e conhecimento especializado para a sua utilização.

1.2. Questões de Investigação

Este trabalho procura responder à seguinte questão de investigação: Será possível automatizar a deteção de vulnerabilidades em aplicações *web*, num cenário de baixa intervenção e conhecimento por parte do utilizador, numa solução compacta e de custo reduzido?

1.3. Objetivos

Como referido anteriormente, o objetivo principal desta dissertação é o estudo e a elaboração de um sistema que possibilite a automação da deteção de vulnerabilidades em aplicação *web*. Com este estudo, pretende-se identificar e entender os principais riscos associados às aplicações, perceber o funcionamento dos *scanners* capazes de detetar vulnerabilidades, e diferenciar quais os *scanners*, que apresentam resultados mais satisfatórios para cada tipo de vulnerabilidade. Será desenvolvida uma plataforma que permitirá detetar as vulnerabilidades e avisar o utilizador quanto à necessidade de retificação das aplicações *web*. Para o desenvolvimento desta plataforma propõe-se a integração de duas abordagens de automação de uma ferramenta *open-source*, maximizando o controlo sobre a mesma, com o intuito de criar um sistema que para além de ser eficiente e escalável, apresente um alcance superior aos *scanners* conhecidos, no que toca à diversidade de deteção de vulnerabilidades.

1.4. Contribuições

A presente investigação apresenta um contributo no domínio da segurança aplicacional, focando-se na automação da deteção de vulnerabilidades em aplicações *web*. O principal objetivo deste trabalho foi desenvolver um sistema capaz de fornecer uma análise detalhada de potenciais vulnerabilidades de segurança ao utilizador, de forma simples e eficiente, sem exigir conhecimentos avançados em cibersegurança. A solução proposta permite que utilizadores com pouca ou nenhuma experiência na área de segurança informática tenham acesso ao tipo de conhecimento que, tradicionalmente, apenas um *penetration tester* especializado poderia fornecer. O sistema desenvolvido tem carácter gratuito, garantindo acessibilidade ao público em geral, independentemente de recursos financeiros ou competências técnicas avançadas. Esta particularidade contribui diretamente para a democratização da segurança aplicacional, possibilitando que pequenas e médias empresas, bem como indivíduos, possam proteger as suas aplicações *web* contra eventuais ataques, contribuindo para um ecossistema digital mais informado e seguro.

1.5. Metodologia de Investigação

A elaboração desta dissertação será conduzida utilizando a metodologia *Design Science Research* (DSR), aliada a uma Revisão Sistemática da Literatura para obter informação relevante sobre o tema e, posteriormente, desenvolver uma solução com base nos resultados. Segundo o estudo de Peffers e Rossi (2006), a DSR define um processo cíclico que compreende 6 fases (Figura 1), sendo estas: Identificação do problema e motivação, Definição do objetivo, Desenho e desenvolvimento, Demonstração, Avaliação e Comunicação. O objetivo principal deste tipo de metodologia é o desenvolvimento de um artefacto.

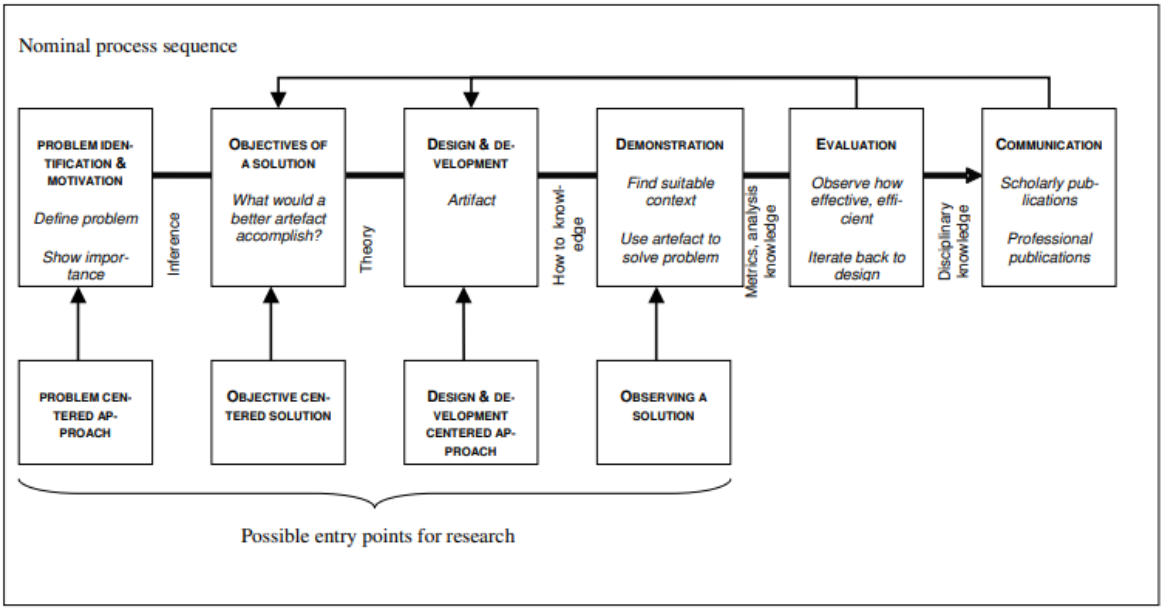

Figura 1 - Metodologia DSR (Peffers & Rossi, 2006)

1.5.1. Identificação do Problema e Motivação

A crescente utilização de aplicações *web* pela sociedade, levou a maioria das empresas a priorizarem a segurança das mesmas. Cada vez mais o valor das aplicações aumenta, quer pela sua utilidade, quer pelos dados que armazenam, sendo alvos frequentes de ataques. Garantir a segurança de uma aplicação *web* é um processo complexo, demorado e requer conhecimentos que a maioria das empresas não possui. Este estudo surge com a motivação de solucionar este problema, automatizando a deteção de potenciais vulnerabilidades em aplicações *web*.

1.5.2. Definição do Objetivo

Como mencionado anteriormente, o objetivo desta dissertação é o estudo e desenvolvimento de um sistema que possibilite a automação da deteção de vulnerabilidades em aplicações *web*. A

proposta para o desenvolvimento deste sistema propõe a integração de duas estratégias de automação de um *scanner*, com o intuito de criar uma plataforma que para além de ser eficiente, apresente um alcance superior aos *scanners* conhecidos, quanto à diversidade de deteção de vulnerabilidades.

1.5.3. Desenho e Desenvolvimento

O desenho e desenvolvimento do sistema é realizado no Capítulo 3, apresentando os motivos da seleção da ferramenta utilizada e do processo desenvolvido.

1.5.4. Demonstração

A demonstração consistirá na execução de análises utilizando a plataforma desenvolvida em "*Python*" para detetar vulnerabilidades numa aplicação sem exigir *expertise* do lado do utilizador e verificar se realmente soluciona o problema identificado.

1.5.5. Avaliação

Com o intuito de avaliar a solução desenvolvida recorreremos a testes sobre aplicações *web* propositadamente vulneráveis e aplicações reais. Os testes serão realizados localmente, para testar o sistema desenvolvido.

1.5.6. Comunicação

Esta fase corresponde à comunicação da dissertação de mestrado através de publicações científicas que irão apresentar os resultados e conclusões do trabalho de investigação.

1.6. Estrutura do Documento

Este trabalho está estruturado em cinco capítulos distintos: o Capítulo 1 diz respeito à introdução; o Capítulo 2 apresenta o estado da arte e a elaboração da respetiva revisão sistemática da literatura; Capítulo 3 apresenta o processo de desenho e desenvolvimento da plataforma para automatizar a deteção de vulnerabilidades em aplicações *web*; o Capítulo 4 aborda a realização dos testes e a análise dos resultados; e por fim, o Capitulo 5 expõe as respetivas conclusões deste trabalho e apresenta sugestões futuras, de modo a dar continuidade a este estudo.

CAPÍTULO 2

Estado da Arte

No presente capítulo, será elaborada uma Revisão Sistemática da Literatura que contextualizará o tema da segurança nas aplicações *web*, sendo possível retirar conclusões sobre os principais riscos de segurança destas aplicações e sobre os *scanners* como parte fundamental da deteção, prevenção e correção desses riscos.

2.1. Revisão Sistemática da Literatura

Com o intuito de aprofundar o conhecimento pelo tema da segurança nas aplicações *web*, as respetivas vulnerabilidades e *scanners* com capacidade de as detetar, foi realizada uma Revisão Sistemática da Literatura seguindo as diretrizes de Kitchenham et al. (2009). Uma revisão sistemática da literatura é um processo de procura baseado em evidências com o objetivo de identificar, interpretar e avaliar a pesquisa sobre um campo de interesse específico.

Para agregar e organizar os artigos resultantes do processo referido acima, foi utilizada a ferramenta *Mendeley*, permitindo filtrar os artigos por ano, autor, título, tendo sido fundamental na organização e gestão das referências.

A Revisão Sistemática da Literatura elaborada compreende três fases principais (Pereira & Serrano, 2020), que se encontram definidas na Tabela 1.

Tabela 1 - Fases da Revisão Sistemática da Literatura (Pereira & Serrano, 2020)

Outlining Systematic Literature Review	Conducting Systematic Literature Review	Reporting the Review
Identificação da necessidade • Desenvolvimento de um sistema para automatizar a deteção de vulnerabilidades em aplicações *web*	Filtragem dos artigos • 39 Artigos	Reportar os resultados obtidos
Objetivo da Revisão da Literatura • Identificar e perceber os riscos mais críticos seguindo a taxonomia do OWASP Top 10 e analisar os *scanners* capazes de os detetar	• Análise das características dos Artigos Finais	
Protocolo da Revisão	Análise dos dados dos Artigos Finais	

• Utilização de uma *Search String*, pesquisa em repositório, aplicação de filtros e definição de critérios de inclusão e exclusão	• Extrair informações sobre vulnerabilidades em aplicações *web* e *scanners* capazes de as detetar	

2.1.1. Perguntas de Pesquisa

A presente Revisão Sistemática da Literatura tem por base as seguintes questões de investigação:

- Quais são os principais riscos e vulnerabilidades das aplicações *web*?

- Quais são as ferramentas que existem e que podem ser usadas para os detetar, como funcionam e qual o desempenho que apresentam?

2.1.2. Base de Dados

Relativamente às bases de dados de publicações científicas utilizadas no presente estudo, optou-se pelas seguintes:

- Web of Science (https://www.webofscience.com)
- IEEE (https://ieeexplore.ieee.org)
- Scopus (https://www.scopus.com)
- Google Scholar (https://scholar.google.com/)

2.1.3. Expressão de Pesquisa

Em relação às expressões de pesquisa, foi designada apenas uma, com o objetivo de realizar uma pesquisa mais completa e direcionada ao tema em específico. A expressão de pesquisa utilizada foi a seguinte:

```
"web app*"
AND ("scanner*" OR "tools" OR "open source")
AND ("vulnerability*" OR detection OR "threat*")
AND (security OR automation OR monitoring OR scanning)
AND "web application vulnerabilities"
```

2.1.4. Critérios de Inclusão e Exclusão

A Tabela 2 demonstra os critérios de inclusão e exclusão utilizados, com o objetivo de restringir a pesquisa e obter os artigos com maior valor para a revisão da literatura a ser executada.

Tabela 2 - Tabela dos Critérios de Inclusão e Exclusão

Critérios de Inclusão	Critérios de Exclusão
"Full Text"	Not in "Full Text"
"Abstract"	Not in "Abstract"
"Articles or Conference"	Not from "Articles or Conference"
"English"	Not in "English"
"From 2010-2023"	"Before 2010"
"No Duplicates"	"Duplicates"

2.1.5. Filtragem

De modo a filtrar os artigos obtidos pela expressão de pesquisa em cada base de dados, foram utilizados alguns filtros para que o resultado fosse o menos vago possível.

O primeiro filtro (*FullText*) teve o intuito de selecionar os artigos relacionados com o tema, devolvendo apenas artigos em que a expressão de pesquisa estivesse contida em qualquer parte do artigo.

O segundo filtro (*Abstract*), apesar de ter muitas parecenças com o primeiro, restringiu-se apenas para os artigos que contém a expressão de pesquisa apenas no seu resumo.

No terceiro filtro (*Articles/Conference*), para conferir desde logo alguma veracidade e qualidade dos artigos selecionados, filtrou-se apenas para artigos de jornais ou de conferências.

Para garantir que todos os resultados seriam em inglês aplicou-se o quarto filtro (*English*).

O quinto filtro (*Date* (2010-2023)), permitiu selecionar apenas artigos mais recentes (com datas superiores ao ano de 2010) dada a evolução da tecnologia nesta última década, tornando-se irrelevante analisar artigos mais antigos.

De forma a realizar-se uma limpeza dos resultados obtidos e excluindo os artigos repetidos aplicou-se o sexto filtro (*No Duplicates*).

Por último, o sétimo filtro (*Manual Filter*), visa excluir os artigos que se tratavam como irrelevantes para a investigação, terminando assim com um total de 39 artigos. É importante referir que estes números servem como uma aproximação, com o objetivo de explicar como a pesquisa foi realizada e como os artigos foram obtidos.

Tabela 3 - Tabela de Filtros

Database	Search String	Filters						
		1	2	3	4	5	6	7
		FullText	Abstract	Articles/ Conference	English	Date(2010-2023)	No Duplicates	Manual Filter
IEEE	"web app*" AND ("scanner*" OR "tools" OR "open source") AND ("vulnerabilit*" OR detection OR "threat*") AND (security OR automation OR monitoring OR scanning) AND "web application vulnerabilities"	25	14	14	14	13	13	9
WoS		48	27	26	26	22	13	6
Google Scholar		415	-	26	26	15	11	6
Scopus		1495	85	78	78	65	43	18
Total							80	39

2.1.6. Características da Amostra

Após a filtragem e seleção dos artigos, estes foram analisados e organizados de modo a facilitar a revisão. Como podemos ver na Figura 2, entre 2019 e 2023, a distribuição dos artigos ao longo dos anos aumentou, uma vez que cerca de 62% dos artigos selecionados, têm a sua data de publicação dentro deste intervalo. Isto pode indicar que a segurança nas aplicações *web* é um problema cada vez mais atual.

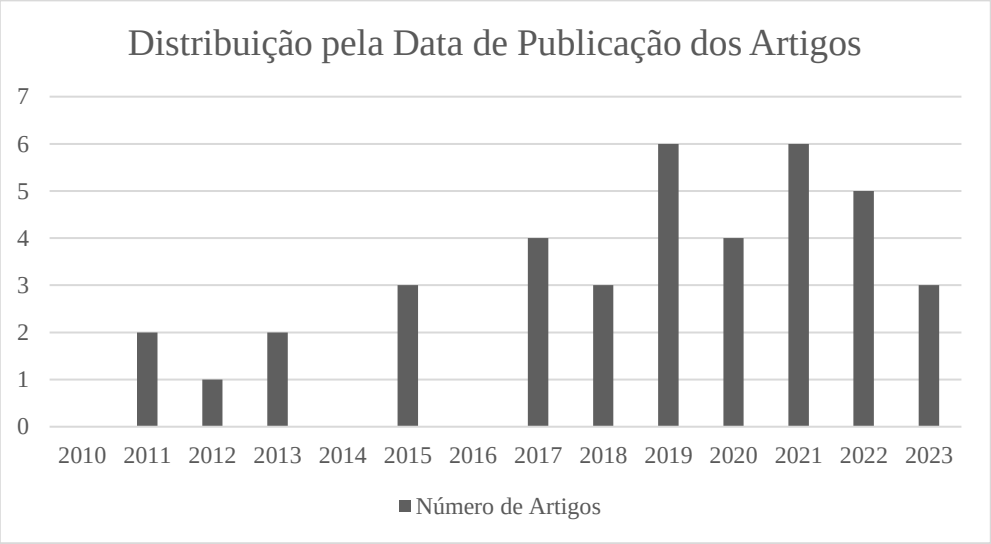

Figura 2 - Distribuição dos artigos ao longo dos anos

2.2. Segurança Aplicacional

Nos últimos anos, o número de ataques a aplicações *web* tem aumentado exponencialmente, fazendo destas um alvo popular para os *hackers*. A título de exemplo, em 2017 um *site* de uma pequena média empresa, reportou em média cerca de 44 ataques por dia. De acordo com a investigação mencionada, este número tenderia a aumentar mais de 50% até ao ano de 2019 (Truong et al., 2019). Cabe aos profissionais do setor assegurar a segurança de aplicações *web* e dos dados presentes nas mesmas. Tendo em conta o número de aplicações *web* existentes,

aliado ao tempo reduzido que estes profissionais dispõem para analisar cada uma, não é escalável a realização destas tarefas sem recorrer a ferramentas automatizadas para facilitar na descoberta e correção das vulnerabilidades. Contudo, a falta de segurança nas aplicações *web* começa logo no processo de desenvolvimento das mesmas. As plataformas utilizadas para o desenvolvimento e análise deste tipo de aplicações oferecem suporte limitado à segurança, o que favorece a ocorrências de erros e esforços substanciais por parte dos profissionais do setor, o que novamente acaba por ser muito complicado devido à pressão de tempo para o produto estar no mercado e à falta de conhecimento ao nível da segurança (Li & Xue, 2011).

2.3. Aplicação Web

Para o desenvolvimento deste estudo, é fulcral ter presente a definição do que é uma aplicação *web*, bem como conhecer a sua arquitetura e o seu funcionamento. Esta é entendida como um *software* ou programa que é acedido e utilizado através de um navegador *web* (Muralidharan et al., 2023). Para que seja possível o utilizador interagir através do navegador *web* com a aplicação é necessário existirem inúmeros componentes e tecnologias que compõem a plataforma a que chamamos de *web*. A *web* é um ecossistema complexo onde estas aplicações são desenvolvidas e hospedadas. Alguns dos componentes e tecnologias que fazem parte da *web* e são fundamentais para o bom funcionamento das aplicações são os protocolos HTTP, servidores *web*, tecnologias para desenvolvimento da aplicação do lado do servidor (exemplo: CGI, PHP, ASP), navegadores *web* e tecnologias para desenvolver a aplicação do lado do cliente (exemplo: *JavaScript*, *Flash*) (Li & Xue, 2011).

2.3.1. Arquitetura de uma Aplicação Web

Como referido anteriormente, uma aplicação *web* é uma parte integral do ecossistema *web* que possibilita a entrega dinâmica de informações e serviços. Em conformidade com a Figura 3, uma aplicação *web* pode conter código tanto no lado do servidor como no lado do cliente. O código do lado do servidor cria as páginas HTML. Durante a execução do código no lado do servidor, a aplicação pode interagir com o sistema de arquivos local ou um banco de dados, localizado no *backend*, para armazenar e recuperar dados.

O código no lado do cliente está incorporado nas páginas HTML e é executado dentro do navegador. Este pode comunicar com o código no lado do servidor e atualizar as páginas HTML de forma dinâmica (por exemplo: AJAX).

Como mencionado anteriormente, existem aplicações *web* especificas para diversos setores, com objetivos e serviços completamente distintos. Isto resulta numa variedade de arquiteturas. Um dos pontos mencionados em diversos estudos está relacionado com o facto da necessidade da arquitetura de uma aplicação ter de estar alinhada com o processo/função de negócio que serve, sendo que existem arquiteturas que apresentam melhor desempenho e são mais adequadas para determinados processos de negócios do que outras.

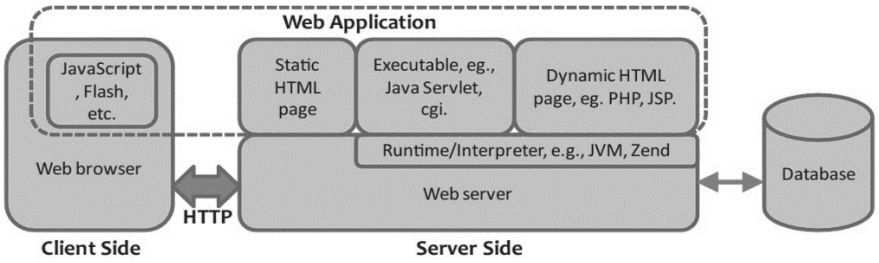

Figura 3 - Arquitetura de uma aplicação web (Li & Xue, 2011)

Segundo o estudo de Lavens et al. (2022), uma tendência mais recente é a migração para APIs da *web*. Estas permitem que uma aplicação, que esteja a correr localmente, utilize a API para comunicar com o servidor.

2.4. Vulnerabilidades em Aplicações Web

Em termos de cibersegurança, a palavra vulnerabilidade é vista como uma falha ou erro que possibilita a descoberta de um ponto de entrada por parte do invasor, permitindo posteriormente a quebra da integridade do sistema (Goe, 2015). Quanto às vulnerabilidades que as aplicações *web* possam apresentar, na medida em que são desenvolvidas e hospedadas num ecossistema complexo, estas estarão diretamente relacionadas com os desafios e debilidades das tecnologias e componentes que as constituem (Li & Xue, 2011).

As vulnerabilidades nas aplicações *web*, não surgem apenas de falhas ou *bugs* associados ao próprio *software* da aplicação, podem surgir também de vírus ou *malware* presentes no sistema que acede a aplicação. Esses vírus podem comprometer a segurança da aplicação ao roubar informações sensíveis, como credenciais de acesso, ou ao modificar o comportamento da aplicação sem o conhecimento do utilizador. Um exemplo disto seria um *trojan* instalado no sistema de um utilizador que, ao interagir com a aplicação *web*, captura as credencias de *login*. Para além disso, a ausência de procedimentos de segurança adequados, como por exemplo a utilização de *password*s "fracas" por parte dos utilizadores, também pode ser considerado uma vulnerabilidade porque permite ao atacante descobrir um ponto de entrada (Goe, 2015).

2.4.1. Categorização das Vulnerabilidades

Ao longo dos anos as vulnerabilidades e os riscos têm sido descobertos e armazenados em base de dados especializadas com o objetivo de categorizá-los. Os riscos comuns em aplicações *web* podem ser categorizados em classes de riscos, existindo diferentes taxonomias como o "OWASP Top 10", "CWE", "CIS Controls", "ASVS" e "WASC Threat Classification". Como mencionado anteriormente, a classificação "OWASP Top 10" (OWASP, 2021), será um ponto chave aquando da realização da presente investigação, uma vez que representa um consenso alargado sobre os riscos de segurança mais críticos para as aplicações *web*. A *Open Worldwide Application Security Project* (OWASP) é uma organização sem fins lucrativos que colabora na melhoria contínua da segurança na *internet*. Não podemos abordar a classificação "OWASP Top 10" sem mencionarmos a *Common Weakness Enumeration* (CWE). A CWE também apresenta uma lista com os 25 riscos de *software* mais perigosos (MITRE, 2023), definindo aspetos de *design* que podem levar a vulnerabilidades. Quando relacionadas com aplicações *web*, as mesmas são igualmente referidas no estudo do "OWASP Top 10", complementando a informação relativamente àquele risco e sugerindo possíveis soluçõesClique ou toque aqui para introduzir texto.Clique ou toque aqui para introduzir texto..

2.4.2. Os 10 Riscos mais importantes em Aplicações Web (OWASP Top 10)

Para abordarmos o tema dos principais riscos em aplicações *web* vamos recorrer ao "OWASP Top 10 2021", como referido acima. Iremos descrever cada um destes riscos presentes neste documento de sensibilização padrão para os programadores e para a segurança das aplicações *web*, que são considerados os mais críticos (apresentados na Tabela 4).

Tabela 4 - Principais riscos segundo a OWASP Top 10 (OWASP, 2021)

Risco	Artigos
Broken Access Control	(Lavens et al., 2022; Shahid et al., 2022; Al Anhar & Suryanto, 2021; Al-Kahla et al., 2021; Mohamed, 2020; Syed et al., 2020)
Cryptographic Failures	(Shahid et al., 2022; Beba et al., 2021)
Injection	(Alazmi & De Leon, 2023; Althunayyan et al., 2022; Badri & Alouneh, 2023; Al Anhar & Suryanto, 2021; Fonseca et al., 2014; Khoury et al., 2011; Lei et al., 2020; Mohammed Draib et al., 2018; Muralidharan et al., 2023; Parvez et al., 2015; Shar & Tan, 2013)
Insecure Design	(Djeki et al., 2021)
Security Misconfiguration	(Al Anhar & Suryanto, 2021; Mohamed, 2020; Syed et al., 2020; Djeki et al., 2021; Gupta & Gupta, 2017)
Vulnerable and Outdated Components	(Al-Kahla et al., 2021; Mohamed, 2020; Syed et al., 2020; Djeki et al., 2021; Gupta & Gupta, 2017)
Identification and Authentication Failures	(Shahid et al., 2022; Al Anhar & Suryanto, 2021; Al-Kahla et al., 2021; Mohamed, 2020; Syed et al., 2020; Djeki et al., 2021; Gupta & Gupta, 2017)
Software and Data Integrity Failures	(Shahid et al., 2022; Djeki et al., 2021)
Securing Logging and Monitoring Failures	(Al Anhar & Suryanto, 2021; Al-Kahla et al., 2021; Mohamed, 2020; Syed et al., 2020; Djeki et al., 2021)
Server-Side Request Forgery (SSRF)	(Shahid et al., 2022; Djeki et al., 2021)

- *Broken Access Control*

O *Broken Access Control*, em português, Quebra de Controlo de Acesso, está relacionado com as políticas criadas para garantir que os utilizadores não possam agir fora das suas permissões. Para clarificar, um administrador de um sistema terá mais privilégios e poderá executar funções de carácter mais importante e crítico do que um simples utilizador. Aquando do comprometimento destas políticas, o invasor consegue obter acesso não autorizado ou aumentar o nível de acesso à respetiva aplicação. Estas falhas resultam, geralmente, na divulgação não autorizada de informações, modificação ou destruição de dados ou até mesmo execuções de funções de negócio fora dos limites do utilizador (Al-Kahla et al., 2021).

Em relação à deteção e prevenção para com esta vulnerabilidade, a mesma deve ser executada e testada do lado do servidor (Al-Kahla et al., 2021). *Broken Access Control* tornou-se a principal classe de riscos, na medida que para ser detetada, pressupõe uma compreensão da lógica do negócio para o qual a aplicação é utilizada. Por este motivo, os *scanners* não oferecem uma defesa satisfatória na sua deteção (Lavens et al., 2022).

- *Cryptographic Failures*

Como o próprio nome indica, este risco está relacionado com falhas da utilização de criptografia, nomeadamente na encriptação de informação sensível. Primeiramente, é importante perceber qual o tipo de informação que estamos a transmitir e que tipo de proteção é necessária. Por exemplo, *passwords*, número de cartões de crédito, relatórios médicos, informação pessoal e segredos de negócio requerem proteção extra, principalmente se esses dados estiverem sob leis de privacidade ou regulamentos. As violações de dados para além de comprometerem os próprios dados, comprometem também a reputação das entidades envolvidas (Djeki et al., 2021).

- *Injection*

O risco de injeção está associado à permeabilidade de uma aplicação para sofrer ataques desse tipo. Alguns tipos de ataques de injeção mais conhecidos são SQL, NoSQL, comandos do sistema operativo (“OS *comand”*), mapeamento objeto-relacional (ORM) e injeção de *Expression Language* (EL) (Al Anhar & Suryanto, 2021). De acordo com Alazmi & De Leon (2023), SQL *Injection* é o ataque mais comum e foi responsável por aproximadamente 2/3 de todos os ataques face ao ano de 2019. SQL (*Structured Query Language*) é uma linguagem usada para criar, aceder e manipular bases de dados (Muralidharan et al., 2023).

Por sua vez, SQL *Injection* é uma forma de ataque contra aplicações *web* que contenham uma base de dados associada, no qual o invasor executa comandos SQL não autorizados, tentando aproveitar vulnerabilidades de entrada não validadas (Khoury et al., 2011). Resumindo, uma aplicação é vulnerável a este tipo de ataques quando os dados fornecidos pelo utilizador não são validados, filtrados ou tratados pela aplicação e quando consultas dinâmicas ou chamadas não parametrizadas sem contexto são utilizadas diretamente no interpretador.

- *Insecure Design*

O Desenho Seguro é uma metodologia que avalia constantemente ameaças e garante que o código seja robustamente projetado e testado para prevenir métodos de ataques conhecidos. É fundamental que nos dias de hoje o desenvolvimento de uma aplicação respeite esta metodologia de Desenho Seguro. Este tipo de risco surge quando a aplicação não é construída com uma perspetiva de segurança adequada. Apesar da maioria das vulnerabilidades, associadas a este risco, em aplicações *web* ser de fácil deteção e evitável, continua a ser um problema em crescimento, pois a maioria dos programadores destas aplicações não têm a consciência da segurança necessária, provocando assim, um grande número de vulnerabilidades em aplicações e *sites* na *internet* (Goe, 2015). Um exemplo destas vulnerabilidades são os *buffers overflows* (Djeki et al., 2021).

- *Security Misconfiguration*

Security Misconfiguration (em português, Configuração de Segurança Incorreta) representa a classe de riscos que engloba todas as vulnerabilidades existentes em aplicações que possuem sistemas de segurança incompletos ou mal configurados. Esta configuração incorreta pode acontecer em diferentes camadas da arquitetura da *web*, sendo mais comuns na camada aplicacional (Eshete et al., 2013). Exemplos comuns desta vulnerabilidade, são os cabeçalhos HTTP passarem informações confidenciais do utilizador na URL, através da utilização dos métodos GET (Syed et al., 2020).

- *Vulnerable and Outdated Components*

A utilização de componentes desatualizadas e vulneráveis aparece na sexta posição das vulnerabilidades mais críticas na lista do "OWASP Top 10". Esta vulnerabilidade acontece devido à utilização de componentes vulneráveis conhecidos, à utilização de *software* vulnerável e desatualizado, à falta de *scanning* regular no sistema, à falta de correção e melhoramento dos componentes e pelo facto de não ser regularmente testada a compatibilidade de atualizações, melhorias ou bibliotecas corrigidas (Al-Kahla et al., 2021).

Para solucionar estas debilidades no sistema é recomendável excluir os componentes, recursos e dependências não utilizadas, registar de forma contínua as versões das componentes tanto do lado do servidor como do cliente e preferenciar apenas componentes, pacotes e *frameworks* oficiais e licenciadas (Syed et al., 2020).

- *Identification and Authentication Failures*

Esta classe de riscos está associada a vulnerabilidades na identificação, autenticação e gestão da sessão do utilizador. É fundamental protegermo-nos contra o tipo de ataques que exploram estas vulnerabilidades para evitar que o atacante consiga acesso a uma sessão válida e ativa de um utilizador, contornando o processo de autenticação entre o utilizador e o servidor para obter acesso aos dados. Algumas vulnerabilidades de identificação e autenticação que podem ser exploradas são a utilização de palavras-passe "fracas", a definição de uma duração de sessão com um valor muito alto, o que origina uma sessão ativa por um longo período sem o utilizador fazer *logout*, e o armazenamento inadequado do identificador da sessão, tonando-o suscetível a roubos por parte dos invasores (Al-Kahla et al., 2021).

- *Software and Data Integrity Failures*

Falhas na Integridade de Dados e *Software*, é a classe de riscos referente a código e infraestruturas que não protegem o sistema contra a violação da integridade do mesmo. Estas vulnerabilidades podem resultar em ataques de código remoto e o seu impacto não pode ser subestimado (Djeki et al., 2021).

- *Securing Logging and Monitoring Failures*

As práticas de segurança defendem que as organizações devem controlar e analisar o tráfico da rede e registar os eventos que ocorrem no sistema. O registo e monitorização adequados não podem evitar que os atacantes iniciem um ataque, mas sem eles fica muito difícil detetar esse ataque, interrompê-los e determinar a extensão dos danos (Mohamed, 2020). Esta vulnerabilidade pode ser detetada facilmente através da análise dos ficheiros de *logs* depois de um *penetration test* (Al-Kahla et al., 2021).

- *Server-Side Request Forgery (SSRF)*

No final da lista, encontramos os riscos associados às solicitações que podem ser executadas a partir do lado do servidor por parte do invasor. Estas falhas ocorrem quando uma aplicação está a tentar aceder a um recurso remoto sem validar a URL fornecida. Ataques a este tipo de debilidades no sistema são provenientes de ações não autorizadas ou acesso a dados dentro do sistema (Djeki et al., 2021).

2.5. Análise de Vulnerabilidades em Aplicações Web

Nesta subsecção iremos apresentar as principais metodologias e abordagens utilizadas na análise de vulnerabilidades, e vamos introduzir o tema dos testes de intrusão, recurso utilizado na verificação das vulnerabilidades em aplicações *web*.

2.5.1. Abordagens "Black-box" e "White-Box"

Relativamente à análise de vulnerabilidades, existem duas abordagens principais: a abordagem *Black-Box* e *White-Box*. Ambas são técnicas para testar *software*, no entanto, enquanto na abordagem *Black-Box* o foco são as entradas disponíveis e as saídas esperadas para cada valor de entrada, na abordagem *White-Box* o foco é a estrutura da aplicação (Verma et al., 2017). Na metodologia *Black-Box* não é necessário conhecer a estrutura interna da aplicação nem ter acesso ao código da mesma, o objetivo é analisar a funcionalidade do sistema. Aquando da utilização de uma abordagem *White-Box* a concentração da análise está na estrutura do código da aplicação, nos ciclos, condições, entre outros elementos. Posto isto, é necessário ter acesso ao código e conhecer a estrutura interna da aplicação a analisar.

2.5.2. Testes de Intrusão

Para determinar se uma aplicação é suscetível a sofrer ataques, ou seja, se é vulnerável, são realizados testes de intrusão para descobrir ou detetar esses problemas de segurança. Esses testes envolvem diversas ferramentas e técnicas, podendo ser distinguidas em duas abordagens diferentes: manual e automática. A abordagem manual recorre à utilização de ferramentas adequadas que permitem, à pessoa ou grupo de pessoas, inspecionarem cuidadosamente a aplicação, à procura de vulnerabilidades, assim como, aplicarem os conceitos que adquiriram com o tempo e experiência. Por outro lado, na abordagem automática, os testes de intrusão são realizados principalmente por *scripts* automatizados e *scanners* de vulnerabilidades, podendo ser utilizados por qualquer pessoa, pois não requerem de um largo conhecimento técnico (Singh et al., 2020). Ambas as abordagens apresentam vantagens e desvantagens. A abordagem automática possibilita a realização de *scans* mais rápidos, diminui significativamente a mão de obra por parte do responsável pela segurança das aplicações e permite uma avaliação contínua dos dados fornecidos pelo utilizador. No entanto, é incapaz de encontrar casos excecionais de vulnerabilidades e são propensos a um número significativo de falsos positivos. Relativamente à abordagem manual, com base na experiência do técnico de segurança, é possível a deteção das vulnerabilidades quando os *scanners* não as detetam, no entanto, é significativamente mais lenta, podendo levar muito tempo para testar completamente uma aplicação.

2.6. Scanners de Vulnerabilidades em aplicações Web

Os *Scanners* de Vulnerabilidades *Web* são ferramentas automáticas que procuram por vulnerabilidades numa aplicação. As técnicas mais comuns são: Testes Estáticos de Segurança de Aplicações (em inglês, *Static Application Security Testing* (SAST)) e Testes Dinâmicos de Segurança de Aplicações (em inglês, *Dynamic Application Security Testing* (DAST)).

Os *scanners* estáticos seguem uma abordagem de teste *White-box*, o que significa acesso completo ao código fonte da aplicação. Estes *scanners* realizam uma análise ao fluxo de dados no código, seguindo por todos os caminhos possíveis procurando por pontos fracos. Uma desvantagem associada a este tipo de *scanners*, é a necessidade de este suportar a linguagem e o *framework* da aplicação *web* a analisar.

Os DAST são ferramentas que utilizam abordagens *Black-box*, que não têm acesso à estrutura interna da aplicação. Utilizam a aplicação por meio da interface *web*, como os utilizadores normais, estando o seu funcionamento normalmente dividido em quatro fases. Resumidamente, a configuração do *scanner* é o ponto de partida e corresponde à primeira fase. A segunda fase, é a fase do rastreamento, o *scanner* cria uma visão geral da aplicação pesquisando recursivamente *links*, *querys* e formulários partindo de uma URL inicial. Na terceira parte, denominada fase da procura por vulnerabilidades, um módulo de ataque é gerado para cada ponto de entrada e tipo de vulnerabilidade. Na quarta e última fase, as respostas provenientes da fase anterior são analisadas e deverão ser confirmadas (Lavens et al., 2022).

Muitos estudos já foram realizados com o objetivo de descobrir qual o *scanner* que apresenta um melhor desempenho e abrange uma maior variedade de vulnerabilidades. Esta avaliação de *scanners* é executada objetivamente quando estes *scanners* são executados em um *benchmark* de vulnerabilidades na *web*. Esses *benchmarks* são projetados especificamente para avaliar os *scanners*, contém código vulnerável baseado nas vulnerabilidades conhecidas das aplicações *web* e são complementados com um conjunto de casos de teste que se assemelham a código vulnerável, mas que não são exploráveis, com o objetivo de testar o desempenho de um *scanner* em detetar falsos positivos (Lavens et al., 2022).

Estes estudos abrangem tanto *scanners* comerciais como os *scanners open-source* e segundo Lavens et al. (2022) na categoria dos *scanners* dinâmicos não existe uma grande diferença entre os *scanners* comerciais e os *open-source*. No entanto, com base na análise do desempenho dos *scanners* estáticos, os *scanners* comerciais por apresentarem um número inferior de falsos positivos conseguem obter pontuações de desempenho superiores aos *scanners open-source*.

De um modo geral, os *scanners* estáticos são mais rápidos, encontram mais vulnerabilidades e conseguem apontar com maior exatidão a localização das vulnerabilidades no código testado. São utilizados principalmente na fase de implementação do Ciclo de Desenvolvimento de *Software* (em inglês, *Software Development Lifecycle* (SDL)) para destacar as vulnerabilidades no código-fonte usando técnicas de análise de fluxo de dados ou análise de contaminação (Tyagi et al., 2019). Os *scanners* dinâmicos são independentes da linguagem em que a aplicação foi construída e reportam falsos positivos. Quanto à cobertura das vulnerabilidades, os *scanners* dinâmicos focam-se principalmente em vulnerabilidades de injeção (Lavens et al., 2022). No entanto, as vulnerabilidades de injeção foram ultrapassadas pelas ameaças de *Broken Access Control* no relatório mencionado anteriormente do "OWASP Top 10". Posto isto, os *scanners* DAST relativamente aos SAST, não são ideais para detetar a nova categoria líder dos riscos mais críticos em aplicações *web*. Adicionando a isto as limitações dos scanners em geral na deteção de violações de limites de confiança, *Path traversal* e injeção de XPATH, é realmente preocupante uma vez que estas violações de limites de confiança geralmente levam ao risco de *Broken Access Control*, o risco considerado como mais crítico (Lavens et al., 2022).

Na Tabela 5, é possível visualizar os analisadores estáticos resultantes da elaboração desta revisão sistemática da Literatura.

Tabela 5 - Scanners estáticos de vulnerabilidades em aplicações web, não comerciais

Nome	Referências
FindSecBugs	(Lavens et al., 2022; Pathirathna et al., 2017)
OWASP WAP	(Tyagi et al., 2019)
RIPS	(Tyagi et al., 2019)
ShiftLeft Scan	(Lavens et al., 2022)
SonarQube	(Lavens et al., 2022)

O funcionamento dos *scanners* dinâmicos normalmente está dividido em quatro fases principais, sendo que agora iremos saltar a fase da configuração do analisador, considerando apenas três fases. A primeira fase é a fase do rastreamento (*crawling*), que é seguida pela fase de ataque (*attack*) e finaliza com a fase da análise (*analysis*) (El Idrissi et al., 2017; Koswara & Asnar, 2019; Qasaimeh et al., 2018). Em seguida explicamos cada uma destas fases pormenorizadamente:

- **Crawling:** Processo que pressupõe reunir todos os URLs acessíveis de uma aplicação *web*. O processo inicia-se a partir de um URL inicial fornecido pelo utilizador e extrai todos os URLs percorrendo página a página até não serem encontrados novos URLs. No final desta fase, o *scanner* deve identificar todos os pontos de entrada, mesmo os que requerem entradas especiais, como nome de utilizador e *password*. Formulários e funções como *GET* e *POST* também são identificados nesta fase.
- **Attack:** No processo de ataque, o *scanner* gera ou utiliza dados já existentes em diretórios para coincidir com os dados reais de entrada requeridos pela aplicação. Os *scanners* utilizam padrões de entrada maldosos para identificar possíveis respostas vulneráveis do servidor *web*. Os caminhos identificados na fase de *crawling*, juntamente com as entradas de dados preenchidas, são enviadas ao servidor para observar a sua resposta.
- **Analysis:** Nesta fase, por sua vez, o *scanner* analisa a resposta proveniente do servidor. O *scanner* lista os erros, que posteriormente, auxiliam na classificação de possíveis vulnerabilidades.

No seguimento da atual revisão da literatura foi possível identificar os seguintes *scanners* dinâmicos presentes na Tabela 6.

Tabela 6 - Scanners dinâmicos de vulnerabilidades em aplicações web, não comerciais

Nome	Referências
Arachni	(Al Anhar & Suryanto, 2021; Alazmi & De Leon, 2023; El Idrissi et al., 2017; Lavens et al., 2022; Mburano & Si, 2018; Qasaimeh et al., 2018; Shahid et al., 2022)
Dirb	(Jain & Jain, 2019)
Nikto	(El Idrissi et al., 2017; Jain & Jain, 2019; Pfrang et al., 2019; Shahid et al., 2022; Sonmez & Kilic, 2021)
Nmap	(Jain & Jain, 2019)
ZAP	(Al Anhar & Suryanto, 2021; Alazmi & De Leon, 2023; Althunayyan et al., 2022; El Idrissi et al., 2017; Lavens et al., 2022; Mburano & Si, 2018; Pfrang et al., 2019; Qasaimeh et al., 2018; Shahid et al., 2022; Sonmez & Kilic, 2021)

Skipfish	(Al Anhar & Suryanto, 2021; Althunayyan et al., 2022; Doupé et al., 2012; El Idrissi et al., 2017; Eshete et al., 2013; Gupta & Gupta, 2017; Lavens et al., 2022; Muralidharan et al., 2023; Qasaimeh et al., 2018; Seng et al., 2019; Shahid et al., 2022)
Vega	(Althunayyan et al., 2022; El Idrissi et al., 2017; Pfrang et al., 2019; Qasaimeh et al., 2018; Shahid et al., 2022)
W3af	(Doupé et al., 2012; El Idrissi et al., 2017; Eshete et al., 2013; Gupta & Gupta, 2017; Jain & Jain, 2019; Koswara & Asnar, 2019; Qasaimeh et al., 2018; Saleh et al., 2015; Shahid et al., 2022)
Wapiti	(Al Anhar & Suryanto, 2021; Dhivya et al., 2022; El Idrissi et al., 2017; Lavens et al., 2022; Pfrang et al., 2019; Qasaimeh et al., 2018; Shahid et al., 2022)
Whatweb	(Jain & Jain, 2019)

Em conformidade com os estudos de Lavens et al. (2022), Pathirathna et al. (2017) e Tyagi et al. (2019), dois *scanners* estáticos *open-source* que apresentam melhores desempenhos são o "OWASP WAP" e o "FindSecBugs", especializados para aplicações desenvolvidas nas linguagens "PHP" e "Java", respetivamente.

De acordo com alguns artigos do SLR, os *scanners* dinâmicos *open-source* que apresentam um desempenho superior são: "ZAP" e "Arachni" (Lavens et al., 2022; Mburano & Si, 2018; Shahid et al., 2022). Após a identificação dos *scanners open-source* que apresentavam os melhores desempenhos, ambas as ferramentas foram estudadas e testadas e a escolha do "ZAP", em detrimento do "Arachni", deveu-se ao facto do "ZAP" oferecer soluções de automatização mais robustas e de apresentar os resultados de uma forma mais organizada e clara.

Destacar também o artigo de Amity University et al. (2019), que descreve a elaboração de uma plataforma "*Python*" para a deteção de vulnerabilidades recorrendo a quatro ferramentas. As ferramentas utilizadas foram "Nmap", "Nikto", "Whatweb" e "Dirb". Neste estudo, são aproveitados os pontos fortes de cada uma das ferramentas anteriormente mencionadas, sendo atribuído a cada uma, um propósito muito específico nas diferentes fases do funcionamento dos scanners *Black-Box*. A fase de recolha de informação ficou encarregue da ferramenta "Whatweb", a análise da rede foi realizada pelo "Nmap", a ferramenta "Nikto" desempenhou a função de detetar as possíveis vulnerabilidades da aplicação e o "Dirb" executou os ataques de força bruta às páginas *web*. Este artigo é um exemplo que apresenta interoperabilidade de diferentes ferramentas para atingir uma maior eficiência no processo de deteção de vulnerabilidades numa aplicação *web*.

Na revisão da literatura, verificou-se a falta de soluções automatizadas e com reduzida intervenção humana que sejam eficazes para a deteção de vulnerabilidades. Embora os *scanners* sejam ferramentas amplamente utilizadas, a sua utilização exige, na maioria dos casos, um certo nível de experiência por parte do utilizador que as vai operar. Além disso, os resultados obtidos por essas ferramentas não são, frequentemente, apresentados de forma intuitiva ou de fácil compreensão, o que dificulta a sua interpretação e aplicação por profissionais menos experientes. Esta lacuna evidencia a necessidade de desenvolver soluções mais acessíveis e automatizadas, que possam ser utilizadas com maior eficácia por um leque mais amplo de utilizadores.

CAPÍTULO 3

Desenho e Desenvolvimento

Ao longo deste capítulo será efetuada uma descrição de todo o desenho e desenvolvimento de um sistema proposto pelo autor e que será responsável pela automação da deteção de vulnerabilidades em aplicações *web*.

3.1. Introdução

Este capítulo detalha o processo de desenho e desenvolvimento do sistema proposto para a automação para a deteção de vulnerabilidades em aplicações *web*. O sistema foi desenvolvido utilizando a linguagem de programação "*Python*", realizando a integração com a ferramenta de análise de vulnerabilidades, ZAP, através da API REST desenvolvida em "*Python*" para permitir a execução de planos de automação de deteção de vulnerabilidades em múltiplas aplicações *web*. A escolha do ZAP, em detrimento de outros *scanner*s conhecidos, deve-se aos seus bons resultados na deteção de vulnerabilidades em aplicações *web*, conforme indicado em investigações anteriores na revisão da literatura. Outros pontos fortes tidos em consideração foram as soluções de automatização que a ferramenta disponibilizava e o controlo granular sobre os resultados que permitia apresentar os mesmos de uma forma simples e organizada.

O principal objetivo deste trabalho é desenvolver um sistema totalmente automatizado com a capacidade de deteção e reporte de potenciais vulnerabilidades em aplicações *web*, que exija uma reduzida (idealmente nula) intervenção e conhecimento por parte de operadores humanos e que represente um custo reduzido para as organizações que o utilizem. Este custo reduzido relaciona-se com a particularidade do sistema em questão poder ser utilizado em sistemas de *hardware* limitado, como por exemplo num "RaspberryPi", ou numa máquina virtual.

Este trabalho segue a linha de um projeto de dissertação de Mestrado realizado anteriormente (Seara, 2023), onde foi desenvolvido um sistema distribuído para deteção de vulnerabilidades em sistemas, não tendo enfoque na segurança de aplicações *web*. O Sistema Sistema Inteligente para Automação de Auditorias de Segurança (SIAAS), desenvolvido anteriormente, é uma ferramenta de rastreamento de vulnerabilidades, que não exige conhecimento especializado em cibersegurança para ser operado, utilizando uma arquitetura de servidor-agente que facilita a escalabilidade do sistema. A arquitetura do SIAAS é composta por múltiplos agentes que se conectam a um servidor central e utilizam comunicação segura via HTTPS. Os agentes analisam a rede local, identificam *hosts*, realizam rastreamentos de vulnerabilidades recorrendo à ferramenta "Nmap" e os dados recolhidos são posteriormente enviados para o servidor. Adicionalmente, foi desenvolvida uma interface de linha de comando (CLI) para facilitar a interação humana com a API do servidor, permitindo gerir configurações e visualizar dados.

Outro objetivo deste trabalho é integrar o novo sistema com o desenvolvido anteriormente, o que impacta na escolha das ferramentas e *software* utilizados. Para alcançar estes objetivos, foram definidas algumas características a serem consideradas na implementação do sistema:

- **Ferramentas e software open-source**: Na seleção das tecnologias, deu-se especial primazia à utilização de ferramentas e *software open-source*. Foi escolhido o sistema operativo "*Linux*", a linguagem de programação "*Python*", a base de dados "*MongoDB*" e o ZAP como ferramenta responsável pelo *scan* das aplicações *web*.
- **Plug-and-play**: O sistema deve ser de fácil instalação e utilização sem a necessidade de configurações complexas.
- **Escalabilidade**: O sistema deve ser capaz de suportar mais ferramentas de análise de vulnerabilidades. Esta característica é obtida através da utilização da "API Python" do ZAP, uma vez que possibilita a decomposição dos resultados a um nível pormenorizado e que permitirá comparar e fazer a ligação com os resultados de outras ferramentas.
- **Segurança**: A comunicação entre os sistemas desenvolvidos e o utilizador final deve ser realizada via HTTPS (usando SSL/TLS) e autenticada. O armazenamento dos dados também deverá ser seguro, assim como a visualização dos resultados, que só deverá ser possível se o utilizador estiver autenticado.

De seguida, serão apresentadas a arquitetura do sistema, as tecnologias utilizadas e uma descrição detalhada do funcionamento do módulo específico que foi desenvolvido, designando-se *siaas-zap*. Este capítulo está dividido nas seguintes secções: Arquitetura do Sistema, Tecnologias Utilizadas e siaas-zap.

3.2. Arquitetura do Sistema

Nesta secção será demonstrado e explicado qual a arquitetura que suporta o sistema desenvolvido. Fazem parte desta arquitetura um conjunto de três módulos principais: *siaas-server*, *siaas-cli* e *siaas-zap*. Na figura 4 apresentamos o diagrama da arquitetura do sistema, assim com as principais interações entre os diferentes componentes (numeradas sequencialmente entre 1 e 10).

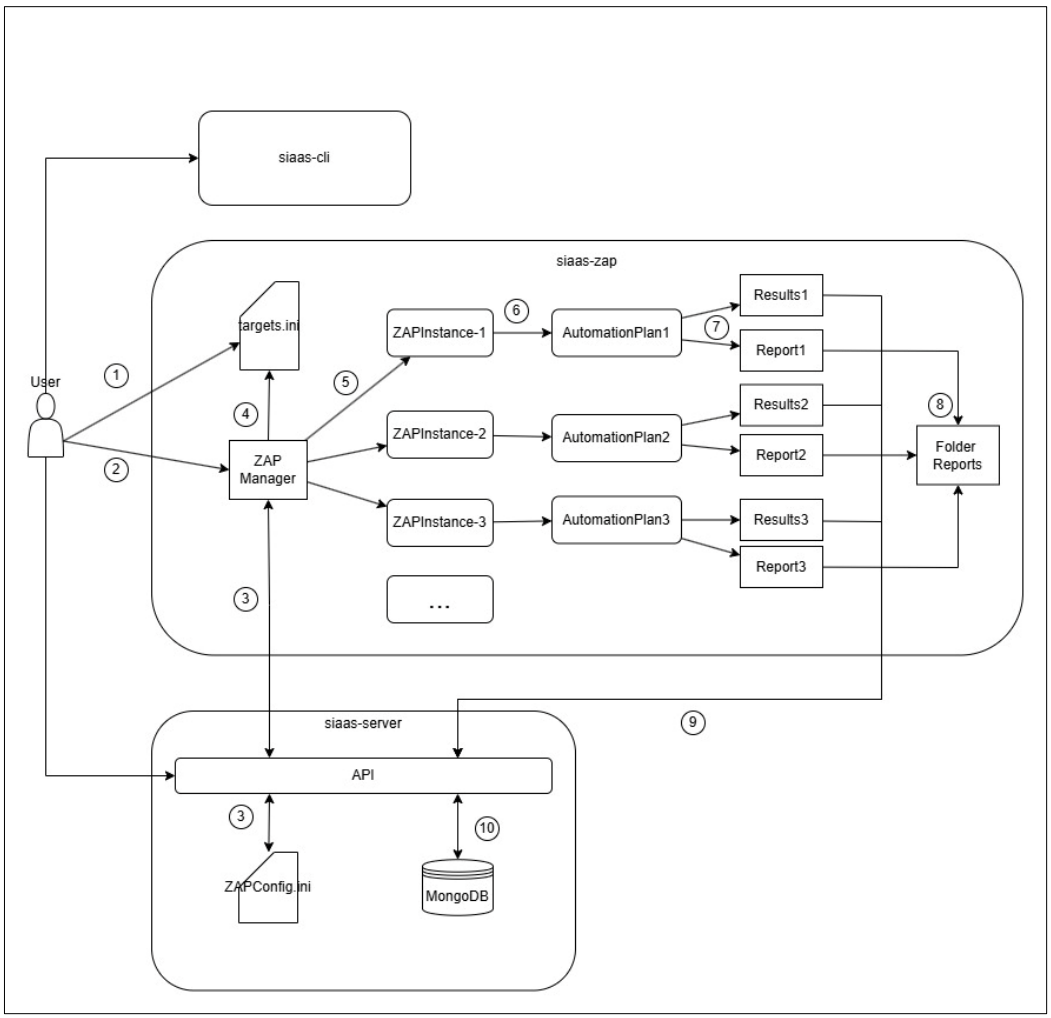

Figura 4 - Diagrama da arquitetura do sistema

O primeiro passo (1) consiste no preenchimento do ficheiro "`targets.ini`" com as informações corretas sobre os *targets* a analisar. Se o utilizador pretender analisar um *site* ou uma aplicação que tenha *login*, terá de fornecer informações como nome da aplicação, URL, *username*, *password* e o URL da página de *login*. Caso o alvo a analisar não tenha a funcionalidade de *login* ou o utilizador não pretenda explorar a aplicação dessa forma, apenas terá de preencher o campo do nome e URL do alvo. Este tema será abordado com maior pormenor na secção 3.5.

Relativamente ao passo (2), posteriormente à configuração do ficheiro, o utilizador dá início ao processo de análise das aplicações, devendo executar o seguinte comando:

```
sudo systemctl start zap_manager
```

Este processo foi criado como uma função de sistema, de forma a ser possível parar e voltar a iniciar de uma maneira mais simples. Estes primeiros 2 passos são os únicos que estão incluídos na lista de tarefas do utilizador para utilizar o sistema, o restante processo é todo automatizado. O processo inicia-se com o "`zap_manager`" a obter as configurações a utilizar na execução dos planos de automação, guardadas do lado do *siaas-server*, através de uma chamada à API do servidor (3), e a realizar a leitura do ficheiro "`targets.ini`", onde o utilizador colocou informações sobre os *targets* a analisar (4).

Depois de obter a informação dos *targets* a analisar naquela execução do serviço, o "`zap_manager`" vai criar uma instância do ZAP para cada *target* (5). Esta instância corresponde a uma inicialização do ZAP num diretório e numa porta diferente. O *scan* dos vários *targets* é sequencial, sendo analisado um *target* de cada vez, por uma questão de *performance*, pois a análise de um *target* numa instância ZAP requer uma quantidade de recursos computacionais considerável. Desta maneira, asseguramos que nenhuma análise é terminada por exaustão do sistema, e que o sistema desenvolvido cumpre o requisito de ser uma solução de baixo custo, ou por outras palavras, que requer recursos computacionais limitados.

Após a criação das instâncias é criado o plano de automação para o respetivo *target*, onde basicamente vamos substituir os campos necessários de um plano pré-definido com as informações sobre as configurações do ZAP e dos *targets* (6).

Quando a análise de um *target* termina, ou seja, o plano de automação está completo, através da utilização da "API *Python*" do ZAP são construídos um relatório, com o resumo das vulnerabilidades, e um ficheiro JSON com os URLs descobertos durante o processo de *crawling*, os alertas de vulnerabilidades e ainda um resumo do plano de automação que permite ao utilizador perceber se este foi executado até ao fim sem erros (7).

Os relatórios gerados serão armazenados numa pasta denominada *reports* dentro do módulo *siaas-zap* (8). Depois da criação deste objeto JSON, o mesmo é enviado através de um *request* HTTP para a API do servidor (9), com as configurações necessárias que asseguram a segurança dos dados e só depois é colocado na base de dados *MongoDB* (10), sendo esta parte tratada do lado do servidor.

Reforçando, este sistema oferece ao utilizador um grande controlo sobre as configurações do ZAP, nomeadamente das suas instâncias, e também dos próprios planos de automação, sendo possível definir atributos como o tempo máximo de certas funcionalidades (em minutos), o número máximo de alertas que podem ser gerados por regra, o número de *browsers* a serem lançados e ainda a profundidade a que o utilizador deseja explorar a aplicação *web*. Com o intuito de preservar a arquitetura *API-centric*, onde a API é o principal ponto de comunicação entre os diferentes módulos do sistema desenvolvido previamente, as configurações de parâmetros do ZAP e dos planos de automação estão armazenadas do lado do servidor e o utilizador consegue visualizá-las e alterá-las através de chamadas à API executadas por detrás da execução de comandos de linha de comandos do módulo *siaas-cli*.

Quanto aos *logs*, cada instância do ZAP criada possui os seus próprios registos. Para além destes, existe também um ficheiro de *logs* do "`zap_manager`" que o utilizador pode consultar. Este ficheiro permite ao utilizador acompanhar, especialmente quando há uma lista de *targets* a analisar, quais os *targets* que já foram analisados, qual o *target* que está a ser analisado e qual o progresso dessa análise.

A escalabilidade do sistema é alcançada pela utilização da "API Python" do ZAP, que permite uma integração flexível para a obtenção dos resultados das análises de vulnerabilidades, possibilitando a construção dos nossos próprios resultados personalizados. Este nível elevado de controlo sobre os dados, favorece o crescimento do sistema no futuro, como por exemplo na integração de outras ferramentas de análise de vulnerabilidades.

Uma vez que, o desenvolvimento do sistema desenvolvido nesta dissertação deriva de um trabalho anterior, é necessário identificar de forma clara quais as minhas contribuições para o mesmo. Foram realizadas alterações nos módulos *siaas-server* e *siaas-cli* previamente desenvolvidos para possibilitar a integração com o novo módulo. Do lado do módulo *siaas-server*, foi adicionado um ficheiro de configuração relacionado com o ZAP e os seus planos de automação. Também se criou uma nova coleção na base de dados existente, onde ficarão armazenados os dados que vêm do módulo *siaas-zap*. Por último, na medida em que o sistema possui uma API para comunicar com os diferentes módulos e visualizar configurações e os dados coletados, foi necessário criar novos *endpoints* na API que possibilitassem essas interações. No que diz respeito ao módulo *siaas-cli*, foram adicionados comandos de linha de comando para permitir que o utilizador consiga interagir com o sistema através da linha de comandos. Por último o módulo *siaas-zap*, foi desenvolvido de raiz, e é o responsável por acrescentar a funcionalidade de deteção de vulnerabilidades em aplicações *web* ao sistema.

3.3. Tecnologias Utilizadas

O sistema de automação de deteção de vulnerabilidades foi desenvolvido utilizando um conjunto de tecnologias robustas, flexíveis e eficientes. As principais tecnologias utilizadas são:

- **Python**: Escolhido pela sua simplicidade, ampla comunidade de suporte e vasta gama de bibliotecas disponíveis, facilitando a integração com a API do ZAP Proxy e outras funcionalidades necessárias. A versão utilizada no desenvolvimento foi a versão 3.8.10.
- **ZAP**: Ferramenta líder em segurança de aplicações *web*, desenvolvida pela OWASP, que oferece uma API robusta que permite automação completa dos testes de segurança. A versão do ZAP utilizada foi 2.15.0.
- **API Python do ZAP**: Permite o controlo programático das funcionalidades do ZAP, essencial para a automação dos testes. A versão da API utilizada no sistema é 0.3.1.
- **Ubuntu**: Escolhido como sistema operativo devido à sua estabilidade, segurança e popularidade entre *developers*, garantindo um ambiente fiável para a execução do sistema. A versão do Ubuntu é a 20.04 LTS (Focal Fossa) (64-bit).
- **MongoDB**: Escolhido como base de dados pela sua flexibilidade e capacidade de armazenar documentos JSON, alinhando-se bem com a estrutura dos resultados obtidos do ZAP.

3.4. Configuração Inicial

Nesta secção serão abordados os passos iniciais necessários para a configuração do sistema. Não esquecendo o requisito de que o sistema deve ser de fácil utilização é fundamental uma configuração inicial simples. Vamos separar a configuração inicial necessária pelos 3 módulos envolvidos neste sistema:

- **siaas-server:**
  - Clonar o repositório: https://github.com/DiogoMoreira4/siaas-server.git;
  - Abrir uma linha de comando na raiz da pasta "*siaas-server*";
  - Executar o comando "`sudo ./siaas_server_install_and_configure.sh`";
  - Iniciar o módulo do servidor: "`sudo systemctl start siaas-server`".

- **siaas-zap:**
  - Clonar o repositório: https://github.com/DiogoMoreira4/siaas-zap.git;
  - Abrir uma linha de comandos na raiz da pasta "*siaas-zap*";
  - Executar o comando "`sudo ./setup.sh`";
  - Executar novamente o script "*setup.sh"* mas desta vez, sem privilégios *root*. O comando fica assim "`./setup.sh`", isto permitirá correr os planos com as permissões de utilizador normal como é suposto.

- **siaas-cli:**
  - Clonar o repositório: https://github.com/DiogoMoreira4/siaas-cli.git;
  - Abrir uma linha de comandos na raiz da pasta "*siaas-cli*";
  - Executar o comando "`sudo ./siaas_cli_install_and_configure.sh`";
  - Editar e exportar o ficheiro com as credenciais "`source ./siaas_env`";
  - Executar os comandos "`siaas-cli …`" existentes para interagir com o sistema e visualizar dados na linha de comandos.

Depois destas configurações iniciais o sistema está totalmente configurado e pronto a ser utilizado. O utilizador pode editar o ficheiro `targets.ini` e iniciar o *scan*, como demonstrado ao pormenor na secção 3.3. Estas configurações também estão disponíveis nos ficheiros "README.md" de cada repositório "git" utilizados.

3.5. SIAAS ZAP

O *siaas-zap* é o módulo responsável pela automação da deteção e análise de vulnerabilidades em aplicações *web*. Considerando sempre o objetivo de desenvolver um sistema que não necessitasse do lado do utilizador final, experiência ao nível da cibersegurança, foi essencial entender o funcionamento do ZAP e conhecer as opções de automação que oferecia para que fosse possível, além de obter resultados de qualidade aquando das análises, obter também um grande nível de automação.

Depois de avaliadas e testadas várias opções de automação que o *scanner open-source* disponibilizava, chegámos à conclusão de que os planos de automação, "*Automation Framework*", juntamente com a utilização da "API *Python*" do ZAP seriam a opção que, apesar de uma dificuldade de utilização mais elevada, oferecia maior flexibilidade e um maior acesso a todas as funcionalidades do ZAP (ZAP DEV Team, 2024).

A "*Automation Framework*" permite controlar o ZAP através de um ficheiro YAML e suporta três domínios muito importantes para o ZAP funcionar da melhor maneira possível: *Environment*, *Authentication* e *Jobs*. No *Environment*, define-se qual a aplicação na qual os *jobs* podem atuar. A *Authentication* é uma subsecção muito importante dentro do *environment* quando o utilizador quer testar uma aplicação ou um *site* que tenha a opção de *login* para uma análise mais profunda, e esta opção de automação é capaz de utilizar todos os mecanismos de autenticação suportados pelo ZAP. Este tema será abordado com mais pormenor em seguida. Por fim, os *jobs*, podemos encará-los como diferentes funcionalidades que o ZAP tem, como por exemplo *spider*, *spiderAjax* e *activeScan*.

Em relação à "API *Python*" do ZAP, esta é outra estratégia de automação utilizada na elaboração deste sistema, que permite ter um controlo praticamente total sobre o ZAP, muito parecido às funcionalidades da interface de *desktop*. É a opção de automação que vai permitir que o sistema desenvolvido consiga acompanhar o progresso da análise e ter acesso a todos os dados necessários para criar os diferentes elementos que compõe o resultado final da análise de cada alvo.

Nas subsecções seguintes serão explicadas de forma pormenorizada o fluxo de trabalho automatizado desenvolvido descrevendo as principais funcionalidades e componentes que integram o sistema.

3.5.1. Leitura do ficheiro dos targets

Quando o serviço "`zap_manager`" é iniciado, após ler os ficheiros necessários para a sua configuração, a primeira tarefa a ser executada é a função responsável pela leitura do ficheiro "`targets.ini`", onde o utilizador final colocou a informação mínima necessária para analisar as aplicações *web* ou *sites* desejados. O utilizador deve alterar o ficheiro "`targets.ini`" antes de iniciar o serviço. A função vai ler o ficheiro onde o utilizador colocou os detalhes sobre os *targets* e vai criar uma lista com objetos que correspondem a cada *target*.

Cada objeto ainda terá um atributo booleano "has_auth" que define qual o plano de automação a modificar e a ser executado para cada *target*. No caso de o utilizador preencher um *target* com todos os campos obrigatórios para realizar um *scan* com autenticação, o valor do atributo "has_auth" obterá o valor de verdadeiro. Caso os campos mínimos para autenticação não estejam preenchidos, o atributo "has_auth" ficará a falso.

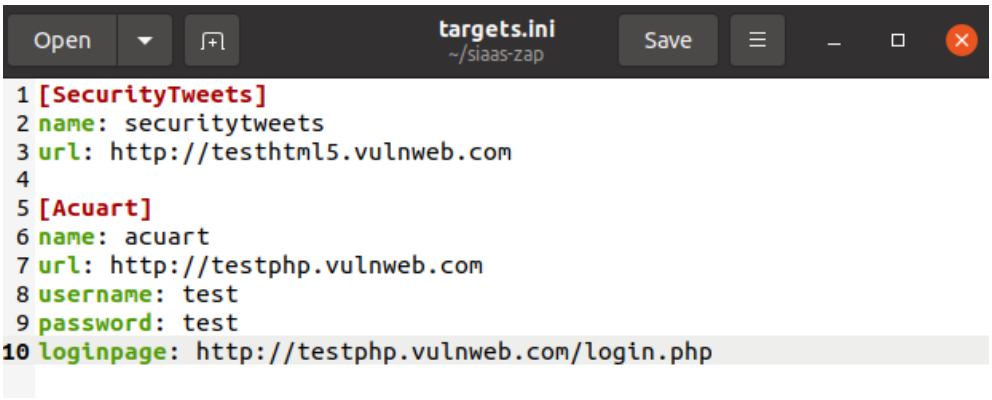

Figura 5 - Exemplo ficheiro targets.ini

Na imagem acima (Figura 5), podemos observar a informação necessária para ambos os cenários, sendo que sobre o *target* "SecurityTweets" será realizada uma análise sem autenticação, indicando neste caso o valor do atributo "has_auth" como falso. Por outro lado, no *target* "Acuart" o atributo "has_auth" já será verdadeiro e o plano de automação, que será executado, já terá em vista a autenticação com as credenciais fornecidas pelo utilizador.

3.5.2. Modificação do plano de automação

Posteriormente à leitura dos *targets*, segue-se a modificação dos campos necessários no plano de automação correspondente. No caso do *target* "SecurityTweets", seria modificado o plano de automação sem autenticação. Os planos de automação são ficheiros YAML que permitem controlar o ZAP. É neste momento que as configurações do ZAP e dos planos de automação

(guardados do lado do servidor e que podem ser modificados pelo utilizador final, através de comandos de linha de comando), são lidos e registados no plano de automação a executar para os próximos *targets*. Exemplos destas configurações podem ser o tempo máximo (em minutos), que o utilizador deseja que os *jobs Spider* e *ActiveScan* estejam a correr, correspondendo aos valores das variáveis "`spider_max_duration`" e "`active_scan_max_scan_duration`", respetivamente. Quando o utilizador não pretende limitar a análise por tempo, quantidades de navegadores a serem lançados, entre outras configurações, deve colocar o valor a 0.

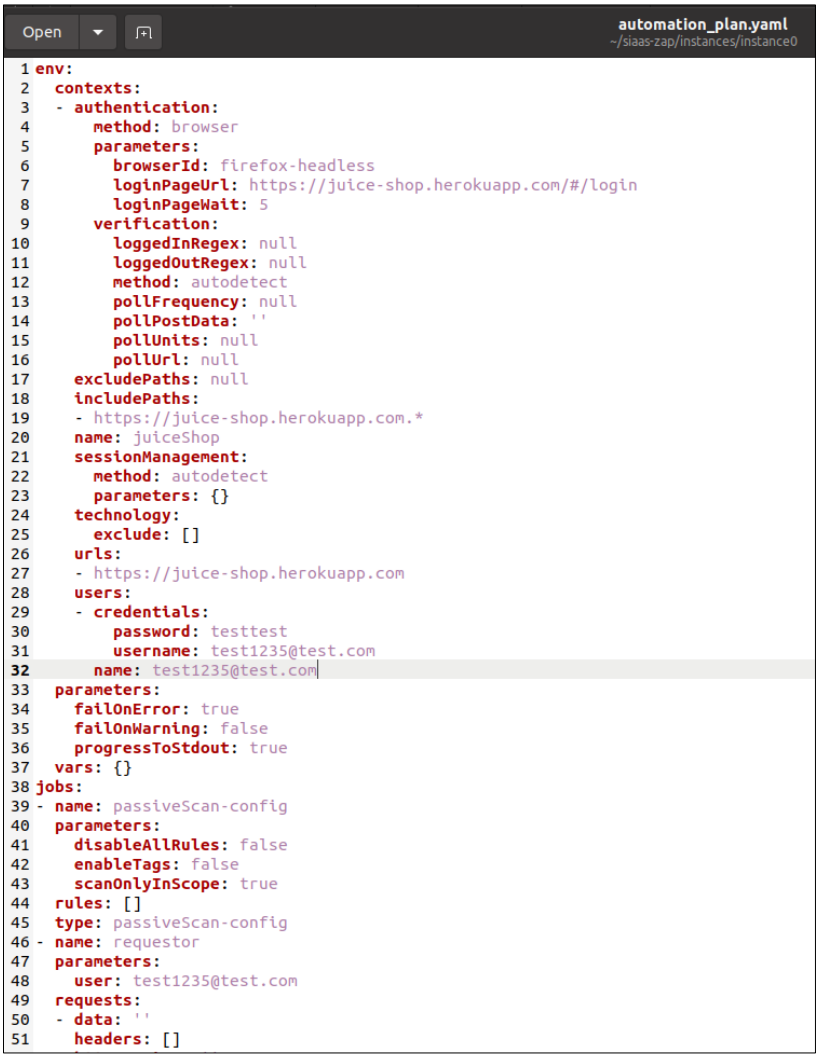

Figura 6 - Exemplo YAML do plano de automação

3.5.3. Autenticação

No capítulo da autenticação é importante perceber como o ZAP lida com este tema, os métodos de autenticação que suporta e as suas limitações. A autenticação é fundamental na perspetiva de ser possível realizar análises de vulnerabilidades mais completas e fidedignas. Um dos aspetos positivos no uso do ZAP, é a capacidade de lidar com diferentes métodos de autenticação para simular interações de utilizadores autenticados em aplicações *web*. Cada método de autenticação no ZAP é configurado dentro de um contexto específico que define como a autenticação é gerida para aquele ambiente. Estes métodos permitem a definição de utilizadores e a gestão automática de sessões *web*, incluindo a reautenticação quando necessária.

Abaixo, são descritos os principais métodos de autenticação suportados pelo ZAP:

- **Autenticação Manual:** é o método mais simples onde o utilizador realiza a autenticação manualmente, através do navegador enquanto utiliza o ZAP como *proxy*. Uma vez autenticado, o utilizador seleciona a sessão HTTP correspondente para ser usada pelo ZAP. Este método não vai ser utilizado neste projeto, uma vez que procuramos a automatização de todo o processo;
- **Autenticação baseada em formulário:** é usada quando a aplicação *web* utiliza um formulário de *login* (submissão de um formulário ou uma solicitação GET para uma URL de *login*) para autenticação com um par de credenciais (nome de utilizador e senha);
- **Autenticação baseada em JSON:** é semelhante à autenticação baseada em formulário, mas em vez de enviar dados do formulário, envia um objeto JSON para a URL de *login* usando um par de credenciais (nome de utilizador e senha);
- **Autenticação HTTP/NTLM:** é usada para cenários em que a autenticação é gerida através de cabeçalhos de mensagem HTTP, recorrendo a esquemas de autenticação como *Basic*, *Digest* ou NTLM. Este método é comum em ambientes corporativos onde o NTLM ou a autenticação HTTP são utilizados para controlar o acesso.
- **Autenticação baseada em Script:** utilizada para cenários onde a autenticação é complexa e não pode ser gerida pelos métodos pré-definidos. Este método permite ao utilizador definir *scripts* personalizados que executam a lógica necessária para autenticar um utilizador numa aplicação *web* específica.

Com a utilização da "*Automation Framework*" é possível automatizar esta etapa de autenticação utilizando um tipo de autenticação adicional: Autenticação baseada no Navegador (*Browser Authentication*). Este tipo de autenticação utiliza um navegador para efetuar o *login* no *site* alvo. Este método é particularmente útil para cenários onde a autenticação é complexa e envolve interações que são mais facilmente realizadas num ambiente de navegador. Utiliza o primeiro campo do tipo "*text*" ou "*email*" para o nome do utilizador e o primeiro campo do tipo "*password*" para a senha. O método é flexível o suficiente para lidar com páginas de *login* onde o campo da *password* só se torna visível após a introdução do nome de utilizador. No entanto, não suporta páginas de *login* que não submetem a autenticação quando a tecla "*Enter*" é utilizada no campo da *password* e páginas de *login* que contenham os campos de nome de utilizador e *password* em páginas diferentes. A resposta que contém o *token* da sessão é identificada como a primeira resposta após o *login* que contenha uma das seguintes opções:

- Um cabeçalho de Autorização (*Authorization header*);
- Dados JSON que contenham um elemento chamado *AccessToken* ou *token*.

Este método adiciona um nível adicional de suporte para processos de autenticação que requerem interações específicas dentro do navegador, o que possibilita, por parte do ZAP, a simulação com maior precisão do comportamento de um utilizador real a navegar e a autenticar-se numa aplicação *web(ZAP DEV Team, 2024)*.

3.5.4. Spider

O processo de *spidering* no ZAP envolve a identificação inicial de URLs (sementes), seguida pelo envio de requisições HTTP por meio de uma *pool* de *threads* geridas pelo "`SpiderController`". Cada requisição é enviada usando o "`HttpSender`", que é configurado para não seguir redirecionamentos automaticamente. As respostas HTTP recebidas são analisadas por *parsers*, que extraem novas URLs e recursos analisando HTML, JavaScript, CSS e outros formatos de dados. Os URLs descobertos são adicionados como novas tarefas na *pool* de *threads*, criando um ciclo contínuo de descoberta e análise. Durante esse processo, o estado do *Spider* pode ser pausado e retomado, utilizando *locks* e condições para sincronização das *threads*. O progresso e os resultados são continuamente notificados aos ouvintes "`SpiderListener`", permitindo a atualização da interface do utilizador e outras ações baseadas nos resultados do varrimento. O processo continua até que todos os URLs sejam visitados ou até que seja manualmente interrompido, garantindo uma cobertura abrangente do *site*.

3.5.5. Ajax Spider

O processo de varrimento do AJAX no ZAP envolve a configuração inicial, onde o "`AjaxSpiderJob`" lê os parâmetros, como contexto, URL inicial, utilizador, e elementos a serem clicados ou excluídos. O "`SpiderThread`" é iniciado para simular a navegação *web* utilizando um *WebDriver* (por exemplo, *Selenium*), que automatiza um navegador real para carregar e interagir com a aplicação *web*. Durante o carregamento da página, são realizadas várias requisições HTTP/HTTPS para obter recursos como HTML, CSS e JavaScript.

O *WebDriver* executa o JavaScript presente na página, realizando requisições AJAX que são monitoradas pelo ZAP. O navegador analisa o DOM para identificar *links*, formulários e elementos interativos. Todos os *links* (<a href>), formulários (<form action>), e requisições AJAX são intercetados e analisados, colecionando URLs adicionais e recursos dinâmicos.

Durante o varrimento, são trocadas requisições HTTP comuns (GET, POST) e requisições AJAX (XMLHttpRequest, Fetch API). As respostas incluem códigos de sucesso (200 OK), erros (4xx, 5xx) e redirecionamentos (3xx).

Semelhante ao que acontece no processo do *Spider*, o *Spider Ajax* continua até que todos os URLs sejam visitados ou até que o tempo limite seja atingido, permitindo que o ZAP descubra URLs e recursos dinâmicos, fornecendo uma análise abrangente da aplicação *web*.

3.5.6. Active Scan

A componente *ActiveScan* do ZAP configura e executa varrimentos ativos em aplicações *web*. Inicialmente, a classe lê os parâmetros de configuração, como contexto, política de varrimento, utilizador e duração máxima. Com base nestes parâmetros, cria um alvo (Target) que define o alvo da varrimento, incluindo URLs e, opcionalmente, o utilizador autenticado. A política de varrimento ScanPolicy define regras específicas para o varrimento, como a força de ataque (AttackStrength) e o limiar de alerta (AlertThreshold) para cada *plugin*.

Durante a execução do varrimento, o "`ExtensionActiveScan`" gere o processo de ataque, enviando diversas requisições HTTP/HTTPS para a aplicação a fim de detetar vulnerabilidades. As requisições incluem métodos comuns como GET e POST, além de modificações para incluir parâmetros maliciosos, com o objetivo de explorar vulnerabilidades como *SQL Injection e Cross-Site Scripting* (XSS). O ZAP também realiza *fuzzing*, injeções de código, ataques de força bruta, análise de portas e testes de segurança de sessão.

As respostas HTTP/HTTPS são analisadas para identificar sinais de vulnerabilidades, como erros de servidor e comportamentos anormais. O *ActiveScan* recolhe e armazena informações sobre as vulnerabilidades detetadas, reportando o progresso de maneira contínua. O varrimento pode ser interrompido manualmente ou automaticamente ao atingir o tempo máximo configurado, e os resultados finais são consolidados para análise e criação de relatórios.

Quando o ZAP recorre a *WebDrivers* para simular a navegação na *web* e interagir com as aplicações *web* alvos, é necessário abrirem-se portas locais temporárias que possibilitam a comunicação entre o *WebDriver* e o navegador real. Se o varrimento incluir múltiplos contextos ou utilizadores, várias instâncias do navegador podem ser iniciadas simultaneamente. Além disso, o certificado CA do ZAP deve ser importado no navegador para intercetar tráfego HTTPS sem alertas de segurança. As portas padrão para HTTP (80) e HTTPS (443), bem como outras portas usadas pela aplicação, devem estar abertas para a comunicação adequada durante o varrimento.

Em seguida, vamos analisar um exemplo de um ataque realizado pelo ZAP à aplicação alvo para clarificar o funcionamento e entender como é que são gerados os alertas e como é que são definidos os valores dos principais atributos do alerta. O exemplo será sobre a deteção de vulnerabilidade de *SQL Injection* sobre o alvo "JuiceShop". O ZAP envia uma requisição POST para o *endpoint* "`/rest/user/login`" do servidor do alvo, com um *payload* malicioso no parâmetro "`email`", neste exemplo, um apostrofo (‘), e um valor de teste no campo "`password`". Esta manipulação tem como objetivo provocar uma falha na consulta SQL executada pelo servidor. O servidor, ao processar a requisição, tenta realizar uma consulta SQL na base de dados SQLite, onde o valor injetado no parâmetro "`email`" resulta num erro de sintaxe SQL. Como consequência, o servidor devolve uma resposta com o código de erro HTTP 500 (*Internal Server Error*), acompanhada por uma mensagem de erro SQLITE_ERROR, indicando que houve uma falha ao reconhecer o *token* injetado no SQL. Esta troca de mensagens está representada na Figura 7. O ZAP, ao receber a resposta do servidor, analisa a mensagem de erro e, com base nas evidências, neste caso, na evidencia da falha no banco de dados, gera um alerta de SQL Injection. É através da análise das resposta obtidas que todo o processo de geração de alertas se desenvolve.

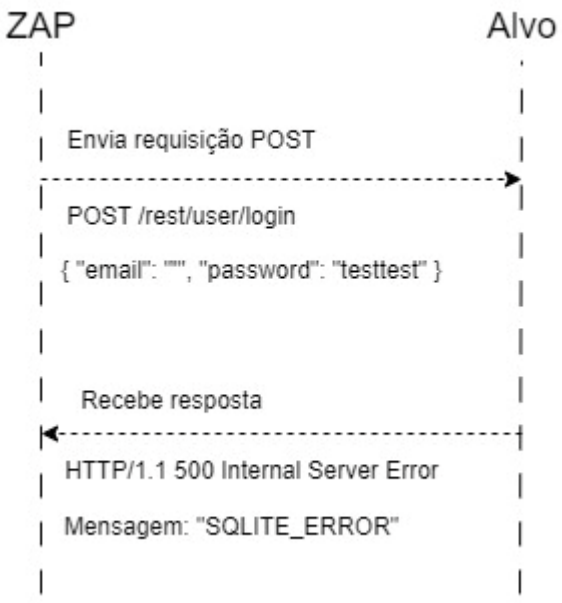

Figura 7 - Esquema da troca de mensagens de um ataque

3.5.7. Resultados

Aquando da conclusão da análise da aplicação, o *siaas-zap* fornece 2 tipos de resultados de análise. É criado um relatório automaticamente por parte do ZAP e é construído um ficheiro JSON com 4 principais componentes: *target*, *urls*, *alerts* e *plan*. Podemos ver um exemplo do relatório no Anexo J. Nesta subsecção será analisada os resultados que resultam no ficheiro JSON e que podem ser consultados na API do servidor e na linha de comandos como podemos visualizar nas Figuras 8 e 9.

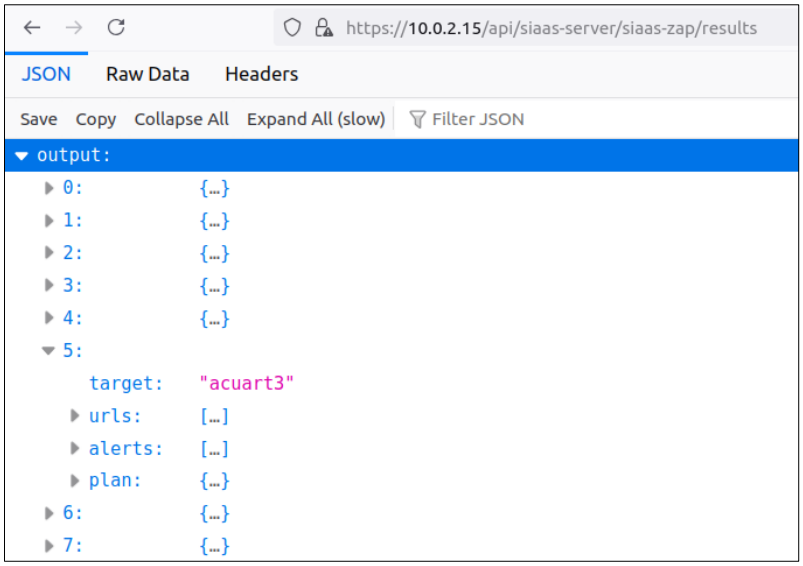

Figura 8 - Visualização dos resultados através da API do servidor

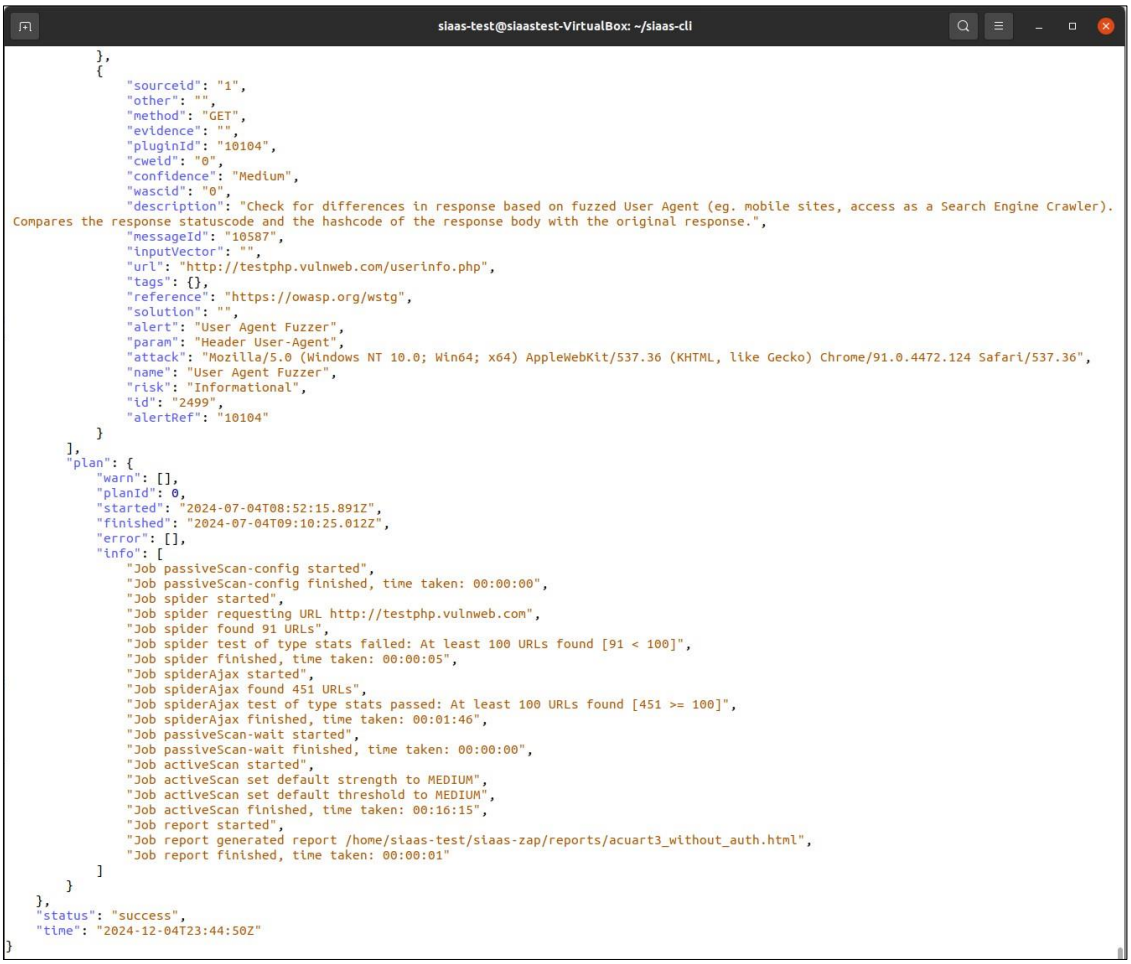

Figura 9 - Visualização dos resultados através da linha de comandos

Os resultados estão divididos em quatro componentes: uma componente *target* que identifica o nome associado à aplicação a analisar; a componente *urls*, que mostra ao utilizador todos os URLs encontrados durante o processo de *crawling*, resultados dos jobs *Spider* e *Ajaxspider*; componente *alerts*, onde o utilizador pode encontrar todos os alertas descobertos durante a análise à aplicação; e, por último, a componente *plan*, que apresenta um resumo do que aconteceu durante a execução do plano de automação ao respetivo *target*.

Como é possível observar na Figura 10, na componente *plan* de cada resultado proveniente da análise de um *target* existem informações que auxiliam o utilizador a perceber se o plano de automação ocorreu como esperado, ou se obtivemos erros ou avisos. Também obtemos informações como a hora de início e de fim do *scan*, os *jobs* que foram executados durante aquele plano de automação e algumas informações sobre esses mesmos *jobs*.

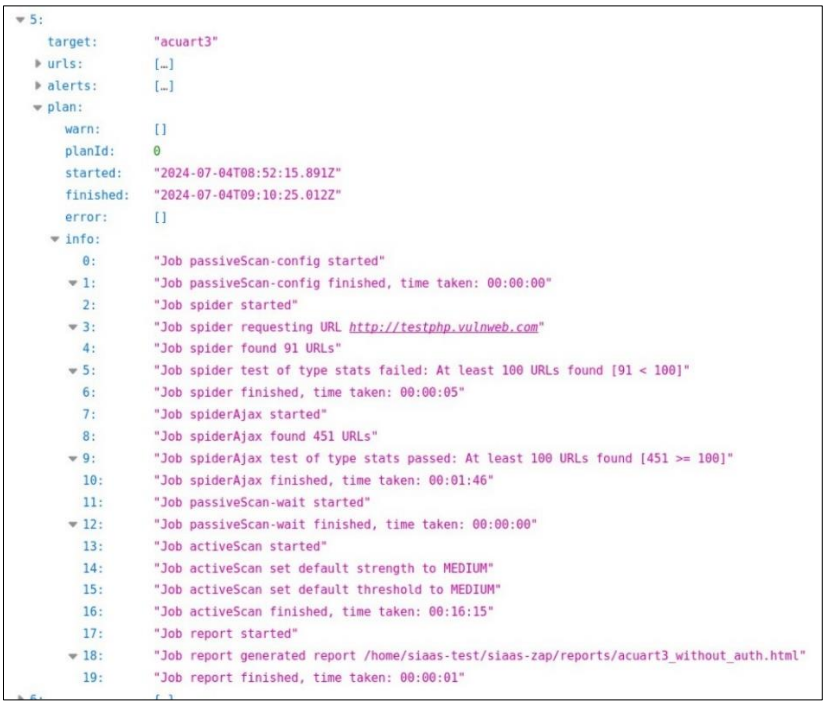

Figura 10 - Visualização da componente "plan" através da API do servidor

Na componente das URLs, Figura 11, são apresentados todos os URLs relacionados com o alvo que foram encontrados pelo ZAP no processo de *crawling*. Os números de URLs que visualizamos na componente do *plan*, na Figura 10, por vezes é superior por ter em consideração alguns URLs que não pertencem ao *target*, ou seja, são encontrados, mas não estão relacionados com o alvo.

Figura 11 - Visualização da componente "urls" através da API do servidor

Quanto à componente *alerts*, apresentada na Figura 12, é aqui que estão guardados os alertas que o ZAP identificou durante todo o processo de análise da aplicação, desde os *scans* passivos procurando pelos URLs, como também durante o verdadeiro "ataque" à aplicação no processo de *scan* ativo. Em seguida, será analisado a estrutura de cada alerta, a sua composição, e também a explicação de como são atribuídos os níveis de confiança.

Figura 12 - Visualização da componente "alerts" através da API do servidor

As características principais dos alertas são os campos de *risk*, *alert*, *description*, *tags*, *confidence* e *solution*. Estes campos estão explicados na Tabela 7.

Tabela 7 - Principais campos de um alerta

Campos	**Descrição**
Alert	Indica qual a categoria/classe de vulnerabilidade a que aquele alerta está associado.
Confidence	Indica o nível de confiança do *scanner* em relação ao alerta despoletado.
Description	Descreve o motivo pelo qual o alerta foi despoletado.

Risk	Indica qual o nível de risco associado à vulnerabilidade identificada no alerta, podendo obter os seguintes valores: High, Medium, Low e Informational.
Solution	Este campo indica uma possível solução para resolver a vulnerabilidade que despoletou o alerta.
Tags	O campo das tags associa o risco da vulnerabilidade encontrada às taxonomias do OWASP e CWE, permitindo através do link obter toda a informação detalhada da vulnerabilidade em causa.

É importante clarificar que o sistema desenvolvido não valida a existência das vulnerabilidades, uma vez que a principal característica do mesmo está relacionada com automação da deteção das mesmas. O sistema deteta potenciais vulnerabilidades nos alvos e gera alertas que serão apresentados ao utilizador. Neste aspeto o campo de *Confidence* de cada alerta vai ter um papel fundamental indicando ao utilizador qual o nível de confiança da existência da respetiva vulnerabilidade, sendo que, para os alertas com o nível de *Confidence High*, a existência da vulnerabilidade é muito provável. A atribuição dos níveis de confiança é a seguinte:

- High: o ZAP tem evidências claras de que o ataque foi bem-sucedido ou da ausência de mecanismos de segurança, ou seja, a aplicação é vulnerável;
- Medium: o ZAP encontra indícios da vulnerabilidade, mas não existiu confirmação da exploração da mesma;
- Low: o ZAP detetou comportamentos suspeitos ou erros relacionados com configurações, mas não tem a confirmação de que a vulnerabilidade pode ser explorada.

CAPÍTULO 4

Testes e discussão de resultados

Neste capítulo serão descritas as condições em que foram realizados os testes com o sistema desenvolvido, quais os alvos que foram analisados, os resultados obtidos, assim como comentários sobre os mesmos.

4.1. Introdução

Nesta dissertação foi desenvolvido um sistema que procura automatizar a deteção de vulnerabilidades em *site*s e aplicações *web*. Recorremos ao ZAP para a realização destas análises e elaboração de relatórios e resultados que mostram para que riscos, categorizados e normalizados segundo o referencial "OWASP Top 10", as aplicações testadas estão vulneráveis. Para testar e validar o sistema desenvolvido, seguindo uma abordagem ética, foram realizados testes locais em ambiente de laboratório, desenvolvido especificamente para o efeito.

Os testes foram executados num único computador equipado com um processador AMD Ryzen 7 5800H with Radeon Graphics correndo a 3.20 GHz, com 16GB de RAM e um SSD de 960GB. O computador através do VirtualBox tinha uma máquina virtual (VM) com 6 GB de RAM, 4 processadores e 25 GB de memória a correr o Ubuntu 20.04 LTS (Focal Fossa).

Com o intuito de testar a viabilidade, adequabilidade e versatilidade do sistema foram realizados vários testes, tendo sido analisados diferentes *site*s e aplicações *web*. De acordo com a lei portuguesa do cibercrime (Portugal, 2009), não é permitido analisar aplicações sem o consentimento dos proprietários das mesmas. Por outro lado, o processo de deteção de vulnerabilidades ataca as aplicações podendo interferir com o bom funcionamento das mesmas, impactando as organizações. Os testes realizados incidiram sobre 2 grupos específicos de aplicações, que foram selecionadas, e que cumpriam com os requisitos anteriormente elencados:

- Aplicações ou *sites* com vulnerabilidades conhecidas, para testar este tipo de ferramentas de análise de vulnerabilidades e para os quais os investigadores têm autorização prévia para testar, a maioria na *web*, sendo que o alvo DVWA estava a correr localmente noutra VM:
    - Acuart (http://testphp.vulnweb.com)
    - Acuforum(http://testasp.vulnweb.com)
    - Altoro (http://demo.testfire.net/index.jsp)
    - bWAPP (https://demo.weblock.ru)

    - DVWA (http://10.0.2.4/DVWA)
    - Juice Shop (https://juice-shop.herokuapp.com)
    - Rest API (http://rest.vulnweb.com)
    - Security Tweets (http://testhtml5.vulnweb.com)

- Aplicação real, o sistema também foi testado numa aplicação real do ISCTE, no ambiente de qualidade (ou seja, não está a ser usada ainda por utilizadores reais), depois de ter sido obtida a autorização apropriada da entidade para o poder fazer.
    - FenixIscteAuth (https://fenix-qua.iscte-iul.pt)

4.2. Recolha de dados

Nesta secção, será discutida a análise realizada aos dois grupos distintos de aplicações *web*: as aplicações propositadamente vulneráveis, desenhadas para testes de segurança, e uma aplicação real, representando um ambiente de produção. Para além da identificação das vulnerabilidades, será também abordado o tempo de execução de cada análise e comparada a *performance* entre os dois grupos.

4.2.1. Aplicações Web Propositadamente Vulneráveis

O primeiro grupo de testes foi realizado em aplicações *web* propositadamente vulneráveis, que têm como principal objetivo fornecer um ambiente seguro para a realização de testes de segurança. Este grupo incluiu aplicações amplamente conhecidas e usadas pela comunidade de segurança para a avaliação de ferramentas de teste de vulnerabilidades.

Referir que no caso das aplicações propositadamente vulneráveis, os resultados obtidos foram comparados com as vulnerabilidades que se esperava detetar de acordo com as informações presentes nos próprios *site*s.

4.2.1.1. Acuart

O “Acuart” é uma aplicação *web* desenvolvida com tecnologias padrão como HTML, CSS, e “JavaScript” no *frontend*, “PHP” no *backend*, e “MySQL” para gestão de dados. O servidor utilizado é o “Apache”, amplamente conhecido pela sua robustez no processamento de requisições HTTP.

Esta aplicação é projetada para servir como um ambiente de testes de segurança, permitindo a identificação de vulnerabilidades comuns em aplicações *web*, como falhas de autenticação, controlo de acesso, exposição de dados sensíveis e configurações inadequadas.

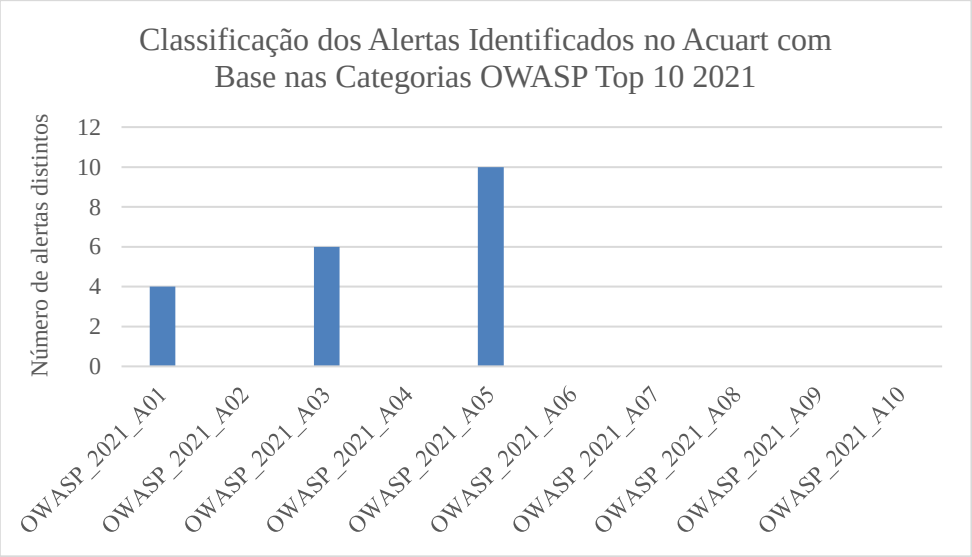

Figura 13 - Classificação dos Alertas Identificados no Acuart

Como podemos observar na Figura 13, o sistema conseguiu detetar potenciais riscos no Acuart. Os alertas foram agrupados em 3 categorias, sendo que a categoria com um número superior de alertas distintos foi a “`OWASP_2021_A05 - Security Misconfiguration`”, com 10 alertas. Esta categoria é crítica, pois abrange erros na configuração de segurança que podem expor a aplicação a uma vasta gama de ataques.

Primeiramente, a tabela em anexo (Anexo A) referente ao "Acuart", evidencia alertas com nível de confiança *High*, que merecem especial atenção, uma vez que representam riscos de vulnerabilidades cuja deteção é altamente fiável, de acordo com a explicação da atribuição dos níveis de confiança dos alertas, abordados no final da subsecção 3.5.7. A vulnerabilidade "*Cross Site Scripting (DOM Based)*", que está associada à categoria "OWASP_2021_A03" e apresenta também um nível de risco *High*, é grave porque ataques XSS podem comprometer a integridade e confidencialidade dos dados da aplicação. Os dois alertas restantes, com um nível de confiança alto estão categorizados sob "OWASP_2021_A05". O "*Content Security Policy (CSP) Header Not Set*" indica que não foi configurado um cabeçalho CPS, o que é crítico para proteger a aplicação contra-ataques de injeção, como o *"Cross Site Scripting*", mencionado em cima. Por sua vez, o "*Server Leaks Version Information via "Server" HTTP Response Header Field*" refere-se à vulnerabilidade do servidor *web* expor informaçãoes desnecessária sobre a versão do *software* ou servidor utilizado através do campo "*Server*" presente no cabeçalho nas respostas HTTP.

Continuando para os restantes alertas com um grau de criticidade mais elevado, a tabela também indica potencias vulnerabilidades de "*SQL Injection*" e "*Cross-Domain Misconfiguration*".

4.2.1.2. Acuforum

O "Acuforum" é uma aplicação *web* propositadamente vulnerável, utiliza o IIS (*Internet Information Services*) como servidor *web*, ASP (*Active Server Pages*) para a lógica do lado do servidor, e Microsoft SQL Server para a gestão e armazenamento de dados.

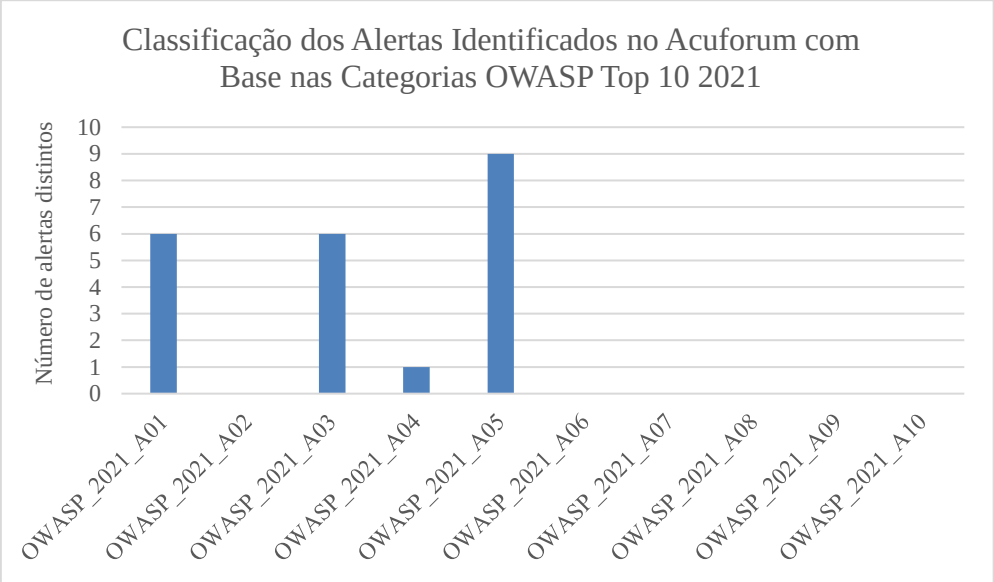

Figura 14 - Gráfico da classificação dos Alertas Identificados no Acuforum

Através da visualização do gráfico da Figura 14, concluímos que para o alvo "Acuforum" foram despoletados alertas que se podem distribuir por 4 categorias do "OWASP Top 10" de 2021. A categoria com a maior diversidade de alertas foi novamente a "OWASP_2021_A05 - *Security Misconfiguration*", seguindo-se as categorias "OWASP_2021_A03 – *Injection*" e "OWASP_2021_A01 – *Broken Access Control*" com 6 alertas diferentes em cada categoria. Neste alvo também foi identificado um risco relacionado com a categoria "OWASP_2021_A04 – *Insecure Design*".

Os alertas presentes, na tabela em anexo (Anexo B), que merecem mais destaque pelo seu nível de confiança, são, à semelhança com o alvo anterior, "*Cross Site Scripting (DOM Based)*", "*Content Security Policy (CSP) Header Not Set*" e "*Server Leaks Version Information via "Server" HTTP Response Header Field*".

No entanto, foram despoletados alertas com uma criticidade mais elevada relacionados com "*SQL Injection*", "*External Redirect*", "*Path Traversal*" e "*Source Code Disclosure - File Inclusion*".

4.2.1.3. Altoro

O “Altoro Mutual” é uma aplicação *web* desenvolvida com tecnologias como o “Apache Tomcat” como servidor *web*, “Java” e “JSP” (JavaServer Pages) para a criação de conteúdo dinâmico no *backend*, e “MySQL” como base de dados, a aplicação simula um ambiente bancário vulnerável.

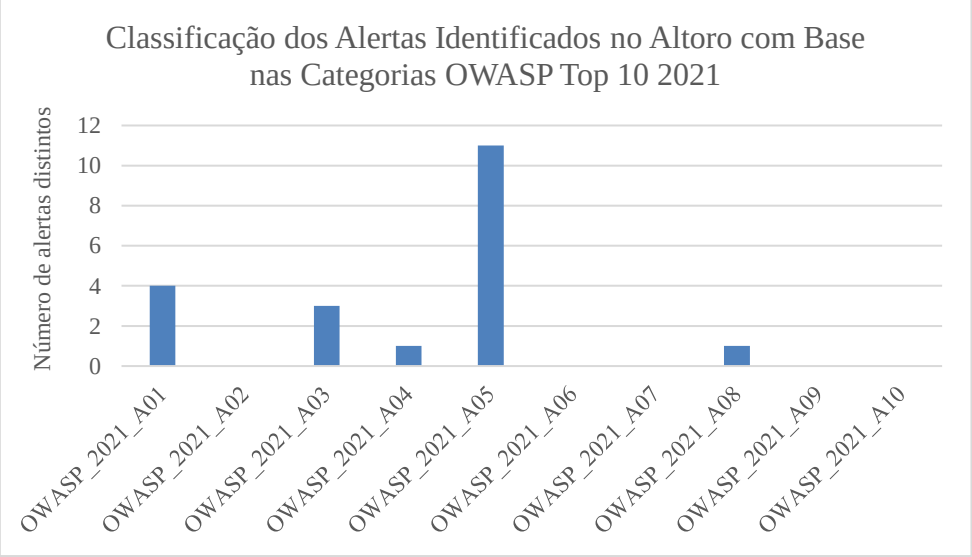

Figura 15 - Gráfico da Classificação dos Alertas Identificados no Altoro

A Figura 15 apresenta a classificação dos alertas identificados no “Altoro” com base nas categorias da “OWASP Top 10”. A categoria mais prevalente foi a “OWASP_2021_A05 - *Security Misconfiguration*”, com 11 ocorrências. Foram detetados riscos que pertencem a outras 4 categoria da taxonomia em estudo, sendo elas: “OWASP_2021_A01 – *Broken Access Control*”, “OWASP_2021_A03 – *Injection*”, “OWASP_2021_A04 – *Insecure Design*” e “OWASP_2021_A08 - *Software and Data Integrity Failures*”.

Analisando a informação presente na tabela em anexo (Anexo C), podemos ver que, para além dos alertas criados com um nível superior de confiança referidos nos 2 targets anteriores, a aplicação “Altoro” apresenta a vulnerabilidade “*Sub Resource Integrity Attribute Missing*”.

Quanto aos alertas de risco elevado falta mencionar os alertas sobre “*Cross Site Scripting (Reflected)*” e “*Cookie Slack Detector*”.

4.2.1.4. bWAPP

O “bWAPP” (Buggy Web Application) é uma aplicação *web* propositadamente vulnerável, desenvolvida para permitir a prática de testes de segurança em ambiente controlado. Utilizando tecnologias como “PHP” no *backend* e “MySQL” para a gestão de dados.

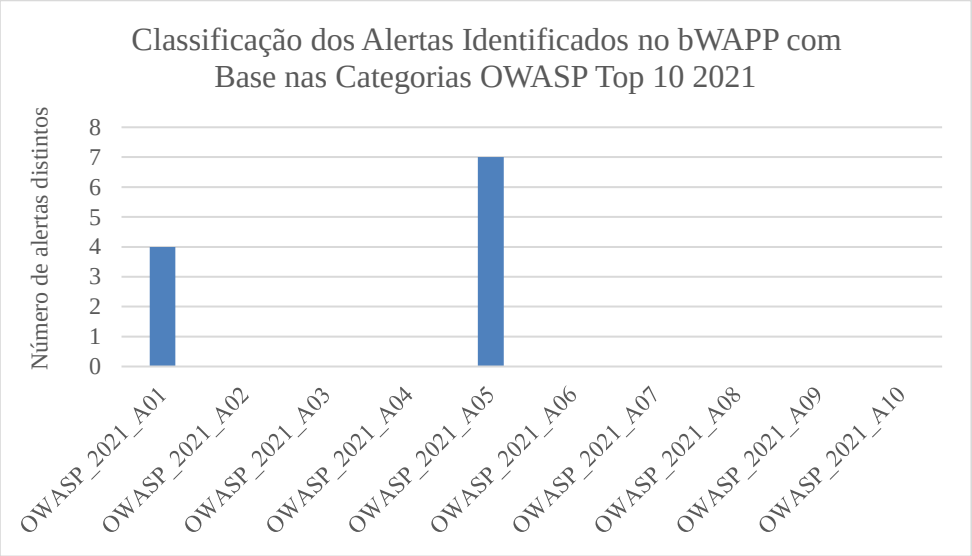

Figura 16 - Gráfico da Classificação dos Alertas Identificados no bWAPP

Na Figura 16, podemos observar a categorização dos alertas identificados durante a análise ao *target* “bWAPP”. As duas categorias identificadas foram a “OWASP_2021_A01 - *Broken Access Control*”, com 4 ocorrências, e a “OWASP_2021_A05 - *Security Misconfiguration*”, com 7 ocorrências.

Os alertas de vulnerabilidades com maior probabilidade de realmente se verificarem no alvo “bWAPP” são “*Content Security Policy (CSP) Header Not Set*” e “*Strict-Transport-Security Header Not Set*”. Não foi detetado nenhum alerta com nível de risco *High*. Estas afirmações, têm como fundamento a visualização da tabela presente no Anexo D.

4.2.1.5. DVWA

Desenvolvida em "PHP" e utilizando "MySQL" para a gestão de dados, a "DVWA" (Damn Vulnerable Web Application) é uma aplicação projetada para oferecer um ambiente seguro, mas repleto de vulnerabilidades, para o treino de técnicas de segurança ofensiva.

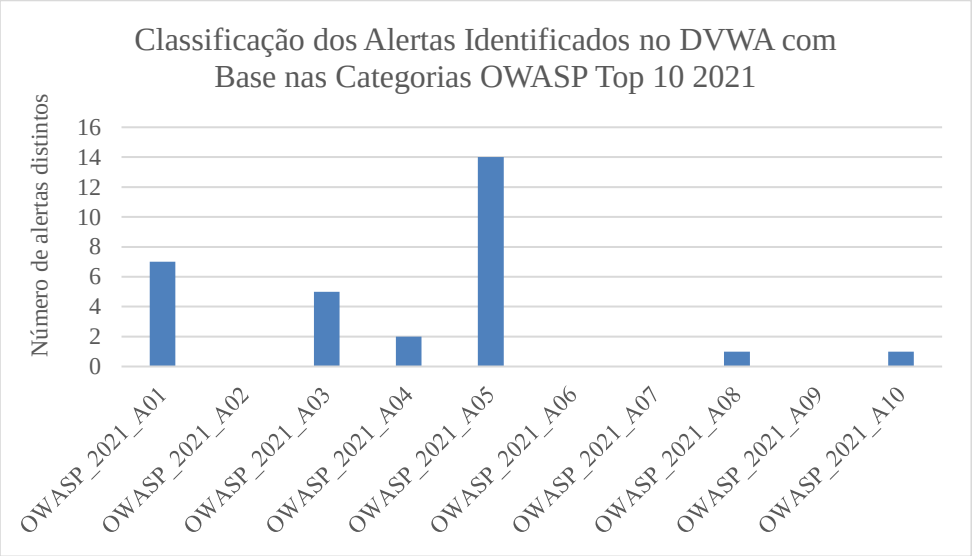

Figura 17 - Gráfico da Classificação dos Alertas Identificados no DVWA

Observando a Figura 17, é percetível de que a "OWASP_2021_A05 - *Security Misconfiguration*" continua a ser a categoria com um número mais elevado de alertas. Para além dessa categoria, o sistema detetou riscos que podem ser agrupados em mais 5 categorias, "OWASP_2021_A01 - *Broken Access Control*", "OWASP_2021_A03 – *Injection*", "OWASP_2021_A04 – *Insecure Design*", "OWASP_2021_A08 - *Software and Data Integrity Failures*" e "OWASP_2021_A10 - *Server-Side Request Forgery* (SSRF)".

No caso da aplicação "DVWA", a correr localmente, os alertas identificados com um grau de confiança superior, além daqueles 3 mencionados nos primeiros 2 *targets*, são "*NoSQL Injection - MongoDB*", que permite ao atacante explorar consultas inadequadas em bases de dados NoSQL, "*CSP: Wildcard Directive*", "*CSP: style-src unsafe-inline*" e "*Sub Resource Integrity Attribute Missing*".

Foram gerados alertas com o nível de criticidade mais alto relacionados com "*Path Traversal*", "*Remote File Inclusion*", "*SQL Injection*" e "*Server Side Request Forgery*" (tabela demonstrada no Anexo E).

4.2.1.6. Juice Shop

O "Juice Shop" é uma aplicação *web* propositadamente vulnerável, desenvolvida pela OWASP. Esta aplicação foi construída com tecnologias como "Node.js" e o "Angular" para o *frontend* e utiliza a "MongoDB" como base de dados.

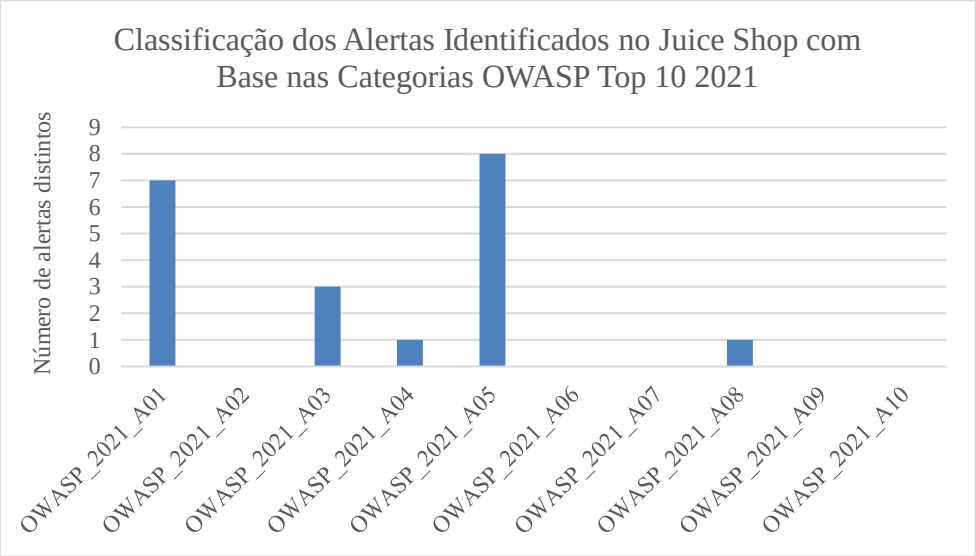

Figura 18 - Gráfico da Classificação dos Alertas Identificados no Juice Shop

Na Figura 18, podemos ver a distribuição dos alertas identificados no "Juice Shop" com base nas categorias do "OWASP Top 10". A maioria dos alertas encontrados, estão agrupados em duas categorias principais, a "OWASP_2021_A05 - *Security Misconfiguration*" e a "OWASP_2021_A01 - *Broken Access Control*", com 8 e 7 alertas distintos, respetivamente. Outras categorias com alertas presentes são "OWASP_2021_A03 – *Injection*", "OWASP_2021_A04 – *Insecure Design*" e "OWASP_2021_A08 - *Software and Data Integrity Failures*".

Pela observação da tabela no Anexo F, é possível concluir que o sistema conseguiu detetar alertas com o nível de confiança máximo, sendo eles, "*CORS Misconfiguration*", "*Content Security Policy (CSP) Header Not Set*", "*Information Disclosure - JWT in Browser localStorage*", "*Session ID in URL Rewrite*", referindo-se ao armazenamento do ID da sessão diretamente no URL que pode expor dados sensíveis e levar ao *hijacking* de sessões, e por último "*Strict-Transport-Security Header Not Set*".

Os alertas mais graves apresentados nesta tabela são "SQL Injection" e "Open Redirect".

4.2.1.7. Rest API

Desenvolvida pela *Acunetix*, a "Rest API" é uma aplicação *web* que utiliza as tecnologias "Apache", "PHP" e "MySQL". A distribuição dos alertas gerados pelas categorias está representada na Figura 19.

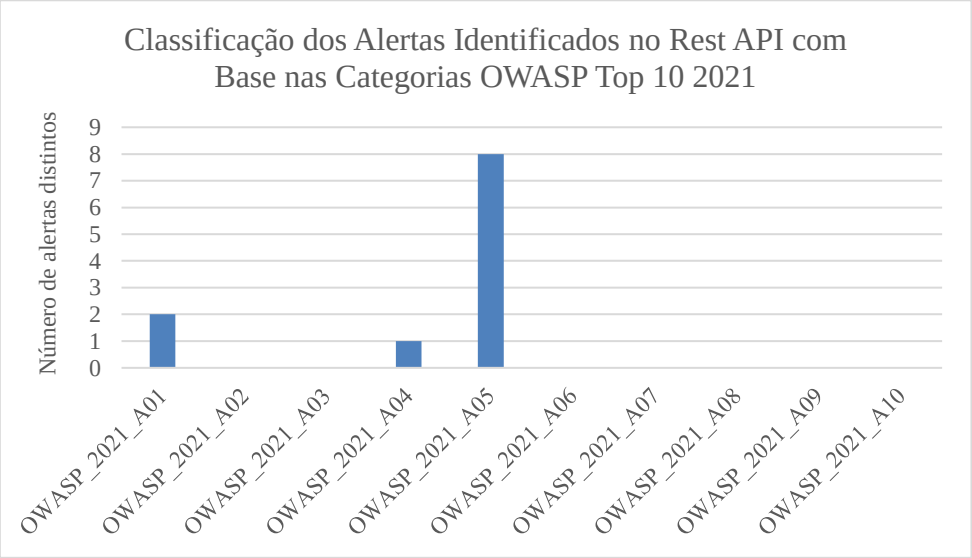

Figura 19 – Gráfico da Classificação dos Alertas Identificados no Rest API

As 3 categorias a que pertencem os riscos identificados no alvo "Rest API" são "OWASP_2021_A05 - *Security Misconfiguration*", "OWASP_2021_A01 - *Broken Access Control*" e "OWASP_2021_A04 – *Insecure Design*".

A tabela apresentada no Anexo G resultante da análise efetuada à aplicação "Rest API" mostra que "*Content Security Policy (CSP) Header Not Set*", "*Sub Resource Integrity Attribute Missing*", "*Server Leaks Version Information via "Server" HTTP Response Header Field*" são os alertas gerados com nível de confiança *High*. Por sua vez, é possível afirmar que não foram despoletados quaisquer alertas com o nível de risco *High*.

4.2.1.8. Security Tweets

A "Security Tweets" foi desenvolvida com o intuito de simular um ambiente típico de redes sociais. Utiliza o "Nginx" como servidor *web*, "Python" com o *framework* "Flask" no *backend*, e "CouchDB" como base de dados.

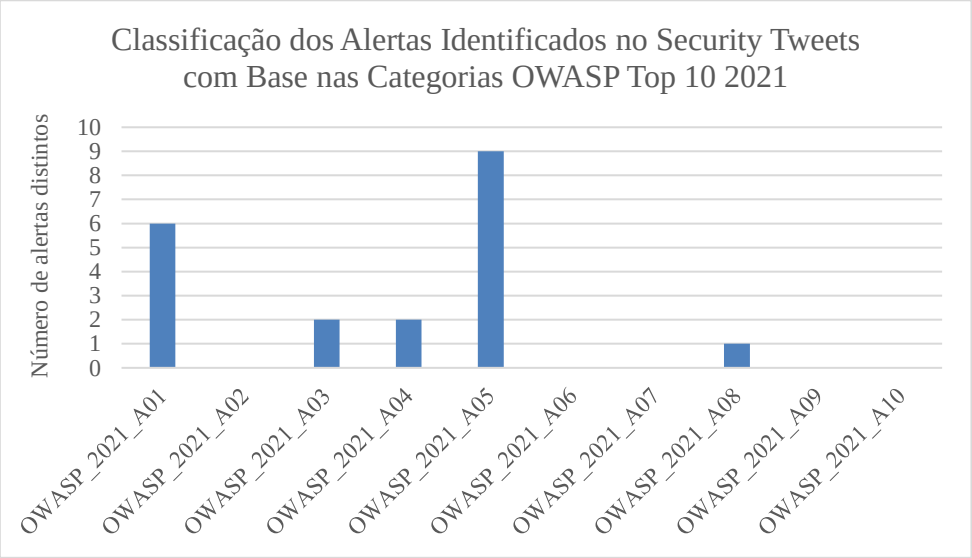

Figura 20 - Gráfico da Classificação dos Alertas Identificados no Security Tweets

Observando a Figura 20, é possível identificar as 5 categorias dos alertas despoletados durante a análise à aplicação "Security Tweets", que por ordem decrescente de número de alertas distintos, "OWASP_2021_A05 - *Security Misconfiguration*", "OWASP_2021_A01 - *Broken Access Control*", "OWASP_2021_A03 – *Injection*", "OWASP_2021_A04 – *Insecure Design*" e "OWASP_2021_A08 - *Software and Data Integrity Failures*".

Analisando a tabela demonstrada no Anexo H, correspondente à agregação dos alertas gerados durante o *scan* do alvo "Security Tweets", podemos observar 5 alertas com o nível de confiança mais elevado, sendo eles, "*NoSQL Injection - MongoDB*", "*CORS Misconfiguration*", "*Content Security Policy (CSP) Header Not Set*", "*Sub Resource Integrity Attribute Missing*" e "*Server Leaks Version Information via "Server" HTTP Response Header Field*". Adicionalmente foram registados mais 2 alertas, que requerem destaque, uma vez que possuem um nível de risco *High*, que são "SQL *Injection*" e "*Source Code Disclosure - File Inclusion*".

4.2.1.9. Duração das Análise nas Aplicações Vulneráveis

O tempo que cada análise demorou a ser executada foi registado e pode ser interessante observar. A duração das análises foi colocada apenas para demonstração e estudo, sendo que esta variável vai depender sempre da capacidade do *hardware* onde o sistema SIAAS vai estar a correr.

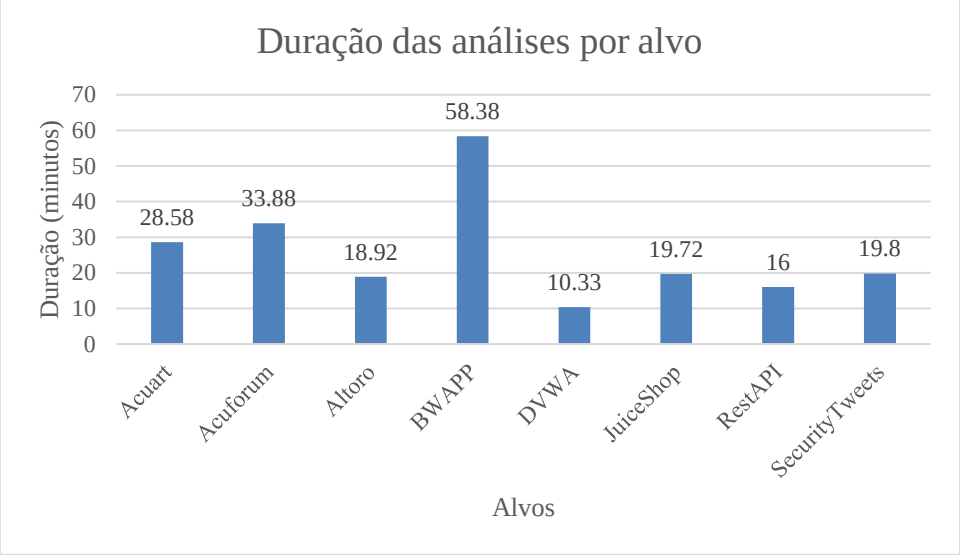

Figura 21 - Gráfico da Duração das análises por alvo

Analisando o gráfico da Figura 21, podemos concluir que a duração das análises executadas nas aplicações propositadamente vulneráveis foram todas relativamente rápidas, menos de uma hora, sendo que o alvo "bWAPP" possuiu a análise de maior duração, cerca de 58.38 minutos. A maioria das análises demorou menos de 20 minutos a terminar, sendo que a média de duração das análises às aplicações propositadamente vulneráveis foi de aproximadamente 25.7 minutos.

4.2.2. Aplicação Real – FenixIscteAuth

O segundo grupo de análise inclui uma aplicação *web* real em ambiente de qualidade. Foram solicitadas as devidas permissões para testar a aplicação do sistema Fénix do ISCTE-IUL, com o intuito de mostrar que o sistema é capaz de detetar vulnerabilidades em aplicações reais.

No caso da aplicação real, foi executada uma validação extra, para garantir que não eram expostas vulnerabilidades que pudessem vir a ser exploradas no futuro, tendo sido efetuados mais alguns testes que garantiram que apesar dos alertas despoletados a aplicação estava bem protegida.

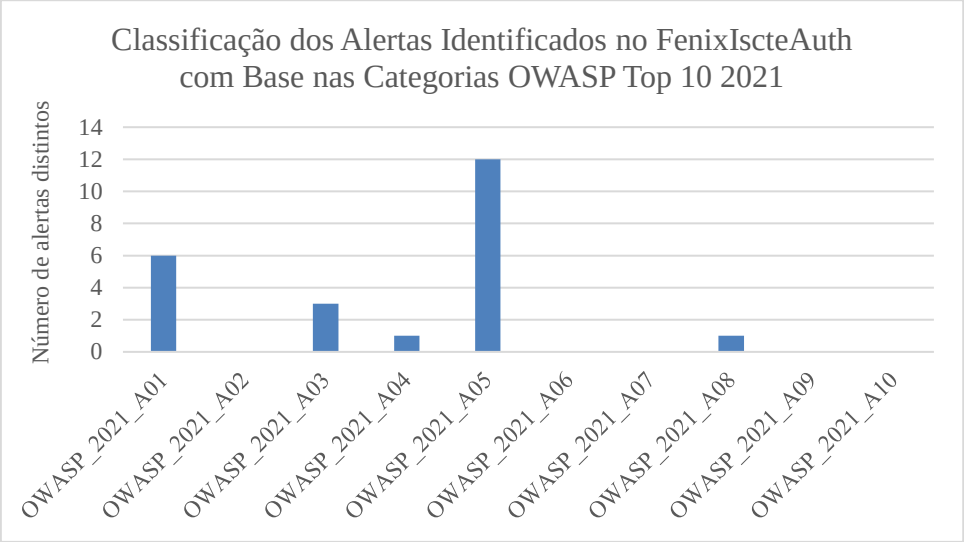

Figura 22 - Gráfico da Classificação dos Alertas Identificados no FenixIscteAuth

De acordo com a análise do gráfico de barras, presente na Figura 22, durante a análise do *site* do ISCTE foi possível identificar riscos de vulnerabilidades associados a 5 grupos de vulnerabilidades, consoante a categorização do "OWASP Top 10". Foram identificados riscos nas categorias de "OWASP_2021_A01 - *Broken Access Control*", "OWASP_2021_A03 – *Injection*", "OWASP_2021_A04 – *Insecure Design*", "OWASP_2021_A05 - *Security Misconfiguration*" e "OWASP_2021_A08 - *Software and Data Integrity Failures*".

Os alertas, com o nível de confiança superior, gerados de acordo com os riscos encontrados na aplicação do ISCTE são "*Content Security Policy (CSP) Header Not Set*", "*Sub Resource Integrity Attribute Missing*", "*Server Leaks Version Information via "Server" HTTP Response Header Field*" e "*Strict-Transport-Security Header Not Set*" (como ilustrado no Anexo I).

Em relação aos alertas com um nível de risco mais crítico, apenas foi despoletado o alerta "*Path Traversal*", no entanto o nível de confiança do mesmo é *Low*, ou seja, apesar do sistema detetar algum comportamento estranho, não conseguiu obter evidências suficientes nem explorar realmente a vulnerabilidade mencionada. Isto indica que existe a possibilidade da potencial vulnerabilidade não existir.

A validação efetuada serviu para concluir que apesar dos alertas gerados pelo sistema a aplicação é segura. Dando um exemplo, o alerta "*Content Security Policy (CSP) Header Not Set*" ocorre quando uma aplicação *web* não define o cabeçalho "*Content Security Policy (CSP)*" nas suas respostas HTTP. Foi verificado que o cabeçalho realmente não estava presente nas respostas, no entanto a aplicação utilizava outro mecanismo de segurança para controlar quais os recursos que podem ser carregados. Resumindo, apesar do alerta despoletado com confiança elevada, a aplicação estava corretamente protegida.

4.2.2.1. Comparação da duração entre os diferentes grupos de análise

Relativamente à duração da análise realizada ao alvo "FenixIscteAuth", esta teve um resultado bem superior aos tempos do primeiro grupo de aplicações testadas, sendo de 303.1 minutos, cerca de 5 vezes maior. Esta diferença pode ser justificada pelo facto de que o "FenixIscteAuth" é uma aplicação maior e com uma lógica de autenticação e permissões complexas, o que naturalmente requer mais tempo de processamento para uma análise exaustiva. As aplicações propositadamente vulneráveis, por outro lado, são mais simples e projetadas para facilitar a deteção de falhas, resultando em tempos de análise mais curtos.

4.3. Análise dos Riscos Identificados

Resumindo, os riscos identificados foram categorizados de acordo com o "OWASP Top 10", com ênfase nas categorias que, segundo o Estado da Literatura (Capítulo 2), representam as ameaças mais críticas para aplicações *web* modernas. Os riscos mais recorrentes identificados nas análises executadas pertencem às seguintes categorias:

- OWASP_2021_A05: Security Misconfiguration
- OWASP_2021_A01: Broken Access Control
- OWASP_2021_A03: Injection

Estas categorias estão entre as mais graves no OWASP Top 10 de 2021, uma vez que "*Security Misconfiguration*" se refere a uma vasta gama de falhas que podem comprometer a segurança da aplicação, enquanto "*Broken Access Control*" diz respeito a falhas que permitem a utilizadores maliciosos aceder a recursos ou funcionalidades não autorizadas. "*Injection*", por sua vez, continua a ser uma das vulnerabilidades mais exploradas, onde dados não confiáveis são inseridos num interpretador (como SQL, NoSQL, LDAP) que executa comandos maliciosos.

Adicionalmente, o ZAP também gera alertas meramente informativos, sobre a construção da aplicação e sobre possíveis melhorias. Estes alertas foram excluídos desta análise devido à sua natureza.

Os resultados indicam que o sistema de deteção de vulnerabilidades é eficaz na identificação de falhas críticas e na sua categorização de acordo com os categorias do OWASP. No entanto, é importante salientar que em dois dos nove alvos analisados, o sistema não foi capaz de realizar análises autenticadas, uma vez que não conseguiu identificar adequadamente os campos de *login*. Este fator constitui uma limitação no que respeita à cobertura total de análise de aplicações que requerem autenticação. No entanto, devido ao facto da Comunidade *open-source* responsável pelo ZAP ser muito ativa e estar continuamente a lançar atualizações e melhorias para o seu *scanner*, este tipo de limitações não são definitivas e a ferramenta ao longo do tempo tenderá a ficar cada vez mais completa.

4.4. Desempenho Computacional

Relativamente ao desempenho computacional, os testes confirmaram que as configurações mínimas especificadas (6 GB de RAM, 4 processadores) são adequadas para a execução do sistema sem falhas. Durante as análises, observou-se que a utilização dos recursos computacionais era considerável durante a realização das análises, no entanto isto pode ser explicado pelo facto, da ferramenta ZAP se adaptar aos recursos computacionais do sistema que o está a utilizar. Estas configurações têm em conta a possível utilização deste sistema em sistema com reduzida capacidade computacional, como é o caso de uma "Raspberry Pi".

Quando as análises foram realizadas numa máquina virtual com configurações mais modestas (4 GB de RAM e 4 processadores), ocorreram falhas devido à falta de recursos na análise a alguns alvos. Esta observação reforça a necessidade de utilizar configurações mínimas adequadas, especialmente em testes sobre aplicações de maior complexidade, como o "FenixIscteAuth".

Para além destas configurações, o utilizador consegue configurar determinados campos do plano de automação, como por exemplo número de *browsers* a serem inicializados durante o processo de *crawling* e assim determinar a profundidade e velocidade da análise. Posto isto, também é possível o utilizador controlar a utilização dos recursos computacionais através da linha de comandos.

CAPÍTULO 5

Conclusões

Este capítulo final tem como objetivo sintetizar os principais resultados alcançados ao longo deste trabalho, destacando as conclusões retiradas da investigação realizada. Também serão mencionadas as limitações do sistema e apresentadas sugestões de trabalho futuro, com o intuito de melhorar as funcionalidades do sistema.

5.1. Conclusões

A complexidade da segurança de aplicações *web* é evidenciada pela diversidade de vulnerabilidades que podem ser exploradas pelos atacantes. A revisão de literatura destaca a evolução das aplicações *web*, desde os seus modelos iniciais até aos complexos sistemas interativos atuais. Este progresso tecnológico resultou num aumento significativo no número de vulnerabilidades identificadas em cada aplicação. Com a elaboração desta dissertação, para além de ter sido possível identificar e entender quais são os principais riscos e vulnerabilidades presentes nas aplicações *web*, estudou-se e elaborou-se um sistema que promove a automação da deteção de potenciais vulnerabilidades em aplicações *web*.

Com base nos resultados obtidos ao longo deste trabalho de investigação, é possível afirmar que a elaboração de uma plataforma utilizando uma única ferramenta *open-source* para a automação da deteção de vulnerabilidades em aplicações *web*, sem exigir um elevado nível de *expertise* por parte do utilizador, é viável. Através da aplicação dessa ferramenta, foi desenvolvido um fluxo de trabalho automatizado, isto é, as diversas etapas sequenciais do sistema, que permitem a deteção de vulnerabilidades, que fornece os resultados de forma organizada e intuitiva ao utilizador.

O sistema desenvolvido para a automação da deteção de vulnerabilidades em aplicações *web* demonstrou ser uma ferramenta valiosa e eficaz, especialmente considerando o seu objetivo de facilitar a deteção de vulnerabilidades sem requerer conhecimentos avançados de cibersegurança. Com base nos testes realizados, o sistema provou ser capaz de identificar e categorizar vulnerabilidades críticas de acordo com o referencial “OWASP Top 10” e na produção de resultados interpretáveis por utilizadores não especializados.

Além disso, os testes realizados numa aplicação real, o FenixIscteAuth, validaram a eficácia do sistema em cenários reais, demonstrando que é capaz de lidar com processos de autenticação e identificar falhas de segurança em aplicações complexas. Contudo, o sistema apresentou limitações no que respeita à realização de *scans* autenticados em determinado tipo de aplicações, o que constitui um ponto a melhorar.

Com este trabalho espera-se facilitar e melhorar a segurança aplicacional, permitindo que empresas ou indivíduos possam, de forma gratuita e simples, estar mais informados sobre as vulnerabilidades existentes nas suas aplicações *web*. Através da utilização de uma única ferramenta *open-source*, foi possível criar um sistema automatizado que torna o processo de deteção de vulnerabilidades mais acessível, mesmo para utilizadores com pouca experiência técnica. Esta solução permite que os utilizadores testem as suas aplicações, identifiquem potenciais falhas de segurança e tomem medidas corretivas, sem a necessidade de recorrer a soluções comerciais complexas ou dispendiosas, promovendo assim uma maior segurança no desenvolvimento e implementação de aplicações *web*.

5.2. Limitações do sistema

Embora o sistema tenha alcançado os objetivos principais propostos, algumas limitações foram identificadas ao longo dos testes, nomeadamente:

- **Incapacidade de realizar análises autenticadas em todas as aplicações:** Em dois dos alvos testados, o sistema não conseguiu realizar análises de vulnerabilidades devidamente autenticados devido à dificuldade em identificar corretamente os campos de autenticação. Este problema surge da diversidade de mecanismos de autenticação utilizados pelas aplicações *web* modernas, que podem variar significativamente em termos de implementação e complexidade. A integração de outras ferramentas de análise de vulnerabilidades poderá ajudar a resolver esta limitação.
- **Necessidade de alguma capacidade computacional:** Como referido anteriormente o sistema requer alguma capacidade computacional aquando da execução dos planos de automação a partir da ferramenta ZAP. O utilizador tem a capacidade de controlar o nível de profundidade da análise, o que impacta diretamente na capacidade de computação exigida pelo sistema. Ainda assim esta particularidade aumenta relativamente a necessidade computacional em relação ao sistema desenvolvido anteriormente.

- **Tamanho do documento JSON a guardar na MongoDB**: Esta não é bem uma limitação do sistema desenvolvido, mas sim uma restrição externa, uma vez que o "MongoDB", utilizado como base de dados para o armazenamento dos resultados das análises, impõe um limite máximo de 16 MB para o tamanho de um documento individual. Como resultado, em aplicações muito extensas e complexas, este tamanho pode ser excedido e os resultados não serão armazenados na base de dados. No entanto, o relatório elaborado pelo ZAP referente ao respetivo alvo ficará disponível na mesma. Uma possível solução passaria por dividir o documento dos resultados da análise. Esta situação ocorreu apenas uma vez durante toda a fase de testes, não tendo sido uma preocupação.

5.3. Propostas de trabalhos futuros

Tendo em consideração o trabalho desenvolvido e as suas limitações, sugere-se que trabalhos futuros se concentrem nos seguintes aspetos:

- **Integração com outras ferramentas de análise de vulnerabilidades:** A integração com ferramentas como "Nikto" ou "Arachni" permitiria a realização de uma análise mais completa e precisa, ampliando o leque de vulnerabilidades detetadas pelo sistema.
- **Tratamento de resultados personalizado:** Esta proposta de trabalho futuro consistiria na geração de relatórios próprios, ou personalizar a estrutura dos resultados de acordo com as preferências dos utilizadores, promovendo uma gestão mais direcionada e adaptada a diferentes contextos de segurança e ao próprio utilizador. Esta proposta poderá estar diretamente relacionada com a proposta de trabalho futuro referida em cima, na medida que para conseguir juntar os diferentes resultados obtidos em diferentes ferramentas de análise de vulnerabilidades, poderá ser necessário uniformizar os resultados de ambas as ferramentas para que seja possível retirar conclusões de valor dos mesmos.

Referências Bibliográficas

Al Anhar, A., & Suryanto, Y. (2021). Evaluation of Web Application Vulnerability Scanner for Modern Web Application. *ICAICST 2021 - 2021 International Conference on Artificial Intelligence and Computer Science Technology*, 200–204. https://doi.org/10.1109/ICAICST53116.2021.9497831

Alazmi, S., & De Leon, D. C. (2023). Customizing OWASP ZAP: A Proven Method for Detecting SQL Injection Vulnerabilities. *Proceedings - 2023 IEEE 9th International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, and IEEE International Conference on Intelligent Data and Security, BigDataSecurity-HPSC-IDS 2023*, 102–106. https://doi.org/10.1109/BigDataSecurity-HPSC-IDS58521.2023.00028

Al-Kahla, W., Shatnawi, A. S., & Taqieddin, E. (2021). A Taxonomy of Web Security Vulnerabilities. *2021 12th International Conference on Information and Communication Systems, ICICS 2021*, 424–429. https://doi.org/10.1109/ICICS52457.2021.9464576

Althunayyan, M., Saxena, N., Li, S., & Gope, P. (2022). Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities. *Electronics (Switzerland), 11*(13). https://doi.org/10.3390/electronics11132049

Badri, A. M. Al, & Alouneh, S. (2023). Detection of Malicious Requests to Protect Web Applications and DNS Servers Against SQL Injection Using Machine Learning. *2023 International Conference on Intelligent Computing, Communication, Networking and Services, ICCNS 2023*, 5–11. https://doi.org/10.1109/ICCNS58795.2023.10193085

Beba, S., Karlsen, M. M., Li, J., & Zhang, B. (2021). Critical Understanding of Security Vulnerability Detection Plugin Evaluation Reports. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2021-December*, 275–284. https://doi.org/10.1109/APSEC53868.2021.00035

Dhivya, K., Kannagi, V., Rajkumar, M., Chandra, I., & Ganesh, S. J. (2022). An Effective Server-Side Attack Identification and Prevention Scheme using Logical Query Processing Strategy. *Proceedings of the International Conference on Electronics and Renewable Systems, ICEARS 2022*, 945–951. https://doi.org/10.1109/ICEARS53579.2022.9752277

Djeki, E., Degila, J., Bondiombouy, C., & Alhassan, M. H. (2021). Security Issues in Digital Learning Spaces. *2021 IEEE International Conference on Computing, ICOCO 2021*, 71–77. https://doi.org/10.1109/ICOCO53166.2021.9673575

Doupé, A., Cavedon, L., Kruegel, C., & Vigna, G. (2012). *Enemy of the State: A State-Aware Black-Box Vulnerability Scanner Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner*. https://www.researchgate.net/publication/235220328

El Idrissi, S., Berbiche, N., Guerouate, F., & Sbihi, M. (2017). Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities. In *International Journal of Applied Engineering Research* (Vol. 12). http://www.ripublication.com

Eshete, B., Villafiorita, A., Weldemariam, K., & Zulkernine, M. (2013). Confeagle: Automated analysis of configuration vulnerabilities in web applications. *Proceedings - 7th International Conference on Software Security and Reliability, SERE 2013*, 188–197. https://doi.org/10.1109/SERE.2013.30

Fonseca, J., Vieira, M., & Madeira, H. (2014). Evaluation of Web Security Mechanisms Using Vulnerability & Attack Injection. *IEEE Transactions on Dependable and Secure Computing*, *11*(5), 440–453. https://doi.org/10.1109/TDSC.2013.45

Goe, D. (2015). *Detection of Web Appication Vulnerability Based on RUP Model*.

Gupta, S., & Gupta, B. B. (2017). Detection, Avoidance, and Attack Pattern Mechanisms in Modern Web Application Vulnerabilities. *International Journal of Cloud Applications and Computing*, *7*(3), 1–43. https://doi.org/10.4018/ijcac.2017070101

Jain, T., & Jain, N. (2019). Framework for Web Application Vulnerability Discovery and Mitigation by Customizing Rules through ModSecurity. *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN) : 7-8 March 2019, Amity School of Engineering and Technology, Noida, India*.

Khoury, N., Zavarsky, P., Lindskog, D., & Ruhl, R. (2011). An Analysis of Black-Box Web Application Security Scanners against Stored SQL Injection. *2011 IEEE Third Int'l Conference on Privacy, Security, Risk, and Trust, and 2011 IEEE Third Int'l Conference on Social Computing, Tijuana, Mexico, 09.10-11.10.2011*.

Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering - A systematic literature review. In *Information and Software Technology* (Vol. 51, Issue 1, pp. 7–15). https://doi.org/10.1016/j.infsof.2008.09.009

Koswara, K. J., & Asnar, Y. D. W. (2019). Improving Vulnerability Scanner Performance in Detecting AJAX Application Vulnerabilities. *Proceedings of 2019 International Conference on Data and Software Engineering (ICoDSE) : Gedung Konferensi Universitas Tanjungpura, Pontianak, Indonesia, November 13th-14th, 2019*.

Lavens, E., Philippaerts, P., & Joosen, W. (2022, August 23). A Quantitative Assessment of the Detection Performance of Web Vulnerability Scanners. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3538969.3544416

Lei, X., Qu, J., Yao, G., Chen, J., & Shen, X. (2020). Design and Implementation of an Automatic Scanning Tool of SQL Injection Vulnerability Based on Web Crawler. *Advances in Intelligent Systems and Computing*, *895*, 481–488. https://doi.org/10.1007/978-3-030-16946-6_38

Li, X., & Xue, Y. (2011). *A Survey on Web Application Security*.

Mburano, B., & Si, W. (2018). Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark. *ICSEng 2018 : 26th International Conference on Systems Engineering : Conference Proceedings : December 18th - 20th, 2018, University of Technology Sydney, Australia*.

MITRE. (2023). *2023 CWE Top 25 Most Dangerous Software Weaknesses*.

Mohamed, T. S. (2020). Analytical View of Web Security and Sophisticated Ways to Improve Web Security. *Journal of Physics: Conference Series*, *1530*(1). https://doi.org/10.1088/1742-6596/1530/1/012023

Mohammed Draib, A., Bakar Md Sultan, A., Azim Abd Ghani, A. B., & Zulzalil, H. (2018). Comparison of Security Testing Approaches for Detection of SQL Injection Vulnerabilities. In *International Journal of Engineering & Technology* (Vol. 7, Issue 1). www.sciencepubco.com/index.php/IJET

Muralidharan, M., Babu, K. B., & Sujatha, G. (2023). W3BnNr: An Automated tool for information gathering, vulnerability scanning, attacking and reporting for injection attacks on web application. *Proceedings of the ACCTHPA 2023 - Conference on Advanced Computing and Communication Technologies for High Performance Applications*. https://doi.org/10.1109/ACCTHPA57160.2023.10083380

OWASP. (2021). *OWASP Top 10*. https://owasp.org/Top10/

Parvez, M., Zavarsky, P., & Khoury, N. (2015). *Analysis of Effectiveness of Black-Box Web Application Scanners in Detection of Stored SQL Injection and Stored XSS Vulnerabilities*.

Pathirathna, P. P. W., Ayesha, V. A. I., Imihira, W. A. T., Wasala, W. M. J. C., Kodagoda, N., & Edirisinghe, E. A. T. D. (2017). *Security Testing as a Service with Docker Containerization*.

Peffers, K., & Rossi, M. (2006). *The design science research process: a model for producing and presenting information systems research*. https://www.researchgate.net/publication/238077290

Pereira, R., & Serrano, J. (2020). A review of methods used on IT maturity models development: A systematic literature review and a critical analysis. *Journal of Information Technology*, *35*(2), 161–178. https://doi.org/10.1177/0268396219886874

Pfrang, S., Borcherding, A., Meier, D., & Beyerer, J. (2019). Automated security testing for web applications on industrial automation and control systems. *At-Automatisierungstechnik*, *67*(5), 383–401. https://doi.org/10.1515/auto-2019-0021

Portugal. (2009). Lei n.º 109_2009 - Diário da República n.º 179_2009, Série I de 2009-09-15. *Diário Da República, 1.ª Série, n.º 179*, 6740–6750.

Qasaimeh, M., Shamlawi, A., & Khairallah, T. (2018). BLACK BOX EVALUATION OF WEB APPLICATION SCANNERS: STANDARDS MAPPING APPROACH. *Journal of Theoretical and Applied Information Technology*, *31*(14). www.jatit.org

Saleh, A. Z. M., Rozali, N. A., Buja, A. G., Jalil, K. A., Ali, F. H. M., & Rahman, T. F. A. (2015). A Method for Web Application Vulnerabilities Detection by Using Boyer-Moore String Matching Algorithm. *Procedia Computer Science*, *72*, 112–121. https://doi.org/10.1016/j.procs.2015.12.111

Seara, J. (2023). *Intelligent System for Automation of Security Audits - SIAAS*. ISCTE-IUL.

Seng, L. K., Ithnin, N., & Shaid, S. Z. M. (2019). The preparation of cross-site scripting in automated web application vulnerability assessment: The quantitative analysis. *International Journal of Advanced Trends in Computer Science and Engineering*, *8*(1.6 S1), 57–63. https://doi.org/10.30534/ijatcse/2019/0981.62019

Shahid, J., Hameed, M. K., Javed, I. T., Qureshi, K. N., Ali, M., & Crespi, N. (2022). A Comparative Study of Web Application Security Parameters: Current Trends and Future Directions. *Applied Sciences (Switzerland)*, *12*(8). https://doi.org/10.3390/app12084077

Shar, L. K., & Tan, H. B. K. (2013). Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. *Information and Software Technology*, *55*(10), 1767–1780. https://doi.org/10.1016/j.infsof.2013.04.002

Singh, N., Meherhomji, V., & Chandavarkar, B. R. (2020). *Automated versus Manual Approach of Web Application Penetration Testing*. https://someattackerwebsite.com/x?=document.

Sonmez, F. O., & Kilic, B. G. (2021). Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results. *IEEE Access*, *9*, 25858–25884. https://doi.org/10.1109/ACCESS.2021.3057044

Syed, A. R., Veeragandham, S., Raheem, S. A., & Gillela, K. (2020). A survey on Major Problems and solutions of Top 10 Web Application Security. *International Journal of Advanced Science and Technology*, *29*(3s), 1–07. https://www.researchgate.net/publication/374814078

Truong, D., Tran, D., Nguyen, L., Mac, H., Tran, H. A., & Bui, T. (2019). Detecting web attacks using stacked denoising autoencoder and ensemble learning methods. *ACM International Conference Proceeding Series*, 267–272. https://doi.org/10.1145/3368926.3369715

Tyagi, S., Rachna. Manav, & Kumar, K. (2019). Evaluation of Static Web Vulnerability Analysis Tools. *2019 International Conference on Robotics and Automation (ICRA) : Palais Des Congres de Montreal, Montreal, Canada, May 20-24, 2019.*

Verma, A., Khatana, A., & Chaudhary, S. (2017). A Comparative Study of Black Box Testing and White Box Testing. *Article in International Journal of Computer Sciences and Engineering*. https://doi.org/10.26438/ijcse/v5i12.301304

ZAP DEV Team. (2024). *ZAP Site*. https://www.zaproxy.org/

Anexos

Anexo A – Tabela de alertas Acuart

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Cross Site Scripting (DOM Based)	High	High	8	OWASP_2021_A03
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	High	Medium	2	OWASP_2021_A03
Advanced SQL Injection - MySQL >= 5.0.12 AND time-based blind (SELECT)	High	Medium	2	OWASP_2021_A03
Advanced SQL Injection - MySQL UNION query (NULL) - 1 to 10 columns	High	Medium	1	OWASP_2021_A03
Cross-Domain Misconfiguration - Adobe - Read	High	Medium	1	OWASP_2021_A05
SQL Injection	High	Medium	2	OWASP_2021_A03
SQL Injection - MySQL	High	Medium	2	OWASP_2021_A03
Content Security Policy (CSP) Header Not Set	Medium	High	16	OWASP_2021_A05
Anti-CSRF Tokens Check	Medium	Medium	16	OWASP_2021_A05
Backup File Disclosure	Medium	Medium	2	OWASP_2021_A05
HTTP Only Site	Medium	Medium	1	OWASP_2021_A05
Missing Anti-clickjacking Header	Medium	Medium	15	OWASP_2021_A05
Source Code Disclosure - SQL	Medium	Medium	2	OWASP_2021_A05
Absence of Anti-CSRF Tokens	Medium	Low	19	OWASP_2021_A01
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	High	19	OWASP_2021_A05
Cookie No HttpOnly Flag	Low	Medium	1	OWASP_2021_A05
Cookie without SameSite Attribute	Low	Medium	1	OWASP_2021_A01
Permissions Policy Header Not Set	Low	Medium	16	OWASP_2021_A01
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	Medium	16	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	18	OWASP_2021_A05

Anexo B – Tabela de alertas Acuforum

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Cross Site Scripting (DOM Based)	High	High	2	OWASP_2021_A03
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	High	Medium	3	OWASP_2021_A03
Cross Site Scripting (Reflected)	High	Medium	1	OWASP_2021_A03
External Redirect	High	Medium	1	OWASP_2021_A03
Path Traversal	High	Medium	2	OWASP_2021_A01
SQL Injection - MsSQL	High	Medium	1	OWASP_2021_A03
Source Code Disclosure - File Inclusion	High	Medium	1	OWASP_2021_A05
Content Security Policy (CSP) Header Not Set	Medium	High	42	OWASP_2021_A05
Anti-CSRF Tokens Check	Medium	Medium	6	OWASP_2021_A05
Bypassing 403	Medium	Medium	9	OWASP_2021_A01
HTTP Only Site	Medium	Medium	1	OWASP_2021_A05
Integer Overflow Error	Medium	Medium	4	OWASP_2021_A03
Missing Anti-clickjacking Header	Medium	Medium	41	OWASP_2021_A05
Absence of Anti-CSRF Tokens	Medium	Low	37	OWASP_2021_A01
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	High	63	OWASP_2021_A05
Cookie No HttpOnly Flag	Low	Medium	1	OWASP_2021_A05
Cookie without SameSite Attribute	Low	Medium	1	OWASP_2021_A01
Permissions Policy Header Not Set	Low	Medium	45	OWASP_2021_A01
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	Medium	63	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	59	OWASP_2021_A05
Cookie Slack Detector	Low	Low	8	OWASP_2021_A05
Dangerous JS Functions	Low	Low	1	OWASP_2021_A04

Anexo C – Tabela de alertas Altoro

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Cross Site Scripting (DOM Based)	High	High	14	OWASP_2021_A03
Cross Site Scripting (Reflected)	High	Medium	2	OWASP_2021_A03
Cookie Slack Detector	High	Medium	2	OWASP_2021_A03
Content Security Policy (CSP) Header Not Set	Medium	High	176	OWASP_2021_A05
Sub Resource Integrity Attribute Missing	Medium	High	1	OWASP_2021_A05
Anti-CSRF Tokens Check	Medium	Medium	11	OWASP_2021_A05
Insecure HTTP Method - PUT	Medium	Medium	20	OWASP_2021_A05
Missing Anti-clickjacking Header	Medium	Medium	65	OWASP_2021_A05
Relative Path Confusion	Medium	Medium	15	OWASP_2021_A05
Source Code Disclosure - SQL	Medium	Medium	1	OWASP_2021_A05
Web Cache Deception	Medium	Medium	4	OWASP_2021_A05
Absence of Anti-CSRF Tokens	Medium	Low	177	OWASP_2021_A01
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	High	219	OWASP_2021_A05
Cookie without SameSite Attribute	Low	Medium	11	OWASP_2021_A01
Cross-Domain JavaScript Source File Inclusion	Low	Medium	1	OWASP_2021_A08
Permissions Policy Header Not Set	Low	Medium	178	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	105	OWASP_2021_A05
Cookie Slack Detector	Low	Low	11	OWASP_2021_A05
Dangerous JS Functions	Low	Low	1	OWASP_2021_A04
Timestamp Disclosure - Unix	Low	Low	2	OWASP_2021_A01

Anexo D – Tabela de alertas bWAPP

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Content Security Policy (CSP) Header Not Set	Medium	High	14	OWASP_2021_A05
Missing Anti-clickjacking Header	Medium	Medium	14	OWASP_2021_A05
Proxy Disclosure	Medium	Medium	15	OWASP_2021_A05
Absence of Anti-CSRF Tokens	Medium	Low	29	OWASP_2021_A01
Strict-Transport-Security Header Not Set	Low	High	36	OWASP_2021_A05
Cookie No HttpOnly Flag	Low	Medium	6	OWASP_2021_A05
Cookie Without Secure Flag	Low	Medium	6	OWASP_2021_A05
Cookie without SameSite Attribute	Low	Medium	6	OWASP_2021_A01
Permissions Policy Header Not Set	Low	Medium	15	OWASP_2021_A01
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	Medium	19	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	34	OWASP_2021_A05

Anexo E – Tabela de alertas DVWA

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Cross Site Scripting (DOM Based)	High	High	2	OWASP_2021_A03
NoSQL Injection - MongoDB	High	High	1	OWASP_2021_A03
Path Traversal	High	Medium	1	OWASP_2021_A01
Remote File Inclusion	High	Medium	1	OWASP_2021_A03
SQL Injection	High	Medium	1	OWASP_2021_A03
Server Side Request Forgery	High	Medium	1	OWASP_2021_A10
CSP: Wildcard Directive	Medium	High	1	OWASP_2021_A05
CSP: style-src unsafe-inline	Medium	High	1	OWASP_2021_A05
Content Security Policy (CSP) Header Not Set	Medium	High	27	OWASP_2021_A05
Sub Resource Integrity Attribute Missing	Medium	High	1	OWASP_2021_A05
Anti-CSRF Tokens Check	Medium	Medium	11	OWASP_2021_A05
Application Error Disclosure	Medium	Medium	4	OWASP_2021_A05
Directory Browsing	Medium	Medium	8	OWASP_2021_A01
HTTP Only Site	Medium	Medium	1	OWASP_2021_A05
Missing Anti-clickjacking Header	Medium	Medium	28	OWASP_2021_A05
Relative Path Confusion	Medium	Medium	6	OWASP_2021_A05
Source Code Disclosure - SQL	Medium	Medium	4	OWASP_2021_A05
XSLT Injection	Medium	Medium	3	OWASP_2021_A03
Absence of Anti-CSRF Tokens	Medium	Low	13	OWASP_2021_A01
Parameter Tampering	Medium	Low	3	OWASP_2021_A04
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	High	43	OWASP_2021_A05
Cookie No HttpOnly Flag	Low	Medium	4	OWASP_2021_A05
Cookie without SameSite Attribute	Low	Medium	4	OWASP_2021_A01
Cross-Domain JavaScript Source File Inclusion	Low	Medium	1	OWASP_2021_A08
Information Disclosure - Debug Error Messages	Low	Medium	4	OWASP_2021_A01
Permissions Policy Header Not Set	Low	Medium	30	OWASP_2021_A01
Private IP Disclosure	Low	Medium	2	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	41	OWASP_2021_A05
Cookie Slack Detector	Low	Low	1	OWASP_2021_A05
Dangerous JS Functions	Low	Low	1	OWASP_2021_A04

Anexo F – Tabela de alertas Juice Shop

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	High	Medium	5	OWASP_2021_A03
Open Redirect	High	Medium	1	OWASP_2021_A03
SQL Injection - SQLite	High	Medium	2	OWASP_2021_A03
CORS Misconfiguration	Medium	High	10	OWASP_2021_A01
Content Security Policy (CSP) Header Not Set	Medium	High	70	OWASP_2021_A05
Information Disclosure - JWT in Browser localStorage	Medium	High	8	N/A
Session ID in URL Rewrite	Medium	High	263	OWASP_2021_A01
Cross-Domain Misconfiguration	Medium	Medium	50	OWASP_2021_A01
Missing Anti-clickjacking Header	Medium	Medium	63	OWASP_2021_A05
Web Cache Deception	Medium	Medium	6	OWASP_2021_A05
Strict-Transport-Security Header Not Set	Low	High	376	OWASP_2021_A05
Application Error Disclosure	Low	Medium	4	OWASP_2021_A05
Cross-Domain JavaScript Source File Inclusion	Low	Medium	6	OWASP_2021_A08
Deprecated Feature Policy Header Set	Low	Medium	11	OWASP_2021_A01
HTTPS Content Available via HTTP	Low	Medium	8	OWASP_2021_A05
Permissions Policy Header Not Set	Low	Medium	63	OWASP_2021_A01
Private IP Disclosure	Low	Medium	1	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	263	OWASP_2021_A05
Dangerous JS Functions	Low	Low	2	OWASP_2021_A04
Full Path Disclosure	Low	Low	4	OWASP_2021_A05
Timestamp Disclosure - Unix	Low	Low	1905	OWASP_2021_A01

Anexo G – Tabela de alertas RestAPI

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Content Security Policy (CSP) Header Not Set	Medium	High	17	OWASP_2021_A05
Sub Resource Integrity Attribute Missing	Medium	High	2	OWASP_2021_A05
Anti-CSRF Tokens Check	Medium	Medium	1	OWASP_2021_A05
HTTP Only Site	Medium	Medium	1	OWASP_2021_A05
Missing Anti-clickjacking Header	Medium	Medium	3	OWASP_2021_A05
Vulnerable JS Library	Medium	Medium	3	OWASP_2017_A09
Web Cache Deception	Medium	Medium	7	OWASP_2021_A05
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	High	53	OWASP_2021_A05
Permissions Policy Header Not Set	Low	Medium	48	OWASP_2021_A01
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)	Low	Medium	18	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	38	OWASP_2021_A05
Dangerous JS Functions	Low	Low	1	OWASP_2021_A04

Anexo H – Tabela de alertas SecurityTweets

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
NoSQL Injection - MongoDB	High	High	1	OWASP_2021_A03
Advanced SQL Injection - AND boolean-based blind - WHERE or HAVING clause	High	Medium	1	OWASP_2021_A03
Source Code Disclosure - File Inclusion	High	Medium	1	OWASP_2021_A05
CORS Misconfiguration	Medium	High	4	OWASP_2021_A01
Content Security Policy (CSP) Header Not Set	Medium	High	24	OWASP_2021_A05
Sub Resource Integrity Attribute Missing	Medium	High	24	OWASP_2021_A05
Anti-CSRF Tokens Check	Medium	Medium	3	OWASP_2021_A05
Bypassing 403	Medium	Medium	7	OWASP_2021_A01
Cross-Domain Misconfiguration	Medium	Medium	2	OWASP_2021_A01
HTTP Only Site	Medium	Medium	1	OWASP_2021_A05
Missing Anti-clickjacking Header	Medium	Medium	21	OWASP_2021_A05
Vulnerable JS Library	Medium	Medium	1	OWASP_2017_A09
Absence of Anti-CSRF Tokens	Medium	Low	2	OWASP_2021_A01
Exponential Entity Expansion (Billion Laughs Attack)	Medium	Low	1	OWASP_2021_A04
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	High	41	OWASP_2021_A05
Cookie No HttpOnly Flag	Low	Medium	1	OWASP_2021_A05
Cookie without SameSite Attribute	Low	Medium	1	OWASP_2021_A01
Cross-Domain JavaScript Source File Inclusion	Low	Medium	11	OWASP_2021_A08
Permissions Policy Header Not Set	Low	Medium	29	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	36	OWASP_2021_A05
Dangerous JS Functions	Low	Low	1	OWASP_2021_A04

Anexo I – Tabela de alertas FenixIscteAuth

Alert Name	Risk Level	Confidence	Incidences	OWASP Category
Path Traversal	High	Low	3	OWASP_2021_A01
Content Security Policy (CSP) Header Not Set	Medium	High	99	OWASP_2021_A05
Sub Resource Integrity Attribute Missing	Medium	High	107	OWASP_2021_A05
Anti-CSRF Tokens Check	Medium	Medium	1	OWASP_2021_A05
Buffer Overflow	Medium	Medium	12	OWASP_2021_A03
Bypassing 403	Medium	Medium	10	OWASP_2021_A01
Format String Error	Medium	Medium	11	OWASP_2021_A03
Integer Overflow Error	Medium	Medium	11	OWASP_2021_A03
Missing Anti-clickjacking Header	Medium	Medium	62	OWASP_2021_A05
Proxy Disclosure	Medium	Medium	7	OWASP_2021_A05
Relative Path Confusion	Medium	Medium	16	OWASP_2021_A05
Vulnerable JS Library	Medium	Medium	4	OWASP_2017_A09
Absence of Anti-CSRF Tokens	Medium	Low	2	OWASP_2021_A01
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	High	232	OWASP_2021_A05
Strict-Transport-Security Header Not Set	Low	High	228	OWASP_2021_A05
Application Error Disclosure	Low	Medium	1	OWASP_2021_A05
Cookie No HttpOnly Flag	Low	Medium	1	OWASP_2021_A05
Cookie Without Secure Flag	Low	Medium	50	OWASP_2021_A05
Cookie without SameSite Attribute	Low	Medium	50	OWASP_2021_A01
Cross-Domain JavaScript Source File Inclusion	Low	Medium	55	OWASP_2021_A08
Permissions Policy Header Not Set	Low	Medium	153	OWASP_2021_A01
X-Content-Type-Options Header Missing	Low	Medium	186	OWASP_2021_A05
Dangerous JS Functions	Low	Low	31	OWASP_2021_A04
Timestamp Disclosure - Unix	Low	Low	15	OWASP_2021_A01

Anexo J – Relatório Exemplo

ZAP Scanning Report

Generated with ZAP on Mon 12 Aug 2024, at 06:40:47

ZAP Version: 2.15.0

ZAP is supported by the Crash Override Open Source Fellowship

Contents

About this report

Report parameters

Contexts

The following contexts were selected to be included:

- FenixIscteAuth2

Sites

The following sites were included:

- https://maxcdn.bootstrapcdn.com
- https://cdnjs.cloudflare.com
- https://fenix-qua.iscte-iul.pt

(If no sites were selected, all sites were included by default.)

An included site must also be within one of the included contexts for its data to be included in the report.

Risk levels

Included: High, Medium, Low

Excluded: High, Medium, Low, Informational

Confidence levels

Included: User Confirmed, High, Medium, Low

Excluded: User Confirmed, High, Medium, Low, False Positive

Summaries

Alert counts by risk and confidence

This table shows the number of alerts for each level of risk and confidence included in the report.

(The percentages in brackets represent the count as a percentage of the total number of alerts included in the report, rounded to one decimal place.)

		Confidence				
		User Confirmed	High	Medium	Low	Total
Risk	High	0 (0.0%)	0 (0.0%)	0 (0.0%)	1 (4.2%)	1 (4.2%)
	Medium	0 (0.0%)	2 (8.3%)	9 (37.5%)	1 (4.2%)	12 (50.0%)
	Low	0 (0.0%)	2 (8.3%)	7 (29.2%)	2 (8.3%)	11 (45.8%)
	Total	0 (0.0%)	4 (16.7%)	16 (66.7%)	4 (16.7%)	24 (100%)

Alert counts by site and risk

This table shows, for each site for which one or more alerts were raised, the number of alerts raised at each risk level.

Alerts with a confidence level of "False Positive" have been excluded from these counts.

(The numbers in brackets are the number of alerts raised for the site at or above that risk level.)

		Risk		
		High (= High)	Medium (>= Medium)	Low (>= Low)
Site	https://fenix-qua.iscte-iul.pt	1 (1)	12 (13)	11 (24)

Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

Alert type	Risk	Count
Path Traversal	High	3 (12.5%)
Absence of Anti-CSRF Tokens	Medium	2 (8.3%)
Anti-CSRF Tokens Check	Medium	1 (4.2%)
Buffer Overflow	Medium	12 (50.0%)
Bypassing 403	Medium	10 (41.7%)
Content Security Policy (CSP) Header Not Set	Medium	99 (412.5%)
Format String Error	Medium	11 (45.8%)
Integer Overflow Error	Medium	11 (45.8%)
Missing Anti-clickjacking Header	Medium	62 (258.3%)
Proxy Disclosure	Medium	7 (29.2%)
Relative Path Confusion	Medium	16 (66.7%)
Sub Resource Integrity Attribute Missing	Medium	107 (445.8%)
Vulnerable JS Library	Medium	4 (16.7%)
Application Error Disclosure	Low	1 (4.2%)
Cookie No HttpOnly Flag	Low	1 (4.2%)

Cookie without SameSite Attribute	Low	50 (208.3%)
Cross-Domain JavaScript Source File Inclusion	Low	55 (229.2%)
Dangerous JS Functions	Low	31 (129.2%)
Permissions Policy Header Not Set	Low	153 (637.5%)
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	232 (966.7%)
Strict-Transport-Security Header Not Set	Low	228 (950.0%)
Timestamp Disclosure - Unix	Low	15 (62.5%)
X-Content-Type-Options Header Missing	Low	186 (775.0%)
Total		24

Alerts

Risk=High, Confidence=Low (1)

https://fenix-qua.iscte-iul.pt (1)

Path Traversal (1)

▶ POST https://fenix-qua.iscte-iul.pt/uiLayer?reload=true&v-1723423169317

Risk=Medium, Confidence=High (2)

https://fenix-qua.iscte-iul.pt (2)

Content Security Policy (CSP) Header Not Set (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Sub Resource Integrity Attribute Missing (1)

▶ GET https://fenix-qua.iscte-iul.pt/personal/school-services/com.qubit.solution.fenixedu.customers.iscte.module.iscteextensions.presentation.screens.help.QualityEnvironmentScreen

Risk=Medium, Confidence=Medium (9)

https://fenix-qua.iscte-iul.pt (9)

Anti-CSRF Tokens Check (1)

▶ GET https://fenix-qua.iscte-iul.pt/static/qubFeedbackCollector/qubSupport.html?date=2024-08-12T00:41:01.356Z

Buffer Overflow (1)

▶ POST https://fenix-qua.iscte-iul.pt/uiLayer/UIDL/?v-uiId=0

Bypassing 403 (1)

▶ GET https://fenix-qua.iscte-iul.pt/

Format String Error (1)

▶ POST https://fenix-qua.iscte-iul.pt/uiLayer?reload=true&v-1723423120523

Integer Overflow Error (1)

▶ POST https://fenix-qua.iscte-iul.pt/uiLayer/UIDL/?v-uiId=0

Missing Anti-clickjacking Header (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Proxy Disclosure (1)

▶ POST https://fenix-qua.iscte-iul.pt/api/bennu-core/fenixedu-profile/extension/loginExtension

Relative Path Confusion (1)

▶ GET https://fenix-qua.iscte-iul.pt/personal/communication/com.qubit.solution.fenixedu.module.cmsui.presentation.screens.manageSites.SearchSite

Vulnerable JS Library (1)

▶ GET https://fenix-qua.iscte-iul.pt/themes/fenixedu-omnis-theme/bower_components/jquery/dist/jquery.min.js

Risk=Medium, Confidence=Low (1)

https://fenix-qua.iscte-iul.pt (1)

Absence of Anti-CSRF Tokens (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Risk=Low, Confidence=High (2)

https://fenix-qua.iscte-iul.pt (2)

Server Leaks Version Information via "Server" HTTP Response Header Field (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Strict-Transport-Security Header Not Set (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Risk=Low, Confidence=Medium (7)

https://fenix-qua.iscte-iul.pt (7)

Application Error Disclosure (1)

▶ GET https://fenix-qua.iscte-iul.pt/api/bennu-core/fenixedu-profile/extension/loginExtension

Cookie No HttpOnly Flag (1)

▶ POST https://fenix-qua.iscte-iul.pt/api/bennu-core/fenixedu-profile/extension/loginExtension

Cookie Without Secure Flag (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Cookie without SameSite Attribute (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Cross-Domain JavaScript Source File Inclusion (1)

▶ GET https://fenix-qua.iscte-iul.pt/personal/school-services/com.qubit.solution.fenixedu.customers.iscte.module.iscteextensions.presentation.screens.help.QualityEnvironmentScreen

Permissions Policy Header Not Set (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

X-Content-Type-Options Header Missing (1)

▶ GET https://fenix-qua.iscte-iul.pt/login

Risk=Low, Confidence=Low (2)

https://fenix-qua.iscte-iul.pt (2)

Dangerous JS Functions (1)

▶ GET https://fenix-qua.iscte-iul.pt/themes/fenixedu-omnis-theme/bower_components/jquery/dist/jquery.min.js

Timestamp Disclosure - Unix (1)

▶ POST https://fenix-qua.iscte-iul.pt/uiLayer?reload=true&v-1723423120523

Appendix

Alert types

This section contains additional information on the types of alerts in the report.

Path Traversal

Source	raised by an active scanner (Path Traversal)
CWE ID	22
WASC ID	33

Reference	▪ https://owasp.org/www-community/attacks/Path_Traversal ▪ https://cwe.mitre.org/data/definitions/22.html

Absence of Anti-CSRF Tokens

Source	raised by a passive scanner (Absence of Anti-CSRF Tokens)
CWE ID	352
WASC ID	9
Reference	▪ https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html ▪ https://cwe.mitre.org/data/definitions/352.html

Anti-CSRF Tokens Check

Source	raised by an active scanner (Anti-CSRF Tokens Check)
CWE ID	352
WASC ID	9
Reference	▪ https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html ▪ https://cwe.mitre.org/data/definitions/352.html

Buffer Overflow

Source	raised by an active scanner (Buffer Overflow)
CWE ID	120
WASC ID	7
Reference	▪ https://owasp.org/www-community/attacks/Buffer_overflow_attack

Bypassing 403

Source	raised by an active scanner (Bypassing 403)
Reference	▪ https://www.acunetix.com/blog/articles/a-fresh-look-on-reverse-proxy-related-attacks/ ▪ https://i.blackhat.com/us-18/Wed-August-8/us-18-Orange-Tsai-Breaking-Parser-Logic-Take-Your-Path-Normalization-Off-And-Pop-0days-Out-2.pdf

	• https://www.contextis.com/en/blog/server-technologies-reverse-proxy-bypass

Content Security Policy (CSP) Header Not Set

Source	raised by a passive scanner (Content Security Policy (CSP) Header Not Set)
CWE ID	693
WASC ID	15
Reference	• https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy • https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html • https://www.w3.org/TR/CSP/ • https://w3c.github.io/webappsec-csp/ • https://web.dev/articles/csp • https://caniuse.com/#feat=contentsecuritypolicy • https://content-security-policy.com/

Format String Error

Source	raised by an active scanner (Format String Error)
CWE ID	134
WASC ID	6
Reference	• https://owasp.org/www-community/attacks/Format_string_attack

Integer Overflow Error

Source	raised by an active scanner (Integer Overflow Error)
CWE ID	190
WASC ID	3
Reference	• https://en.wikipedia.org/wiki/Integer_overflow • https://cwe.mitre.org/data/definitions/190.html

Missing Anti-clickjacking Header

Source	raised by a passive scanner (Anti-clickjacking Header)
CWE ID	1021
WASC ID	15
Reference	• https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

Proxy Disclosure

Source	raised by an active scanner (Proxy Disclosure)
CWE ID	200
WASC ID	45
Reference	• https://tools.ietf.org/html/rfc7231#section-5.1.2

Relative Path Confusion

Source	raised by an active scanner (Relative Path Confusion)
CWE ID	20
WASC ID	20
Reference	• https://arxiv.org/abs/1811.00917 • https://hsivonen.fi/doctype/ • https://www.w3schools.com/tags/tag_base.asp

Sub Resource Integrity Attribute Missing

Source	raised by a passive scanner (Sub Resource Integrity Attribute Missing)
CWE ID	345
WASC ID	15
Reference	• https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity

Vulnerable JS Library

Source	raised by a passive scanner (Vulnerable JS Library (Powered by

	Retire.js))
CWE ID	829
Reference	▪ https://github.com/jquery/jquery/issues/2432 ▪ http://blog.jquery.com/2016/01/08/jquery-2-2-and-1-12-released/ ▪ http://research.insecurelabs.org/jquery/test/ ▪ https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/ ▪ https://nvd.nist.gov/vuln/detail/CVE-2019-11358 ▪ https://github.com/advisories/GHSA-rmxg-73gg-4p98 ▪ https://nvd.nist.gov/vuln/detail/CVE-2015-9251 ▪ https://github.com/jquery/jquery/commit/753d591aea698e57d6db58c9f722cd0808619b1b ▪ https://github.com/jquery/jquery.com/issues/162 ▪ https://bugs.jquery.com/ticket/11974 ▪ https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/

Application Error Disclosure

Source	raised by a passive scanner (Application Error Disclosure)
CWE ID	200
WASC ID	13

Cookie No HttpOnly Flag

Source	raised by a passive scanner (Cookie No HttpOnly Flag)
CWE ID	1004
WASC ID	13
Reference	▪ https://owasp.org/www-community/HttpOnly

Cookie Without Secure Flag

Source	raised by a passive scanner (Cookie Without Secure Flag)
CWE ID	614

WASC ID	13
Reference	• https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes.html

Cookie without SameSite Attribute

Source	raised by a passive scanner (Cookie without SameSite Attribute)
CWE ID	1275
WASC ID	13
Reference	• https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site

Cross-Domain JavaScript Source File Inclusion

Source	raised by a passive scanner (Cross-Domain JavaScript Source File Inclusion)
CWE ID	829
WASC ID	15

Dangerous JS Functions

Source	raised by a passive scanner (Dangerous JS Functions)
CWE ID	749
Reference	• https://angular.io/guide/security

Permissions Policy Header Not Set

Source	raised by a passive scanner (Permissions Policy Header Not Set)
CWE ID	693
WASC ID	15
Reference	• https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Permissions-Policy • https://developer.chrome.com/blog/feature-policy/ • https://scotthelme.co.uk/a-new-security-header-feature-policy/ • https://w3c.github.io/webappsec-feature-policy/

	▪ https://www.smashingmagazine.com/2018/12/feature-policy/

Server Leaks Version Information via "Server" HTTP Response Header Field

Source	raised by a passive scanner (HTTP Server Response Header)
CWE ID	200
WASC ID	13
Reference	▪ https://httpd.apache.org/docs/current/mod/core.html#servertokens ▪ https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648552(v=pandp.10) ▪ https://www.troyhunt.com/shhh-dont-let-your-response-headers/

Strict-Transport-Security Header Not Set

Source	raised by a passive scanner (Strict-Transport-Security Header)
CWE ID	319
WASC ID	15
Reference	▪ https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html ▪ https://owasp.org/www-community/Security_Headers ▪ https://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security ▪ https://caniuse.com/stricttransportsecurity ▪ https://datatracker.ietf.org/doc/html/rfc6797

Timestamp Disclosure - Unix

Source	raised by a passive scanner (Timestamp Disclosure)
CWE ID	200
WASC ID	13
Reference	▪ https://cwe.mitre.org/data/definitions/200.html

X-Content-Type-Options Header Missing

Source	raised by a passive scanner (X-Content-Type-Options Header Missing)
CWE ID	693
WASC ID	15
Reference	▪ https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85) ▪ https://owasp.org/www-community/Security_Headers