Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

DESENVOLVIMENTO DE FERRAMENTAS E PRÁTICAS PARA UMA IMPLEMENTAÇÃO EFETIVA DE DEVSECOPS

Relatório de Estágio

NUNO ANDRÉ FAUSTINO BERNARDINO

Leiria, Junho de 2024

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

DESENVOLVIMENTO DE FERRAMENTAS E PRÁTICAS PARA UMA IMPLEMENTAÇÃO EFETIVA DE DEVSECOPS

NUNO ANDRÉ FAUSTINO BERNARDINO
Número: 2230461

Supervisores: Fábio Antunes e Eduardo Piza

Relatório de estágio realizado sob orientação do Professor Doutor Filipe Santos Neves (fneves@ipleiria.pt).

Leiria, Junho de 2024

AGRADECIMENTOS

Em primeiro lugar, gostaria de expressar a minha gratidão à minha família, em especial aos meus pais, pelo seu apoio incondicional, compreensão e incentivo ao longo de todo o meu percurso académico. Agradeço também aos meus amigos, cuja companhia e amizade foram essenciais para manter o equilíbrio e a estabilidade emocional durante estes anos.

Agradeço ao meu orientador, Professor Doutor Filipe Santos Neves, pela sua orientação, disponibilidade e pelas valiosas sugestões e conselhos que contribuíram para o sucesso deste trabalho.

Gostaria também de expressar o meu profundo agradecimento à empresa *VOID SOFTWARE, S.A.* e a todos os seus colaboradores que, de forma direta ou indireta, contribuíram para o sucesso deste estágio. Em particular, deixo um agradecimento especial aos meus supervisores, Eduardo Piza e Fábio Henriques. Não posso deixar de mencionar a colaboradora Catarina Reis, responsável pelo estágio, cujo apoio contínuo, disponibilidade e valiosos *feedbacks* foram fundamentais tanto para o desenvolvimento das atividades como para a elaboração deste documento.

Por fim, agradeço a todos aqueles que, de alguma forma, contribuíram para que este estágio fosse possível, mesmo que não mencionados diretamente, o vosso apoio e incentivo foram fundamentais.

Obrigado a todos.

RESUMO

Com o rápido avanço da tecnologia e a adoção generalizada de metodologias ágeis, a integração da segurança — conhecida como DevSecOps — tornou-se essencial para assegurar a integridade e a segurança do software. Esta mudança de paradigma centra-se na incorporação de testes de segurança ao longo de todo o ciclo de vida do desenvolvimento de software, promovendo a realização de testes de segurança contínuos (*Continuous Security Testing*), em vez de os adiar para fases posteriores.

Neste documento, são descritas três ferramentas personalizadas, adaptadas a fases específicas do ciclo de desenvolvimento. Através da utilização destas ferramentas, as organizações podem reforçar as suas práticas de segurança e garantir a integridade do software ao longo do processo de desenvolvimento. É apresentada uma análise preliminar destas ferramentas, com o objetivo de melhorar a segurança da informação nas organizações. Este trabalho oferece um contributo valioso ao documentar e avaliar o impacto de ferramentas tecnológicas no ambiente organizacional, destacando a importância de soluções personalizadas para uma maior eficiência interna.

Além das ferramentas, foi também realizada uma outra atividade durante o estágio curricular: a execução de testes de complexidade de palavras-passe em duas das redes internas da organização, o que contribui igualmente para o fortalecimento da segurança da organização.

Em conclusão, o impacto positivo das ferramentas exploradas nesta tese reforça a necessidade de um investimento contínuo em tecnologias de suporte organizacional, oferecendo *insights* úteis para futuras implementações e melhorias.

Palavras-chave: *DevSecOps*, segurança, testes de segurança contínuos, automação, vulnerabilidades.

ABSTRACT

With the rapid advancement of technology and the widespread adoption of agile methodologies, the integration of security - known as DevSecOps - has become essential to maintain the integrity and security of software. This paradigm shift focuses on incorporating security testing throughout the software development lifecycle, advocating Continuous Securiy Testing rather than postponing it until later stages.

This document describes three customized tools adapted to specific phases of the development cycle. By using these tools, organizations can strengthen their security practices and maintain software integrity throughout the development process. A preliminary analysis of these tools with the aim of improving information security in organizations is presented. In addition to the tools, another activity was also carried out during the internship: running password complexity tests on two of the organization's internal networks.

In addition to the tools, another activity was carried out during the curricular internship: the execution of password complexity tests on two of the organization's internal networks, which also contributes to strengthening the organization's security.

In conclusion, the positive impact of the tools explored in this thesis reinforces the need for continuous investment in organizational support technologies, providing useful insights for future implementations and improvements.

Keywords: DevSecOps, security, continuous security testing, automation, vulnerabilities.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGENS

LISTA DE ABREVIATURAS

2FA	Two-factor Authentication.
AA	Authenticator Address.
ACL	Access Control List.
AP	Access Point.
API	Application Programming Interface.
ASVS	Application Security Verification Standard.
BD	Bases de Dados.
BIC	Bulk Issue Creator.
CI/CD	Continuous Integration and Continuous Delivery.
CIS	Center of Internet Security.
CLI	Command-Line Interface.
CPU	Central Processing Unit.
CSV	Comma-Separated Values.
DAST	Dynamic Application Security Testing.
DER	Diagrama de Entidade e Relacionamento.
DevOps	Development, Operations.
DevSecOps	Development, Security, and Operations.
EAPOL	Extensible authentication protocol over LAN.
GMK	Group Master Key.
GPU	Graphics Processing Unit.
GTK	Group Temporal Key.

HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
I/O	Input/Output.
IA	Inteligência Artificial.
IAST	Interactive Application Security Testing.
IDS	Intrusion Detection System.
IPS	Intrusion prevention system.
JSON	JavaScript Object Notation.
JWT	JSON Web Token.
MDM	Mobile Device Management.
MIC	Message Integrity Code.
MobSF	Mobile Security Framework.
MSK	Master Session Key.
NIC	Network Interface Card.
OWASP	Open Web Application Security Project.
PMK	Pairwise Master Key.
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analysis.
PTK	Pairwise Transient Key.
RAM	Random Access Memory.
SA	Supplicant Address.
SAST	Static Application Security Testing.
SCA	Software Composition Analysis.
SDLC	Software Development Lyfe Cicle.
SIEM	Security Information and Event Management.

SLA	Service Level Agreement.
SO	Sistema Operativo.
SP	Special Publication.
TI	Tecnologia da Informação.
URL	Uniform Resource Locator.
WSAP	Web Security Application Project.
XDR	eXtended Detection and Response.
ZAP	Zed Attack Proxy.

1

INTRODUÇÃO

Nos últimos anos, a transformação digital tornou-se um tema central em diversos setores da economia. À medida que o mundo transita para uma era cada vez mais digital, as empresas são impelidas a adotar novas tecnologias para manter a sua competitividade e garantir a sustentabilidade a longo prazo. No entanto, esse movimento rumo à digitalização traz consigo novos desafios, sendo a cibersegurança um dos mais críticos.

Com a crescente dependência de sistemas digitais, as organizações estão cada vez mais expostas a uma ampla gama de ameaças cibernéticas, como ataques direcionados, *data leaks*[1] e violações de privacidade. Estes incidentes podem causar danos significativos, incluindo perdas financeiras, danos reputacionais, interrupção das operações e, em muitos casos, a violação de regulamentações e leis de proteção de dados.

Consequentemente, a proteção dos ativos digitais, a integridade das informações e a privacidade dos clientes tornaram-se fatores críticos para o sucesso das organizações no ambiente digital. Dada a sofisticação crescente dos ciberataques e a rápida evolução do panorama de ameaças, torna-se imperativo que as empresas adotem uma abordagem proativa em matéria de cibersegurança. Isso inclui não apenas a implementação de soluções tecnológicas robustas, mas também o desenvolvimento de uma cultura organizacional orientada para a segurança. Ao investir em medidas abrangentes de proteção cibernética, as empresas podem reduzir significativamente o risco de ataques e, em caso de incidentes, mitigar os seus impactos.

Neste sentido, o presente documento descreve todo o trabalho desenvolvido durante o estágio curricular realizado na *VOID SOFTWARE, S.A* [2], com vista a uma redução do risco de ataques e mitigação dos impactos daí decorrentes.

1.1 MOTIVAÇÃO E OBJETIVOS

Acarretando a transformação digital dos novos desafios de segurança da informação, a empresa propôs que o principal foco deste estágio fosse na avaliação de segurança, tanto de aplicações como da infraestrutura da mesma. Esta avaliação iria ser feita

[1] Incidente em que informações sensíveis são acedidas por pessoas não autorizadas devido a falhas internas ou erros de configuração[1]

através da realização de *pentests*. Para consultar um maior detalhe nas tarefas que foram propostas inicialmente, consultar o Apêndice A.

Para a prossecução deste trabalho, foi necessário tomar contacto com as ferramentas de segurança utilizadas na *VOID SOFTWARE, S.A.* e por fim perceber como funciona o processo de *reporting* das vulnerabilidades de segurança encontradas para com a equipa de desenvolvimento e integrar esta etapa no Software Development Lyfe Cicle (SDLC).

No decorrer do estágio foram alinhadas as necessidades mais prementes da empresa, não desvirtuando os objetivos propostos para o estágio, o que resultou numa redefinição de tarefas de acordo com o apresentado na secção 1.5.

1.2 ENTIDADE DE ACOLHIMENTO

Fundada em 2006, a *VOID SOFTWARE, S.A.* é uma empresa especializada no desenvolvimento de software à medida, com recurso a uma vasta gama de tecnologias de programação.

Figura 1: Logo *VOID SOFTWARE, S.A.* [3]

No que toca ao desenvolvimento dos seus produtos, esta organização segue uma abordagem ágil, o que ajuda a ir ao encontro das necessidades individuais de cada cliente, privilegiando a experiência emocional do utilizador.

Com clientes espalhados por toda a parte do mundo, a *VOID SOFTWARE, S.A.* tem atualmente um foco especial em áreas como as de Inteligência Artificial (IA), *Machine Learning* e *Blockchain*. Desde 2016 esta organização começou a apostar no desenvolvimento de produtos inovadores (próprios ou em parceria) e não tem parado desde então, participando em diferentes *startups* e outros projetos na área digital.

A *VOID SOFTWARE, S.A.* atualmente conta com mais de 80 colaboradores que se encontram divididos por áreas. Estas áreas podem ser visualizadas na seguinte Figura 2, que é o organograma que representa visualmente a estrutura organizacional desta empresa.

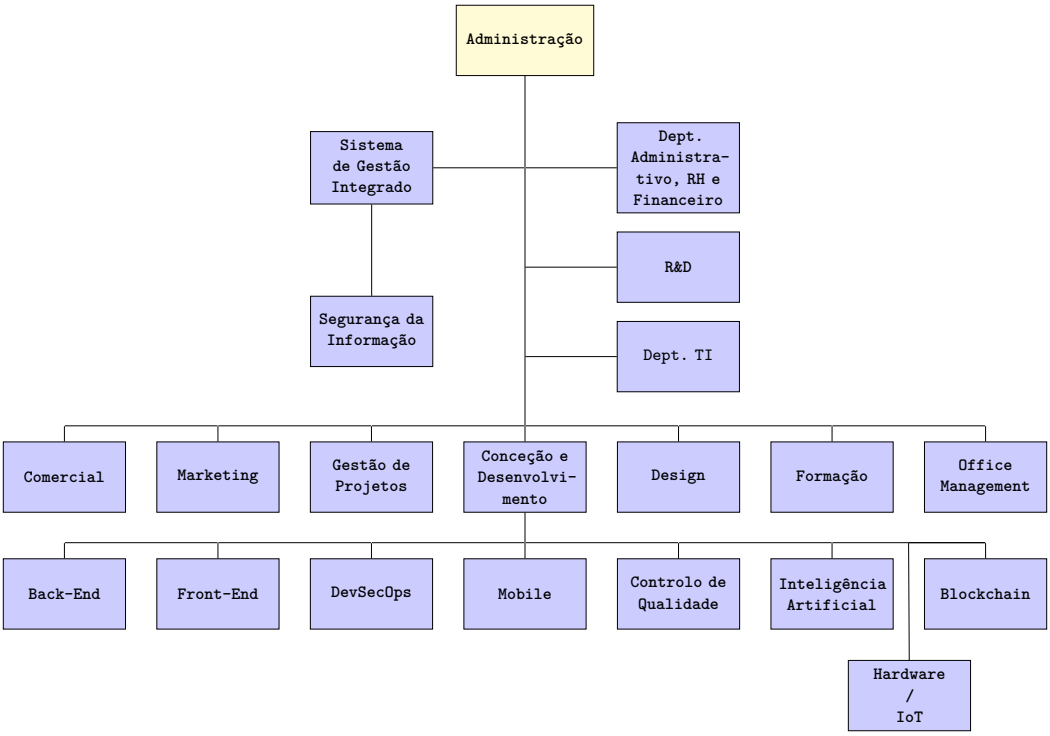

Figura 2: Diagrama organizacional da *VOID SOFTWARE, S.A.*

As funções desempenhadas durante a realização deste estágio curricular enquadram-se na área Development, Security, and Operations (DevSecOps). Os profissionais desta área desempenham diversas funções, das quais se destacam:

- Manter e monitorizar as infraestruturas dos projetos.
- Garantir as condições técnicas necessárias para executar e manter o software desenvolvido.
- Configurar e manter as ferramentas de apoio ao processo de desenvolvimento.

Por fim, os valores da empresa *VOID SOFTWARE, S.A.* são:

- Colocar sempre as pessoas antes do negócio.
- Compromisso dentro da equipa e perante os clientes
- Construção contínua do conhecimento através da experiência
- Orgulho e excelência em tudo o que fazem.

1.3 PLANEAMENTO

Nesta secção podemos analisar como se encontra distribuída a carga laboral pelos 9 meses de duração do estágio curricular. Na seguinte Figura 3 é possível ver o diagrama de *Gantt* dessa distribuição por semana.

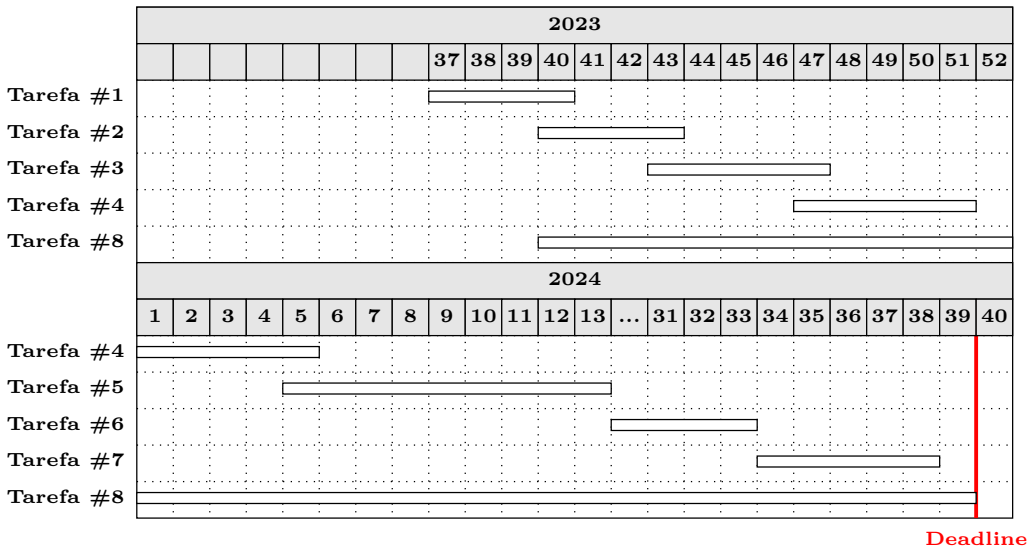

Figura 3: Diagrama de *Gantt*

As tarefas representadas por "Tarefa **#n**" encontram-se definidas na tabela 1.

Número da Tarefa	**Designação**
Tarefa #1	Visão geral das tecnologias e ferramentas utilizadas
Tarefa #2	Visão geral do processo de desenvolvimento e padrões de segurança em prática dentro da empresa
Tarefa #3	Conceção do processo de *reporting* de problemas de vulnerabilidade e integração no *SDLC*
Tarefa #4	Desenvolvimento da ferramenta *Bulk Issue Creator (BIC)*
Tarefa #5	Desenvolvimento da ferramenta *Version Checker*
Tarefa #6	Desenvolvimento da ferramenta *Cloud Cleaner*
Tarefa #7	Análise de Segurança em Redes *Wi-Fi*
Tarefa #8	Escrita do Relatório

Tabela 1: Lista de Tarefas

O estágio foi realizado em regime presencial, todos os dias úteis das 9h-18h, com exclusão das sextas-feiras, o dia reservado exclusivamente para a elaboração do presente relatório. Todas as sextas-feiras existiram reuniões (online/presenciais) para ser acompanhado o progresso do relatório.

1.4 ENQUADRAMENTO DAS ATIVIDADES DO ESTÁGIO

Após a alteração das atividades do estágio curricular, as novas atividades podem ser enquadradas em várias etapas do ciclo de DevSecOps. A seguinte Figura 4 ilustra esse mesmo ciclo.

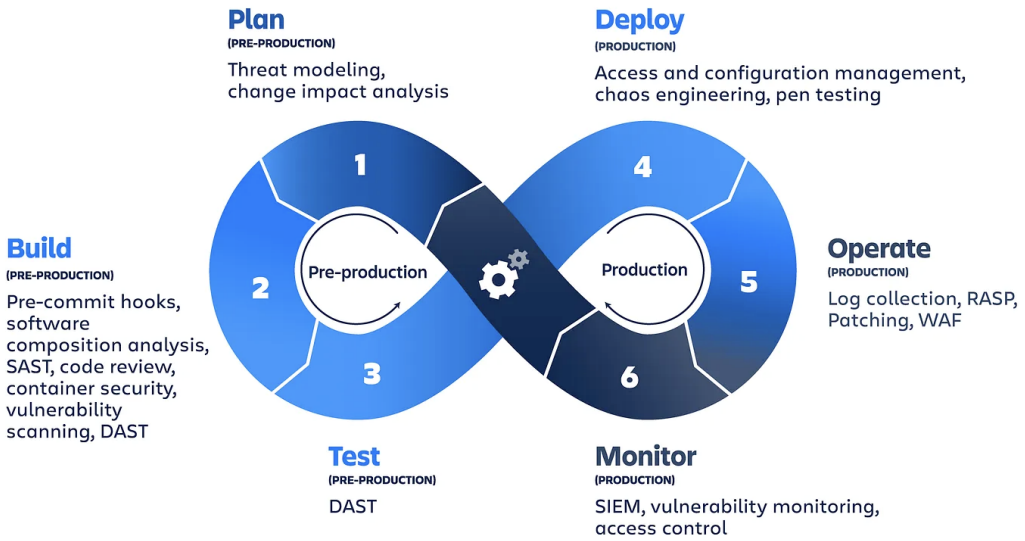

Figura 4: Ciclo DevSecOps[4]

DevSecOps é um termo que surge da combinação de três conceitos principais:

1. **Desenvolvimento (Dev)**: Refere-se à equipa de desenvolvimento de software que envolve todos os profissionais envolvidos no desenvolvimento de aplicações, desde programadores, engenheiros de software entre outros.
2. **Segurança (Sec)**: Representa a integração de práticas de segurança de todas as etapas do SDLC. Ao invés da segurança ser tratada como uma etapa separada (após o desenvolvimento das aplicações), os DevSecOps procuram incorporá-la (a segurança) desde o ínicio do desenvolvimento de uma aplicação.
3. **Operações (Ops)**: Refere-se às operações de Tecnologia da Informação (TI), que envolvem a implementação, monitorização e manutenção tanto das aplicações como da infraestrutura de uma organização.

Assim, DevSecOps é a designação dada a uma abordagem ao desenvolvimento de software que automatiza o processo de desenvolvimento, teste, implementação e manutenção de aplicações, garantindo simultaneamente o cumprimento dos requisitos de segurança e de conformidade para com as normas de segurança da organização [5].

O ciclo de DevSecOps é frequentemente representado através de um símbolo do infinito, que ilustra a natureza contínua e integrada deste processo. Tal como o símbolo sugere, as fases de desenvolvimento, segurança e operações estão interligadas e ocorrem de forma ininterrupta, sem um ponto final definido. Isto reflete o facto de que, no ciclo de DevSecOps, o departamento assume a responsabilidade por todas as fases representadas no SDLC, num fluxo contínuo que abrange desde a criação e implementação até à monitorização e otimização das aplicações. Este processo cíclico assegura que as aplicações são continuamente melhoradas e protegidas, sem

comprometer a agilidade da entrega de software, mantendo a segurança como um pilar essencial.

Nesta representação, é possível observar como as funções desempenhadas pelo departamento de DevSecOps, dentro de uma organização, se interligam de forma dinâmica e contínua, sublinhando a importância de integrar a segurança em todas as fases do ciclo de desenvolvimento.

No decorrer do estágio curricular, foram desenvolvidas várias ferramentas que se enquadram em diferentes etapas deste ciclo. Na secção 1.5, será apresentado o enquadramento dessas ferramentas.

1.4.1 *Publicação*

Durante o decorrer do estágio, tive a oportunidade de apresentar parte do trabalho desenvolvido numa conferência científica, o que permitiu divulgar os resultados das atividades realizadas com a comunidade académica e profissional. O artigo, intitulado "*Enhancing DevSecOps: Three Custom Tools for Continuous Security*", foi publicado na 11ª edição da conferência "*2024 IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)* [6]", que decorreu entre os dias 28 e 30 de Junho, em Shanghai, China.

Este trabalho concentrou-se na descrição e análise de três ferramentas principais utilizadas e desenvolvidas durante o estágio, as quais são igualmente detalhadas neste documento. A publicação explorou a implementação e o impacto dessas ferramentas no contexto organizacional, destacando as suas funcionalidades, benefícios e a aplicação prática. É importante destacar que o desenvolvimento deste artigo contou com o valioso apoio do orientador: Professor Doutor Filipe Santos Neves; os supervisores de estágio: Fábio Henriques, Eduardo Piza e dois colaboradores da organização: Bernardo Sequeira e Catarina Reis, cuja colaboração foi essencial para a sua elaboração e finalização.

O artigo pode ser consultado online em [7] e no Apêndice B.

1.5 ESTRUTURA DO RELATÓRIO

O restante deste documento encontra-se estruturado da seguinte forma:

No segundo capítulo encontra-se descrito todo o trabalho relacionado com o tema do relatório do Estágio. Este capítulo encontra-se subdividido em duas secções. A secção 2.1 destaca um artigo que destaca desafios e objetivos para uma correta aplicação da cultura DevSecOps. A secção 2.2 apresenta a revisão de literatura que

foi efetuada na fase inicial do projeto, que compara os resultados do que foi obtido com o estágio com os resultados encontrados.

No capítulo 3 é apresentada a ferramenta BIC. Esta primeira ferramenta desenvolvida no estágio curricular, foi desenvolvida com o intuito de facilitar a ação de reportar vulnerabilidades (*issues*) de uma aplicação antes desta ir para produção.

Pelas funções que esta ferramenta desempenha, enquadra-se nas etapas 2 e 3 ("*Build*" e "*Test*") do ciclo de DevSecOps.

No capítulo 4 é apresentada a ferramenta *Version Checker*. Esta segunda ferramenta desenvolvida tem a funcionalidade de verificar se os softwares usados internamente numa organização se encontram na última versão estável dos mesmos. O processo de desenvolvimento e o funcionamento desta ferramenta encontra-se detalhado no capítulo 4.

Tendo em conta que o intuito desta ferramenta é monitorizar a infraestrutura da organização, prevenindo a existência de software desatualizado e assim colmatar possíveis fontes de ataques, esta ferramenta enquadra-se na etapa 6 ("*Monitor*") da Figura 4.

No capítulo 5 é apresentado a ultima ferramenta, *Cloud Cleaner*. Esta ferramenta tem a funcionalidade de fornecer uma listagem sobre todas as pastas/ficheiros partilhadas(os) existentes nas diferentes *Clouds* da organização.

Esta ferramenta fornece uma lista com informação relativa a todos os *shares*[1] (internos e para o exterior) com alguma informação relativa ao conteúdo partilhado. O seu desenvolvimento e a sua explicação encontra-se detalhada mais à frente neste documento no Capítulo 5.

Por recorrer à análise de ficheiros (*Logs*) para retirar informação relativa aos *shares* com o intuito de prevenir a partilha de informação desnecessária, esta ferramenta pode ser enquadrada nas etapas 5 e 6 ("*Operate*" e "*Monitor*").

O Capítulo 6 apresenta as conclusões decorrentes do desenvolvimento das atividades do estágio, destacando como os objetivos previamente estabelecidos foram alcançados. Este capítulo inclui ainda diversas propostas para trabalhos futuros, visando a continuidade e aperfeiçoamento das soluções apresentadas.

No Apêndice C é possível visualizar a execução dos testes de complexidade das palavras-passe em duas das redes internas da organização. Esta tarefa foi-me delegada e sugerida durante a realização de um *pentest* interno que estava a ser conduzido na organização na altura. Este capítulo foi movido para apêndice, uma vez que o tema se desvia ligeiramente do título e da temática principal deste relatório.

[1] Um *share* é uma partilha. Por exemplo, no caso de um utilizador possuir uma pasta partilhada com outros dois utilizadores distintos, existem dois *shares* para essa pasta

No entanto, sendo uma das atividades realizadas, faz todo o sentido que esteja presente no documento.

2

TRABALHO RELACIONADO

Este capítulo tem como objetivo fornecer uma visão abrangente do estado da arte, identificando o trabalho previamente realizado, os principais avanços tecnológicos, e o conhecimento atual relacionado a aplicações que desempenham atividades semelhantes às ferramentas desenvolvidas ao longo do estágio curricular.

A aplicação da segurança ao longo do SDLC é crucial para manter a integridade e a segurança das aplicações de software. Todos os dias surgem novas ameaças, que evoluem juntamente com a tecnologia. A Microsoft[1], já em 2006, abordou a necessidade de garantir a segurança, uma vez que "tudo está interligado", e introduziu o seu Ciclo de Vida de Desenvolvimento de Segurança [8].

Desde então, foram introduzidas e implementadas várias ferramentas e práticas ao longo do SDLC para garantir a segurança em diferentes fases do desenvolvimento. Os testes estáticos de segurança das aplicações (Static Application Security Testing (SAST)) são utilizados para identificar vulnerabilidades de segurança no código-fonte. Os testes dinâmicos de segurança das aplicações (Dynamic Application Security Testing (DAST)) testam as aplicações em execução em tempo real. Os testes interativos de segurança das aplicações (Interactive Application Security Testing (IAST)) combinam os SAST com os DAST. A análise da composição do software (Software Composition Analysis (SCA)) identifica e rastreia as vulnerabilidades relacionadas com as dependências. Estas são as categorias mais conhecidas e amplamente utilizadas, entre outras categorias e tipologias [9].

No entanto, e apesar da popularidade e dos benefícios percebidos, os aspetos de segurança de software no contexto do Development, Operations (DevOps) continuam a ser uma preocupação para a maioria das organizações. Para lidar com algumas questões de segurança, a cultura organizacional é um elemento essencial que necessita de ser adequadamente abordado no âmbito do DevSecOps.

A cultura DevSecOps é um tópico emergente que vai além da simples transferência de algumas práticas de segurança para uma fase anterior do SDLC.

[1] https://www.microsoft.com/pt-pt/

2.1 DESAFIOS E OBJETIVOS IDENTIFICADOS PARA A CORRETA IMPLEMENTAÇÃO DA PRÁTICA DEVSECOPS

Revisões recentes da literatura mostram que há várias necessidades por resolver e que ainda há um longo caminho a percorrer para alcançar o nível desejado de testes de segurança contínuos [10, 11, 12].

Especificamente, [12] identificou vários desafios que precisam de ser abordados, bem como direções ou metas futuras a atingir.

A lista seguinte apresenta os desafios mais relevantes, que foram considerados prioritários para este trabalho:

- D1: Questões de segurança resultantes da complexidade das ferramentas e dos desafios de integração
- D2: Incapacidade de automatizar completamente as práticas de segurança tradicionalmente manuais, de modo a integrá-las nas práticas de DevSecOps
- D3: Dificuldade em realizar uma avaliação rápida dos requisitos de segurança

Destaco também os seguintes objetivos:

- O1: A necessidade de ferramentas de segurança dirigidas aos programadores, e não exclusivamente aos especialistas em segurança (*p.e.*, segurança centrada no programador)
- O2: Testes de segurança de aplicações como um serviço
- O3: Descoberta contínua de vulnerabilidades e práticas de gestão

2.2 METODOLOGIA DA REVISÃO SISTEMÁTICA DA LITERATURA

Nesta secção foi feita uma pesquisa sobre o trabalho relacionado para cada ferramenta.

2.2.1 *Bulk Issue Creator*

Esta secção apresenta uma comparação entre a ferramenta BIC e outras disponíveis no mercado, analisando de forma sucinta as alternativas, com destaque para as suas principais características e a possibilidade de implementação da funcionalidade de criação automática de *issues*. No final, será apresentada uma tabela de comparações entre a ferramenta BIC e as mencionadas.

1. ***Jira Automation***: é uma ferramenta nativa da plataforma *Jira* que permite automatizar processos e fluxos de trabalho de forma simples e eficaz através de regras que são ativadas por *triggers* definidos pelos utilizadores [13]. Esta ferramenta permite automatizar uma variedade de tarefas dentro do *Jira*, como por exemplo:

   - **Transição automática de estados**: Quando uma *issue* é marcada como concluída, o *Jira Automation* pode automaticamente mover essa tarefa para o estado "Done".
   - **Atribuição automática de tarefas**: Atribuir uma *issue* automaticamente a um membro da equipa com base num critério (*p.e.*, o tipo do *issue* - *Bug*, *Task*, *Improvement*, entre outros.
   - **Fechar *issues* sem atividade**: Fechar automaticamente *issues* que estiverem sem atividade por um determinado período de tempo definido.
   - **Integração com outras Ferramentas**: Criar automações baseadas em integrações com outras plataformas, como *GitHub* [14], *Bitbucket* [15], *Slack* [16], entre outras. Um exemplo da integração com o *GitHub* é: se um *pull request* for aprovado no *GitHub*, uma *issue* no *Jira* pode ser automaticamente movida para a coluna "*In Progress*".

   Pela pesquisa realizada, esta ferramenta interna do *Jira* não oferece suporte direto para ler ficheiros *Excel* locais e criar *issues* automaticamente na *Board* do respetivo projeto.

2. ***Open Web Application Security Project (OWASP) DefectDojo***: É uma plataforma *open-source* dedicada à gestão de segurança de uma organização [17]. Esta ferramenta foi desenvolvida com o propósito de:

   - Automatizar testes de segurança;
   - Analisar vulnerabilidades;
   - Realizar deduplicação, ou seja, eliminar cópias redundantes de dados e reduzir a sobrecarga de armazenamento;
   - Integrar e automatizar diversas ferramentas.

   Com esta ferramenta, é possível importar os *issues* para o *OWASP DefectDojo* e, após estabelecer a conexão com o *Jira*, replicar automaticamente esses *issues* na plataforma do *Jira*. Ficando assim com os *issues* em duas plataformas.

3. ***Synk***: É uma plataforma de segurança focada em programadores, projetada para ajudar a criar código seguro desde o início do seu desenvolvimento. Integra-se em todo o SDLC para identificar e ajudar a corrigir vulnerabilidades presentes no código [18].

As principais tarefas que se podem esperar desta ferramenta são:

- Automatizar a deteção de vulnerabilidades em várias fases do desenvolvimento.
- Fornecer *insights* acionáveis para que os programadores resolvam problemas de segurança de forma eficiente.
- Integrar-se em *pipelines*, melhorando a segurança sem interromper o fluxo de trabalho.

Com base nas informações recolhidas, foi possível elaborar a Tabela 2, que compara funcionalidades entre a ferramenta BIC e as ferramentas mencionadas anteriormente.

Funcionalidade	**BIC**	***Jira Automation***	***OWASP DefectDojo***	***Snyk***
Foco principal	Criar *issues* automaticamente para o *Jira*	Automação de fluxos gerais.	Gestão de vulnerabilidades.	Segurança de código.
Automatização da criação de *issues*	Sim	Sim	Sim, focado em vulnerabilidades.	Sim, focado em vulnerabilidades de código.
Integração com *Excel*	Sim, direta	Não diretamente.	Não diretamente.	Não.
Suporte para *Jira*	Sim	Sim	Sim	Sim
Custo (Licenciamento)	Gratuito	Depende do plano *Jira*.	Com planos pagos.	Com planos pagos.
Facilidade de Configuração	Fácil	Fácil	Moderada	Fácil

Tabela 2: Tabela de comparações (BIC)

A análise da Tabela 2 revela que, embora todas as ferramentas permitam a criação de *issues*, cada uma se foca em áreas específicas, como a gestão de vulnerabilidades ou a segurança de código. A ferramenta BIC destaca-se pela capacidade de criar *issues* automaticamente no *Jira* de forma direta, com um foco específico na análise de código, utilizando um ficheiro *Excel* (OWASP Application Security Verification Standard (ASVS)). Além disso, por ser uma solução gratuita, o BIC torna-se mais acessível em comparação com as outras ferramentas avaliadas.

2.2.2 *Version Checker*

Nesta secção, será realizada uma comparação entre a ferramenta *Version Checker* e outras disponíveis no mercado. Tal como na secção anterior, será apresentada uma breve descrição de cada ferramenta selecionada, destacando as suas principais características. A análise focar-se-á na funcionalidade de comparação de versões de cada uma. No final, será apresentada uma tabela comparativa, evidenciando as vantagens da *Version Checker* em relação às outras opções.

1. ***Nagios***: Esta ferramenta fornece uma visão instantânea da infraestrutura de TI de uma organização, permitindo a monitorização de serviços e sistemas. O

Nagios ajuda a detetar, resolver e mitigar problemas antes que estes afetem os utilizadores finais e os clientes [19].

Esta ferramenta oferece um conjunto robusto de funcionalidades, como:

- Planeamento de atualizações para evitar falhas em sistemas desatualizados.
- Emissão de alertas em tempo real.
- Garantia de conformidade com os Service Level Agreement (SLA)s da organização estão a ser cumpridos.
- Monitorizar toda a sua infraestrutura e processos empresariais.
- Corrigir automaticamente os problemas quando estes são detetados.

A capacidade deste software pode ser estendida através da instalação/criação de *plugins* (*scripts*). Ele pode monitorizar desde métricas básicas, como a utilização de Central Processing Unit (CPU) e memória, até aspetos mais específicos de serviços críticos, como *webservers* ou Bases de Dados (BD). A partir da utilização destes *scripts* (em Bash, Python, etc.) é possível verificar as versões dos serviços.

2. ***Ansible***: é uma ferramenta de automação de TI *open-source* que facilita a gestão, configuração e orquestração de sistemas em infraestruturas de TI. A sua principal característica é a simplicidade de utilização, uma vez que não requer a instalação de agentes nos sistemas geridos. Em vez disso, o *Ansible* utiliza SSH para se conectar a servidores e executar tarefas definidas através de *playbooks*.

   O *Ansible* foi desenvolvido com base em princípios fundamentais que o tornam uma escolha popular entre os profissionais de TI [20]:

   - **Arquitetura sem agentes**: Esta abordagem resulta em baixos custos de manutenção, pois evita a necessidade de instalar software adicional na infraestrutura de TI.
   - **Simplicidade**: Os *playbooks* utilizam uma sintaxe *YAML* simples, que é fácil de ler e compreender.
   - **Escalabilidade e flexibilidade**: A escalabilidade é simples e rápida, permitindo automatizar sistemas de forma eficiente através de um *design* modular que suporta uma ampla variedade de sistemas operativos, plataformas de *cloud* e dispositivos de rede.

   Como mencionado anteriormente, é possível verificar as versões instaladas e o seu estado (atualizadas ou não) através da configuração de *playbooks*.

3. ***Puppet***: é mais uma ferramenta de automação de TI *open-source* à semelhança da ferramenta anterior (*Ansible*). Com este software é possível realizar ações como:

   - Automatizar a instalação de software.
   - Configuração de sistemas.
   - Implementação de políticas de segurança, garantindo que os servidores estejam sempre em conformidade com as especificações definidas.

   Para monitorizar versões de software, o *Puppet* utiliza *manifests*, que são *scripts* que definem as configurações desejadas para os sistemas. É possível, por exemplo especificar a versão exata de um software que deve ser instalado. Se o *Puppet* verificar que a versão instalada não corresponde à versão desejada, ele irá automaticamente aplicar as alterações necessárias para atualizar a versão.

Com esta informação, foi possível construir a Tabela 3 que nos faz uma comparação de métricas entre as ferramentas abordadas anteriormente.

Métrica	***Version Checker***	***Nagios***	***Ansible***	***Puppet***
Objetivo Principal	Verificação de versões	Monitorização de sistemas, redes e serviços	Automação de TI, gestão de configuração	Automação de TI, gestão de configuração
Instalação e Configuração	Simples	Moderada	Moderada	Complexa
Necessidade de Agentes	Não	Sim	Não	Sim
Capacidade de Customização	Baixa	Alta	Alta	Alta
Alertas e Relatórios	Relatório por *e-mail*	Alertas por *e-mail*, e outros canais	Suporte para *logs* e integração com sistemas de alertas	Alertas limitados, mais focado em *logs* e relatórios
Custo (Licenciamento)	Nenhum	Com planos pagos	Com planos pagos	Com planos pagos
Flexibilidade para Outras Funções	Não	Sim	Sim	Sim
Foco na Verificação de Versões	Sim (dedicado)	Não diretamente, mas possível com *plugins*	Não diretamente, mas possível com *playbooks* customizados	Não diretamente, mas possível com *manifests* customizados

Tabela 3: Tabela de comparações (*Version Checker*)

Embora existam ferramentas robustas e complexas, como as mencionadas anteriormente (ferramentas de gestão de TI) ou mesmo soluções de Mobile Device Management (MDM)[1], a *Version Checker* oferece uma abordagem mais simples e eficaz para empresas que necessitam de monitorizar exclusivamente as versões dos serviços instalados.

Desenvolvida com foco na comparação de versões de software, a *Version Checker* é ideal para organizações que desejam garantir que os seus serviços estão atualizados, evitando discrepâncias ou desatualizações que possam comprometer a segurança da organização. Esta ferramenta destaca-se por ser uma solução mais acessível, tanto

[1]software que faz a gestão de dispositivos como telemóveis, tablets ou portáteis. Ajuda a controlar o processo de gestão de todo o ciclo de vida dos dispositivos móveis utilizados no local de trabalho [21]

em termos de complexidade como de custos, quando comparada a plataformas mais abrangentes.

A *Version Checker* sobressai pela sua simplicidade e objetividade, permitindo às empresas concentrarem-se no essencial: perceber o estado atual das versões dos seus serviços. Além disso, as ferramentas mencionadas anteriormente têm a capacidade de emitir alertas quando as versões dos serviços instalados se encontram desatualizadas enquanto que o *Version Checker* envia um relatório para um endereço de *e-mail*.

Contudo, para organizações que ainda não dispõem de um sistema MDM ou que necessitam de implementar uma ferramenta de gestão de TI, pode ser mais vantajoso optar pelas soluções anteriormente mencionadas, ou outras, dado que essas plataformas oferecem uma ampla gama de funcionalidades para além da simples verificação de versões de software.

2.2.3 *Cloud Cleaner*

Nesta subsecção é utilizada a metodologia Preferred Reporting Items for Systematic Reviews and Meta-Analysis (PRISMA) utilizada para conduzir a revisão sistemática da literatura para a ferramenta *Cloud Cleaner*, permitindo uma abordagem estruturada e rigorosa na seleção e análise das fontes relevantes. Em seguida, são apresentados os estudos e trabalhos mais relevantes que abordam o tema em questão, destacando as soluções existentes, as suas características, e as inovações introduzidas. Por fim, é realizada uma análise crítica das soluções e arquiteturas identificadas, discutindo as suas limitações, desafios e oportunidades para futuras investigações e desenvolvimentos, de forma a contextualizar o trabalho desenvolvido durante o estágio no panorama atual.

A Figura 5 apresenta um diagrama de fluxo que ilustra o processo de seleção dos estudos relacionados com o tema em análise. O diagrama está organizado em três fases. A primeira, denominada identificação, consiste na obtenção dos artigos relevantes. A segunda fase é o *screening*, onde ocorre uma filtragem dos artigos para selecionar os mais pertinentes. A terceira e última fase inclui o cálculo do número total de artigos considerados relevantes para inclusão nesta secção do trabalho. Adicionalmente, o diagrama está dividido em duas colunas: a primeira refere-se aos artigos e registos encontrados em bases de dados científicas; a segunda, aos registos obtidos através de outros métodos. Foi apenas utilizado como outro método as referências mencionadas nos artigos que foram identificados na primeira coluna do diagrama.

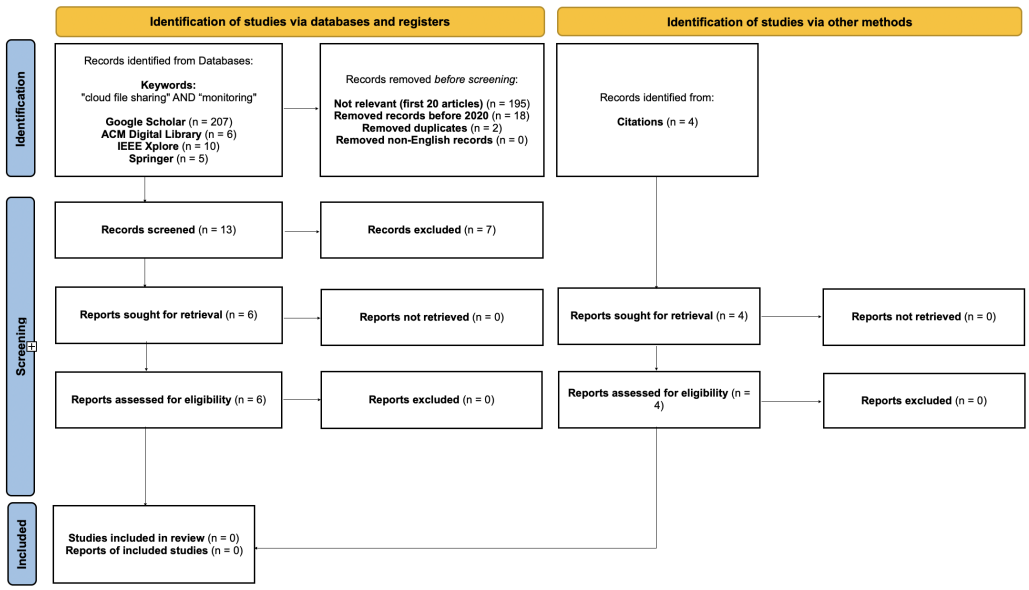

Figura 5: Diagrama do processo de Revisão de Literatura para a ferramenta *Cloud Cleaner*

Para reunir toda a informação relevante resultante do processo de pesquisa, foram utilizadas quatro bases de dados académicas: *Google Scholar* [22], *IEEE Xplore* [23], *ACM Digital Library* [24] e *Springer* [25]. Estas bases de dados são amplamente reconhecidas na comunidade académica por disponibilizarem um vasto número de artigos revistos e validados cientificamente. A pesquisa realizada em todas elas utilizou a seguinte expressão: **"cloud file sharing"AND "monitoring"**.

Numa fase inicial, foram identificados 228 registos nas quatro bases de dados consultadas. Dada a quantidade de resultados obtidos, a pesquisa foi restringida antes da triagem inicial. Para este estudo, foram considerados apenas documentos escritos em inglês e publicados a partir de 2020. Além disso, para limitar ainda mais o número de artigos, apenas os primeiros vinte resultados de cada base de dados, ordenados por relevância, foram incluídos. Após filtrar os registos por relevância, data de publicação, idioma e remover duplicados (encontrados em mais de uma base de dados), o número total de registos analisados reduziu-se a apenas 13.

Dos registos obtidos na etapa anterior, foi feita a leitura do resumo (*abstract*) de cada um para avaliar a sua relevância. Os artigos foram classificados numa escala de 0 a 2. Após esta avaliação, estabeleci que todos os registos com uma pontuação de 1 ou mais seriam considerados suficientemente relevantes para uma análise mais aprofundada. No total, foram selecionados 6 artigos, enquanto os 7 restantes foram descartados.

A etapa seguinte consistiu em obter os artigos selecionados e avaliá-los de forma mais aprofundada, analisando se os seus conteúdos seriam relevantes para o estudo. Como todos os artigos estavam acessíveis, nenhum foi excluído nesta fase. No entanto, durante a leitura detalhada de cada artigo, verifiquei que todos abordavam

temas que não estavam relacionados com a ferramenta em questão, o que levou ao descarte de todos.

Uma possível explicação para o número reduzido de artigos relacionados que foram encontrados pode ser o facto de a ferramenta *Cloud Cleaner* ser altamente específica e objetiva. Esta particularidade limita o campo de investigação, e, consequentemente, o número de artigos disponíveis sobre o tema.

Inicialmente, planeei realizar uma análise PRISMA para cada ferramenta. No entanto, a recolha limitada de artigos através das pesquisas exigiu um ajuste na metodologia da Revisão de Literatura das ferramentas anteriores (*BIC* e *Version Checker*).

3

BULK ISSUE CREATOR

A segurança de aplicações é uma componente essencial no desenvolvimento de software, especialmente na era digital atual, onde ciberataques estão a tornar-se cada vez mais sofisticados e frequentes. Uma etapa crítica neste processo é a análise de vulnerabilidades antes do lançamento das aplicações para produção.

Entre as responsabilidades de um analista de segurança, encontra-se a tarefa de analisar o código da aplicação a lançar e detetar a má implementação de controlos de segurança. Existem várias formas de realizar essa análise, incluindo abordagens automáticas que utilizam ferramentas como "*Scanners* de Segurança" (DAST, SAST, IAST, entre outros), capazes de fornecer relatórios detalhados sobre a postura de segurança da aplicação, identificando falhas e apresentando soluções para mitigar estes riscos.

Por outro lado, existe também a análise manual, que pode ser realizada através da *checklist* OWASP *ASVS*, onde um analista de segurança verifica e valida os controlos de segurança da aplicação. Um procedimento comum que segue a validação manual acima mencionada é registar novos *issues* na *board* do projeto correspondente no *Jira* (sistema de gestão de projetos, referido na secção 3.4.3), um por cada vulnerabilidade detetada.

3.1 OBJETIVO

A ferramenta BIC foi desenvolvida para simplificar e automatizar o processo de *reporting* de vulnerabilidades no *Jira*, facilitando o trabalho dos analistas de segurança. O objetivo principal é otimizar o processo manual de registo de *issues* no *Jira*, reduzindo o tempo despendido e minimizando possíveis erros (derivados ao fator humano).

Após o preenchimento da *checklist* OWASP *ASVS* no ficheiro *Excel* (mencionado na secção 3.4.4) e a identificação das vulnerabilidades, o analista pode utilizar a ferramenta *BIC* para reportar os *issues* automaticamente. Cada execução da ferramenta gera um relatório (ficheiro *.txt*) com uma listagem dos *issues* criados na *board* do *Jira*. Este processo torna-se assim mais eficiente e confiável. Para mais informações, tanto o código da ferramenta como o manual técnico estão disponíveis em [26].

3.2 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Esta secção apresenta os requisitos funcionais e não funcionais identificados para o desenvolvimento desta ferramenta. Estes requisitos são fundamentais para assegurar o correto funcionamento da ferramenta e o cumprimento das necessidades dos utilizadores.

3.2.1 *Requisitos Funcionais*

Os requisitos funcionais descrevem as funcionalidades que a ferramenta deve implementar. Para este projeto, a ferramenta deve:

1. Na primeira execução da aplicação, adaptar a estrutura do ficheiro *Excel* (secção 3.5.1)
2. Ler um ficheiro *Excel* com a estrutura baseada no OWASP *ASVS*.
3. Criar automaticamente *issues* na *board* do projeto correspondente, utilizando a plataforma *Jira*.
4. Gerar um relatório em formato *.txt*, contendo a lista das *issues* criadas na *board* do projeto.

3.2.2 *Requisitos Não Funcionais*

Os requisitos não funcionais definem as condições e restrições que asseguram o desempenho, qualidade e eficácia da ferramenta. Para este projeto, a ferramenta deve:

1. Utilizar obrigatoriamente a *checklist* do OWASP *ASVS*.
2. Exigir que o analista de segurança preencha o ficheiro *Excel*, respeitando a sua estrutura original sem quaisquer modificações.
3. Utilizar a plataforma *Jira* como sistema de gestão de projetos, assegurando que cada projeto possua uma *board* dedicada à criação e monitorização das *issues*.
4. Requerer que o analista de segurança configure as variáveis definidas no início do *script* com os valores específicos do projeto no *Jira*, garantindo uma integração adequada.

3.3 ARQUITETURA E DEFINIÇÃO DE TECNOLOGIAS UTILIZADAS

Fundamental para o desenvolvimento de qualquer ferramenta/software é a sua arquitetura, a forma como está organizado internamente para garantir uma resposta adequada aos requisitos funcionais e não funcionais. Na seguinte Figura 6 é possível visualizar a arquitetura geral da ferramenta BIC.

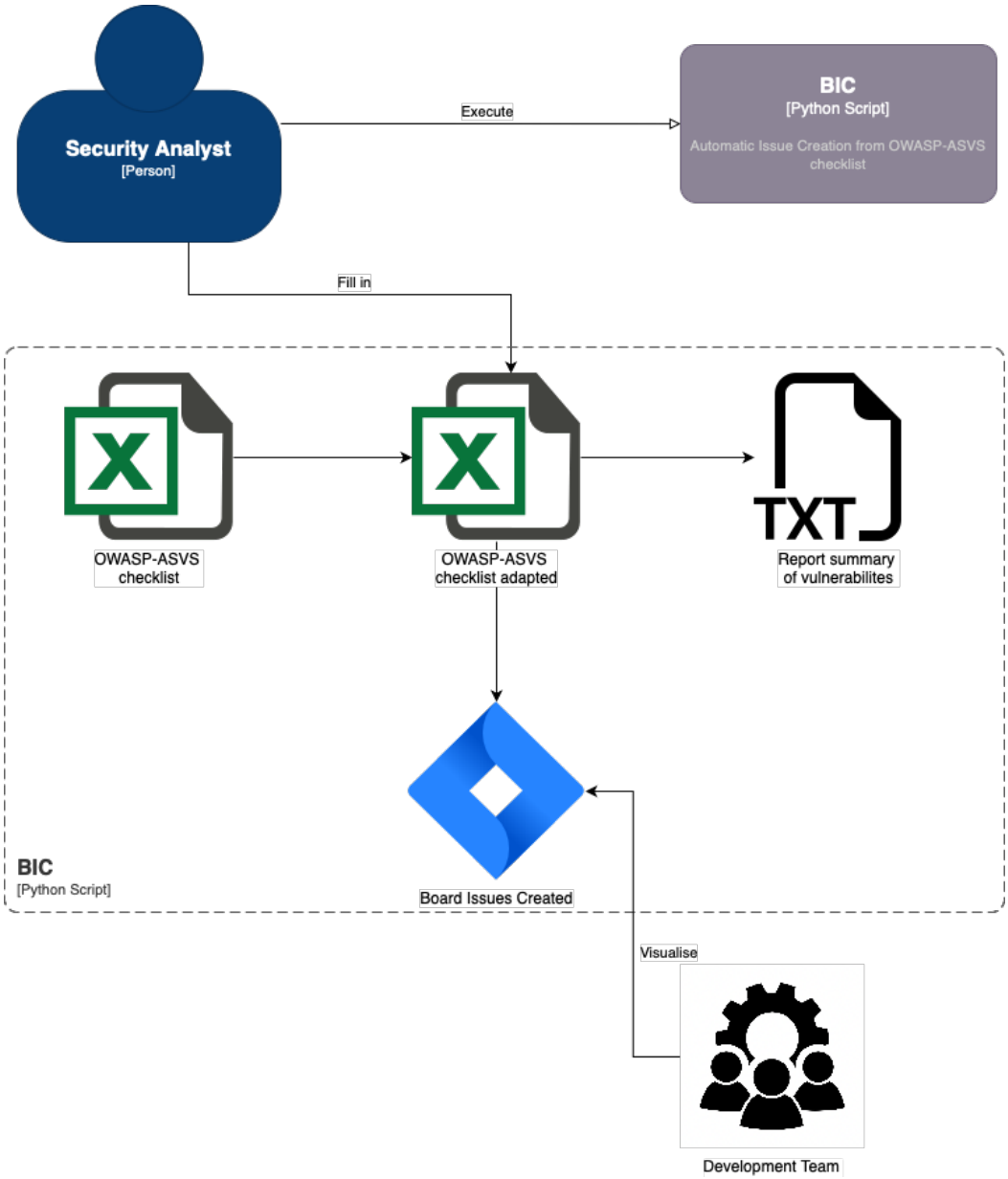

Figura 6: Arquitetura Geral da ferramenta BIC

Nas figuras 7 e 8 é possível observar os diagramas C4 nível um e nível dois, que representam a arquitetura na qual foi baseada o desenvolvimento desta ferramenta.

Na Figura 7 é mostrado o fluxo esperado para a correta execução da ferramenta.

Figura 7: Diagrama C4 nível 1 da ferramenta BIC

A ferramenta é executada por um Analista de Segurança, cargo responsável por analisar e testar o Software, que cria automaticamente *Issues* na *Board* do *Jira*.

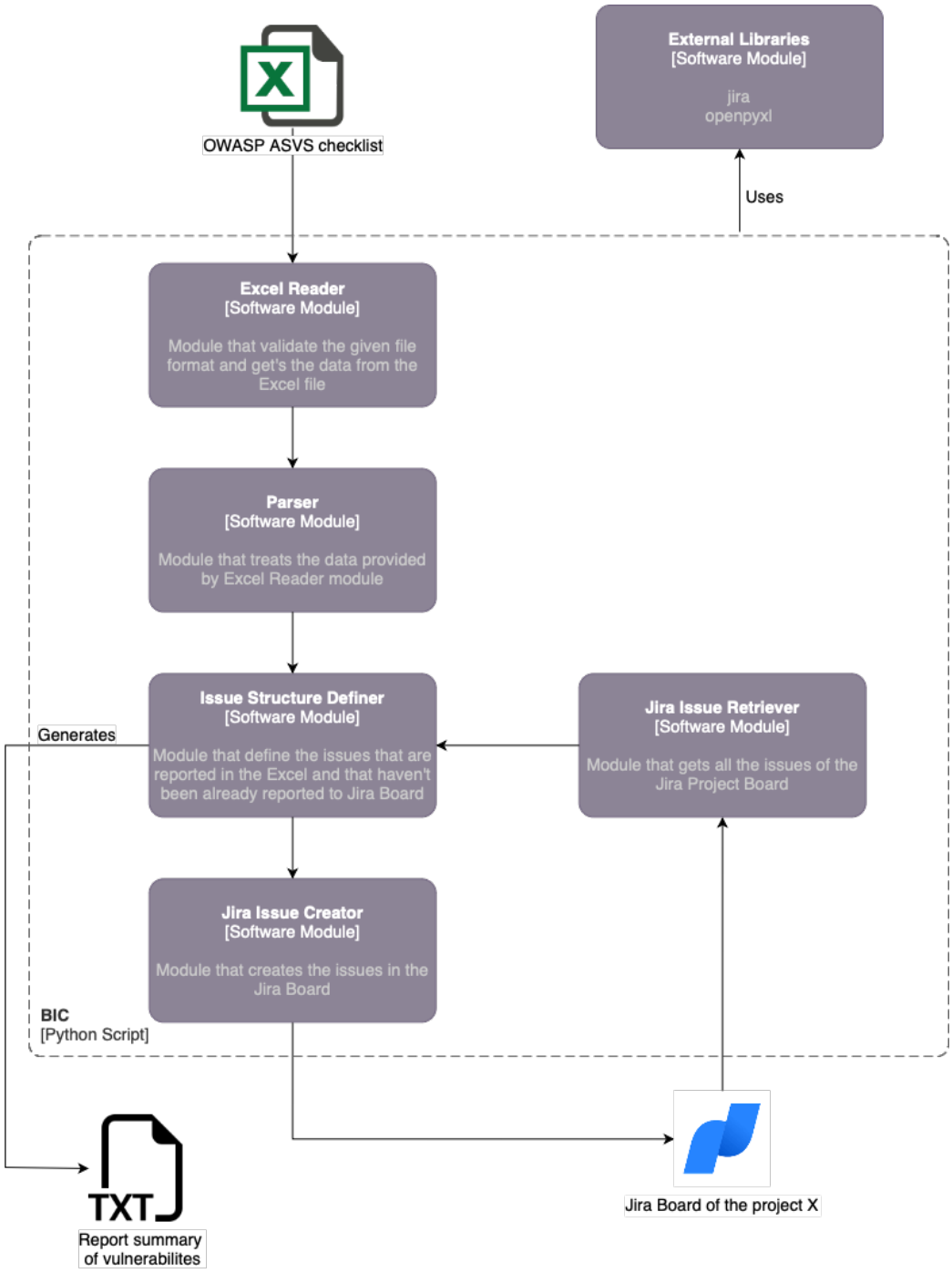

Figura 8: Diagrama C4 nível 2 da ferramenta BIC

Na Figura 8, podemos ver com mais detalhe os módulos que compõem esta ferramenta. Estes são:

1. ***Excel Reader***: Módulo responsável por validar o formato do ficheiro que é fornecido como argumento e por retirar os dados do mesmo.
2. ***External Libraries***: A Tabela 4 mostra as bibliotecas externas utilizadas para o desenvolvimento desta ferramenta.

BIBLIOTECA	VERSÃO	DESCRIÇÃO
jira	3.5.2	Iterage com a *Application Programming Interface (API)* do *Jira* para automatizar a criação dos *issues*.
openpyxl	3.1.2	Biblioteca usada para ler e escrever em ficheiros *Excel*.
requests	2.31.0	Simplifica a realização de pedidos *HTTP*, como *GET* e *POST*, para interagir com *API*.

Tabela 4: Bibliotecas externas utilizadas (BIC)

3. ***Parser***: Módulo responsável por tratar os dados recolhidos do ficheiro *Excel* (através do módulo *Excel Reader*).
4. ***Issue Structure Definer***: Define a estrutura do *issue* para este ser reportado com os atributos corretos preenchidos.
5. ***Jira Issue Retriever***: Parte da aplicação que vai à *board* do projeto e que vai recolher todos os *issues* que já se encontram reportados (independentemente da coluna da onde este se encontra).
6. ***Jira Issue Creator***: Este módulo é responsável pela criação dos *issues* no *Jira* utilizando a *API REST*, com os dados necessários (*summary*, descrição, tipo de *issue*), e depois atualiza a sua prioridade.

3.4 DETALHES DA IMPLEMENTAÇÃO

Para entender melhor o funcionamento desta ferramenta vão ser introduzidos conceitos e ferramentas/*frameworks*.

3.4.1 *Pipeline CI/CD*

Uma pipeline Continuous Integration and Continuous Delivery (CI/CD) é uma metodologia de automação usada no desenvolvimento de software para garantir a correta integração do novo código com o atual e a entrega continua, desde o seu desenvolvimento até a produção, de maneira eficiente, segura e ágil.

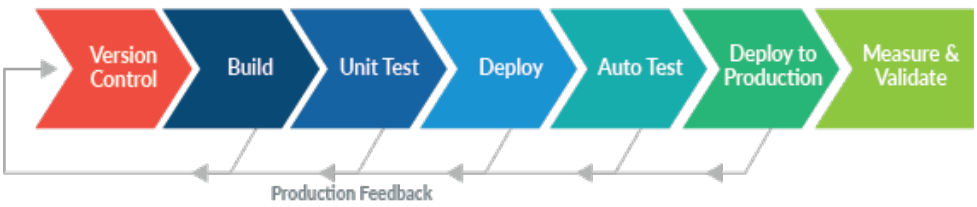

Figura 9: Fases CI/CD *Pipeline* [27]

1. **Integração Contínua (CI - *Continuos Integration*)** - é o processo onde os *developers* integram o seu código ao repositório central de forma frequente, geralmente várias vezes ao dia (por *sprints*). A cada integração, são executados automaticamente testes e *builds* (compilações) para garantir que o novo código é compatível com o existente. Os principais benefícios desta prática são:

    - Deteção antecipada de problemas de integração.
    - Melhoria da qualidade e estabilidade do código.
    - Maior colaboração e eficiência da equipa.
    - Identificação e resolução de erros de forma mais facilitada.

2. **Entrega Contínua (CD - *Continuous Delivery*)** - Na Entrega Contínua, o código que passou pelos testes automatizados na fase de integração é automaticamente preparado para a implementação em ambientes de produção. Assim, o principal objetivo desta metodologia é automatizar o processo de *release* para que as alterações de código estejam sempre num estado implementável, permitindo lançamentos frequentes, fiáveis e de baixo risco para a produção. Existem diversos benefícios, assim como:

    - *Releases* mais rápidos e mais fiáveis
    - Redução dos riscos e erros de implementação
    - Melhoria da qualidade do produto através de atualizações frequentes

Uma *pipeline* CI/CD é composta por várias etapas automatizadas que levam o código desde o desenvolvimento até a produção. Um exemplo simplificado destas etapas encontra-se representado na Figura 10.

Existem diversas ferramentas que ajudam a construir *pipelines* CI/CD, como *Jenkins* [28], *GitLab CI/CD* [29], *Travis CI* [30], *CircleCI* [31] entre outras, cada uma com a sua especificidade.

Desta forma, fica claro que a adoção de práticas como as *pipelines* CI/CD traz inúmeros benefícios ao desenvolvimento dos projetos em geral. Entre eles, destacam-se a agilidade proporcionada pela automação dos processos, a qual permite que o código seja testado e implementado de maneira rápida e eficaz; a redução significativa de falhas em produção, resultado de uma abordagem contínua de testes e correções antes da entrega ao utilizador final; e a maior eficiência, ao eliminar tarefas manuais repetitivas.

3.4.2 *Jenkins*

Jenkins é um software de open-source que funciona como um servidor de automação, facilitando a implementação de CI/CD no desenvolvimento de software.

Quando integrado com *pipelines*, o *Jenkins* automatiza as etapas de *build*, teste e *release*, permitindo que as equipas integrem alterações de forma mais eficiente e forneçam software de maneira fiável.

O conceito de *Pipeline as Code* [32] refere-se a um conjunto de funcionalidades que permitem aos utilizadores (do *Jenkins*) definir processos de trabalho em *pipeline* através de código, que é armazenado e "versionado"num repositório. Esses recursos possibilitam que o *Jenkins* descubra, faça a gestão e execute tarefas em vários repositórios de código-fonte e *branches*, eliminando a necessidade de criar e gerir tarefas manualmente.

A presença de um *Jenkinsfile* na raiz de um repositório torna-o elegível para que o *Jenkins* possa gerir e executar automaticamente os trabalhos com base nos *branches* do repositório, contendo um script de pipeline que especifica os *nodes* (máquinas onde as tarefas serão executadas) e as etapas para a realização das tarefas.

Mais em baixo (na Figura) é possível visualizar vários retângulos (verdes) que mostram o tempo de duração de cada *stage* e clicando nestes, na aplicação, é possível visualizar os vários *steps* pelo qual cada *stage* é composto.

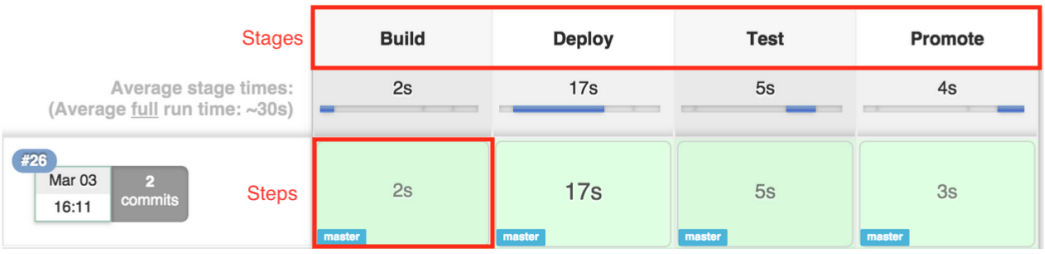

Figura 10: Exemplo de uma *Build* do *Jenkins*

3.4.3 *Jira*

O *Jira* é uma ferramenta popular de software desenvolvida pela *Atlassian*, usada principalmente para a gestão de projetos e acompanhamento de *issues*.

Este tem como principais funcionalidades:

1. **Gestão de Tarefas**: O *Jira* permite criar, organizar e priorizar tarefas, chamadas de *issues*, que podem representar *user-stories*, *bugs* ou qualquer outro tipo de tarefas/itens de trabalho.

2. ***Workflows* Personalizáveis**: outra funcionalidade é a capacidade de definir fluxos de trabalho personalizados, que permitem às equipas mapear o ciclo de vida de uma tarefa, desde a sua criação até a sua conclusão, passando pelos diferentes de estados de revisão, aprovação e testes.
3. **Quadros *Kanban* e *Scrum***: o *Jira* oferece suporte a quadros *Kanban* e *Scrum*, o que permite que as equipas planeiem *sprints*, acompanhem o progresso e façam a gestão do *backlog* de forma visual e eficiente.

3.4.4 *OWASP-ASVS*

O OWASP *ASVS* é uma *framework* criada pela fundação OWASP para a verificação da segurança de aplicações web. Esta fornece um conjunto abrangente de requisitos de segurança que podem ser usados para projetar, desenvolver e testar a segurança de aplicações. Como o nome diz, esta *checklist* foi fornecida pela fundação OWASP[1].

OWASP é uma fundação sem fins lucrativos que trabalha para melhorar a segurança de software. Trata-se de uma comunidade aberta dedicada a permitir que as organizações concebam, desenvolvam, adquiram, operem e mantenham aplicações em que se possam confiar [33]. A fundação OWASP também fornece uma lista que serve como um guia para destacar os riscos mais críticos que os *developers* de software devem abordar para proteger as suas aplicações contra ataques cibernéticos - *OWASP TOP 10* (secção 3.4.5).

Esta *framework* OWASP *ASVS* está disponível em vários formatos, como *Excel*, *Word* ou PDF. Para a utilização desta ferramenta em específico foi utilizado o ficheiro *Excel*.

O principal objetivo desta *framework* é fornecer uma base comum para garantir que as aplicações tenham um nível adequado de segurança. Esta serve como um guia tanto para *Developers* como para Analistas de Segurança, ajudando a identificar e corrigir vulnerabilidades em diferentes níveis de profundidade, dependendo das necessidades da aplicação e do contexto em que ela será utilizada.

Esta *framework* encontra-se dividida em três níveis de *ASVS* sendo estes:

- **Nível 1**: Adequado para aplicações que requerem segurança básica. Este nível cobre os controlos de segurança mínimos e é recomendado para todas as aplicações.
- **Nível 2**: Ideal para aplicações que contêm dados sensíveis e que precisam de maior proteção. Este nível cobre mais controlos de segurança e pode acrescentar rigor a prévios controlos de segurança.

[1] https://owasp.org/

- **Nível 3**: Destinado a aplicações críticas que exigem os mais altos padrões de segurança, como por exemplo aplicações que lidam com informações financeiras, de saúde, ou governamentais. É o nível mais rigoroso e abrangente.

Esta *framework* encontra-se também dividida em diversos capítulos (cada *sheet* do *Excel*), que por sua vez se encontram divididos em várias áreas (cada célula de valor diferente na primeira coluna de cada *sheet*) e que por fim se subdividem por requisitos.

Na figura apresentada no Apêndice D é possível ver o exemplo desta organização referente ao capítulo "*Cryptografy at Rest*".

No que diz respeito às colunas representadas no documento:

- **Area** - Secção do respetivo capítulo (*sheet* selecionada).
- # - Número de identificação de cada requisito.
- **ASVS Level** - Nível do *ASVS* do requisito (valores inteiros presentes no intervalo [1, 3]).
- **CWE** - é uma lista desenvolvida pela comunidade de tipos de pontos fracos de software e hardware. Serve como uma linguagem comum, uma medida para ferramentas de segurança e como uma base para a identificação de pontos fracos, mitigação e esforços de prevenção [34].
- **NIST** - A partir da versão 4.0 do OWASP *ASVS*, os capítulos (*sheets*) "Autenticação" e "Gestão de Sessão" estão em conformidade com o NIST Special Publication (SP) 800-63[1].
- **Verification Requirement** - Descrição do requisito.
- **Valid** - Coluna com o intuito de auxiliar se o controlo de segurança da aplicação Web se encontra ou não bem implementado. Os valores podem ser {*Valid*, *Non-Valid*, *Not Applicable*}.
- **Source Code Reference** - tem como propósito fornecer uma referência a trechos de código ou frameworks específicos que estão relacionados com o controle de segurança em questão. Basicamente, esse campo indica onde no código-fonte ou em quais partes do código o desenvolvedor pode encontrar, aplicar ou verificar a implementação daquele controle de segurança.
- **Comment** - Coluna que pode servir para colocar qualquer comentário auxiliar.

[1] O conjunto da SP 800-63 fornece requisitos técnicos para as agências federais que implementam serviços de identidade digital. A publicação inclui: uma visão geral das estruturas de identidade; utilização de autenticadores, credenciais e afirmações num sistema digital; e um processo baseado no risco para selecionar níveis de garantia [35].

- **Tool Used** - Software/Procedimento que foi utilizado de modo a verificar a implementação do controlo de segurança (ex: Burp Suite, *Developer Tools* – no browser, análise de código, entre outros).

Em relação à coluna "#", o OWASP *ASVS* encontra-se organizado da seguinte forma: <capítulo>.<secção>.<requisito> onde cada elemento é um número inteiro.

- **<capítulo>** - referente ao capítulo onde se encontra o requisito.
- **<secção>** - corresponde à secção dentro desse capítulo onde o requisito aparece
- **<requisito>** - representa o requisito específico dentro dessa secção

Este documento pode ter de ser alterado consoante atualizações feitas ao mesmo, sendo necessário colocar a versão do documento atrás do número de identificação do requisito.

- Exemplo: v4.0.3-3.0.1.

Este documento é um *must* para as organizações, pois trata-se de um método que fornece excelentes bases para ser avaliada a segurança das aplicações que estão a ser desenvolvidas. Ao implementar os controlos de segurança recomendados, as organizações melhoram a postura geral da segurança das suas aplicações.

Dentro da organização esta *framework* é utilizada pelos Analistas de Segurança, que a utilizam como *checklist* de requisitos contra os quais a segurança da aplicação pode ser testada.

3.4.5 *OWASP – Top 10 API Security Risks*

Como referido anteriormente, o *OWASP – Top 10* é uma lista das dez principais ameaças à segurança de aplicações Web, compilada pela *OWASP*. Este documento (versão de 2021) partilha informação sobre os riscos de segurança mais críticos para estas aplicações. Esta publicação define um risco de segurança como um potencial ponto de falha numa aplicação [36].

Os dez principais riscos descritos nesta publicação são:

- **A1**: *Broken Access Control* - impõe políticas de forma a que os utilizadores não possam agir fora das permissões pretendidas. As falhas normalmente levam à divulgação não autorizada de informações, modificação, entre outras.
- **A2**: *Cryptographic Failures* - os algoritmos criptográficos são essenciais para proteger a privacidade e a segurança dos dados. No entanto, estes algoritmos podem ser muito sensíveis a erros de implementação ou configuração.

Esta ameaça inclui a não utilização de cifragem, configurações incorretas de algoritmos criptográficos e gestão insegura de chaves criptográficas.

- **A3**: *Injection* - injeção de dados inválidos com intenção maliciosa.
- **A4**: *Insecure Design* - enquanto muitas das vulnerabilidades na lista lidam com erros de implementação, esta vulnerabilidade descreve falhas na conceção que prejudicam a segurança do sistema. Por exemplo, uma aplicação que armazena e processa dados sensíveis deve incluir um sistema de autenticação
- **A5**: *Security Misconfiguration* - esta vulnerabilidade está relacionada com a má implementação da segurança, coisas como, por exemplo, as contas padrão (com as credenciais *default*) estarem disponíveis ou a não utilização de recursos prescindíveis (por exemplo portos, serviços, contas ou privilégios desnecessários).
- **A6**: *Vulnerable and Outdated Components* - é muito comum os atacantes tentaram inserir código malicioso ou vulnerável em bibliotecas e/ou dependências que são habitualmente utilizadas. A não atualização imediata destas dependências pode deixar vulnerabilidades exploráveis abertas a ataques
- **A7**: *Identification and Authentication Failures* - este risco ocorre quando uma aplicação se baseia em processos de autenticação fracos ou não consegue validar corretamente as informações de autenticação. Por exemplo, uma aplicação que não disponha de autenticação multifator pode ser vulnerável a um ataque de *brute force*, em que um atacante faz um *spam* de tentativas de combinações de nome de utilizador e palavra-passe a partir de uma *wordlists*[1] que contém credenciais fracas, comuns ou predefinidas.
- **A8**: *Software and Data Integrity Failures* - estas "falhas" no software ou na infraestrutura permitem a um atacante modificar ou apagar dados de forma não autorizada. Os atacantes podem explorar estas vulnerabilidades para obter acesso a informações sensíveis ou causar danos ao sistema.
- **A9**: *Security Logging and Monitoring Failures* - muitos incidentes de segurança são possibilitados pelo facto de uma aplicação não registar eventos de segurança significativos ou de estes ficheiros de registo não serem devidamente monitorizados. Este processo é essencial dentro de uma organização para detetar rapidamente um potencial incidente de segurança e responder ao mesmo o mais rápido possível.
- **A10**: *Server-Side Request Forgery* - Este tipo de ataque envolve um atacante que abusa da funcionalidade do servidor para aceder ou modificar recursos. O atacante tem como alvo uma aplicação que suporta a importação de dados

[1] ficheiro com uma lista de palavras que são utilizadas frequentemente como *passwords*. Existem listas destas públicas. A título de exemplo consultar [37]

de Uniform Resource Locator (URL) ou que permite a leitura de dados (de URL). Os URL's podem assim ser manipulados.

3.5 RESULTADOS

Nesta secção, vão ser apresentados os resultados que foram obtidos durante o desenvolvimento da aplicação.

3.5.1 *Alteração da estrutura do Excel OWASP-ASVS*

O primeiro passo no funcionamento da ferramenta corresponde à criação de uma nova estrutura de folha de *Excel*, muito semelhante à original (Figura 11).

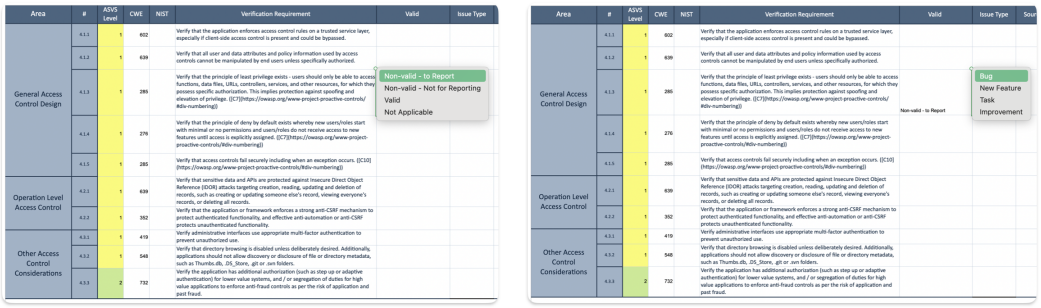

Figura 11: Nova estrutura da Folha de *Excel*

A nova coluna *Valid* tem os seguintes valores *Valid*, *Not Applicable*, *Non-valid - to Report*, *Non-valid - not for Reporting*}. Isto permite ao analista de segurança distinguir entre controlos de segurança que não são validados e aqueles que devem - ou não -, ser comunicados. Apenas os problemas assinalados com *Non-Valid - to Report* serão reportados.

Por outro lado, foi criada uma nova coluna *Issue Type*, que permite ao utilizador definir o tipo de problema que será reportado ao *Jira*. Os valores possíveis são: *Bug*, *New Feature*, *Task* (*default*) e *Improvement*. Relativamente ao projeto *Jira* que servirá de destino, é necessário garantir que os tipos de *issue* referenciados estão corretamente configurados.

3.5.2 *Comportamento da Ferramenta*

Esta ferramenta (BIC) pode ter quatro comportamentos distintos, consoante a situação em que se encontre durante a sua execução. As situações são:

1. Na primeira vez, quando a ferramenta é executada e quando o *Excel* ainda é o de origem, aparece o seguinte output:

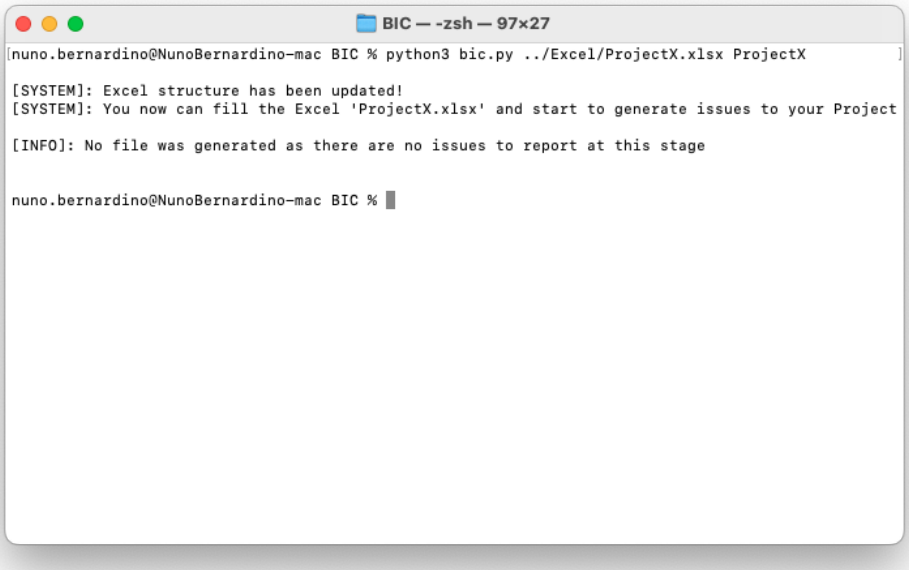

Figura 12: Output 1

A aplicação informa que a estrutura do ficheiro *Excel* foi atualizada para garantir que está corretamente formatada e compatível com a ferramenta. O utilizador pode agora preencher o ficheiro *Excel*, identificado como "ProjectX_xlsx", e, uma vez preenchido, utilizar a aplicação para começar a gerar os *issues* automaticamente para a *Board* do projeto no *Jira*.

Adicionalmente, a mensagem de informação indica que, nesta primeira execução, não foi gerado nenhum *report* com os *issues* reportados, pois ainda não foram identificados ou reportados problemas. Isto é esperado, uma vez que o *Excel* ainda está a ser preparado e não contém dados de *issues* para serem reportados neste momento.

2. Quando a ferramenta é executada, o *Excel* está com a estrutura atualizada mas sem *issues* para serem reportados:

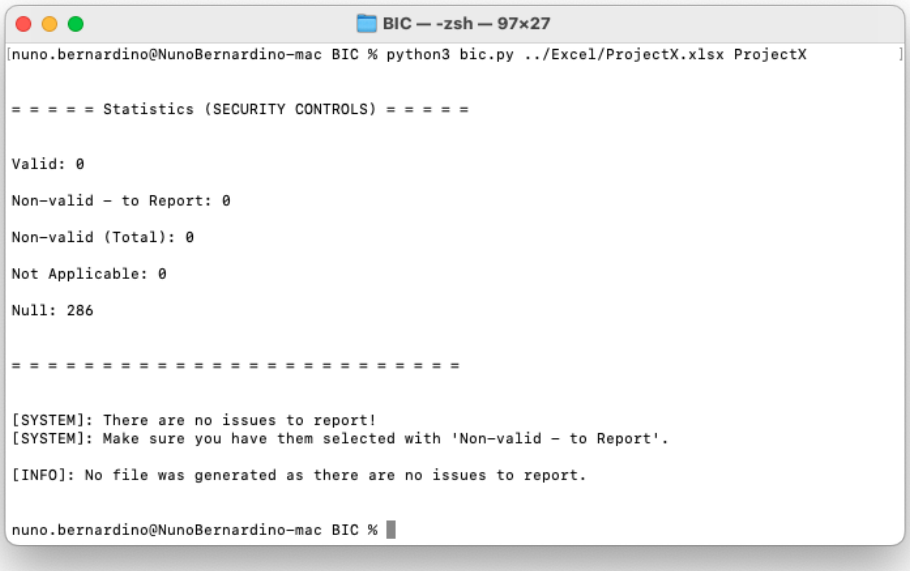

Figura 13: Output 2

Estas mensagens indica que a estrutura do ficheiro *Excel* foi atualizada corretamente e está pronto para ser atualizado. O utilizador pode agora preencher o ficheiro *Excel* para começar a gerar *issues* para o projeto. No entanto, a mensagem de informação avisa que, nesta fase, não foi gerado nenhum ficheiro, pois ainda não existem *issues* a reportar.

Além disso, uma secção no terminal mostra a contagem dos tipos de *issues* processados:

a) Válidos.

b) Não válidos - a reportar.

c) Não válidas - a não reportar.

d) Não aplicáveis.

e) Sem informação (Não preenchidos).

O facto de estes valores anteriormente referidos estarem a zero, significa que, embora a estrutura do *Excel* esteja correta, não foram encontrados dados válidos para serem reportados como *issues*. As 286 (número total de requisitos) entradas nulas sugerem que ainda não foi inserida a informação necessária no *Excel*, e por isso não houve criação de *issues* nesta execução. O utilizador deve preencher o *Excel* com os dados adequados para permitir o correto processamento e reporte das *issues* no *Jira*.

3. Quando a ferramenta é executada, o *Excel* está com a estrutura atualizada e existem *issues* para reportar. É importante notar que ainda não existe nenhum *issue* na *Board* do *Jira*.

   Depois da estrutura do *Excel* já estar atualizada, o mesmo pode começar a ser preenchido.

   De referir que só é criado o *report*, com o(s) *issue*(s) que vão ser criados no *Jira*, quando existe algum para ser reportado (com o estado "*Non valid - to Report*").

   A seguinte Figura 14 mostra um exemplo de como o ficheiro *Excel* pode ser utilizado. Convém mencionar que as únicas modificações que ocorreram no ficheiro, são as que se encontram na página ilustrada pela Figura.

Area	#	ASVS Level	CWE	NIST	Verification Requirement	Valid	Issue Type	Source Code Reference	Comment	Tool Used
General Access Control Design	4.1.1	1	602		Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	Non-valid - to Report	Improvement		Este é o meu comentário #1	
	4.1.2	1	639		Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	Non-valid - to Report	Task		Este é o meu comentário #2	
	4.1.3	1	285		Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege. ([C7](https://owasp.org/www-project-proactive-controls/#div-numbering))	Non-valid - to Report			Este é o meu comentário #3	
	4.1.4	1	276		Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned. ([C7](https://owasp.org/www-project-proactive-controls/#div-numbering))	Non-valid - to Report	New Feature		Este é o meu comentário #4	
	4.1.5	1	285		Verify that access controls fail securely including when an exception occurs. ([C10](https://owasp.org/www-project-proactive-controls/#div-numbering))	Non-valid - to Report	Bug		Este é o meu comentário #5	
Operation Level Access Control	4.2.1	1	639		Verify that sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.	Valid				
	4.2.2	1	352		Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	Not Applicable				
Other Access Control Considerations	4.3.1	1	419		Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	Non-valid - Not for Reporting				
	4.3.2	1	548		Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	Non-valid - Not for Reporting				
	4.3.3	2	732		Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.	Valid				

Figura 14: Exemplo do *Excel* preenchido

Após a execução bem-sucedida da ferramenta, é possível visualizar no terminal, ilustrado nas Figuras 15 a 18 a uma contagem dos *issues* preenchidos no ficheiro *Excel*, classificados por tipo (como mencionado anteriormente).

São também apresentadas as características dos *issues* reportados, incluindo o capítulo, a área, o ID, o nível ASVS, o requisito de verificação, o tipo de *issue* e a sua descrição, que será posteriormente inserida no *Jira*.

Após essa listagem dos *issues* a serem reportados no *Jira*, o utilizador é informado de que a prioridade está a ser atualizada de acordo com o tipo de *issue* (Figuras 17 e 18), seguindo a lógica apresentada na Tabela 5.

Tipo de Issue	**Prioridade**
Bug	Highest
Improvement	Low
New Feature	High
Task	Medium

Tabela 5: Prioridade por tipo de *Issue*

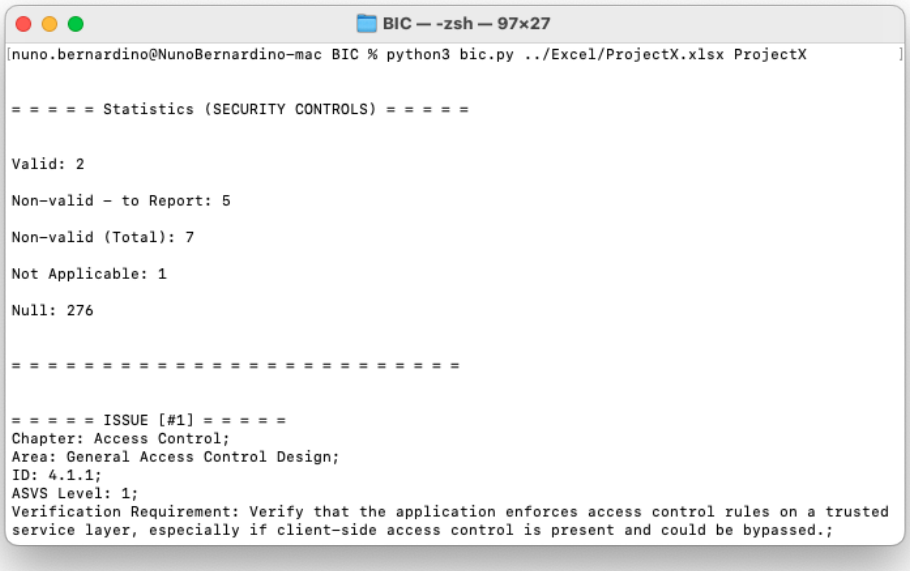

Figura 15: Output 3 - Parte 1

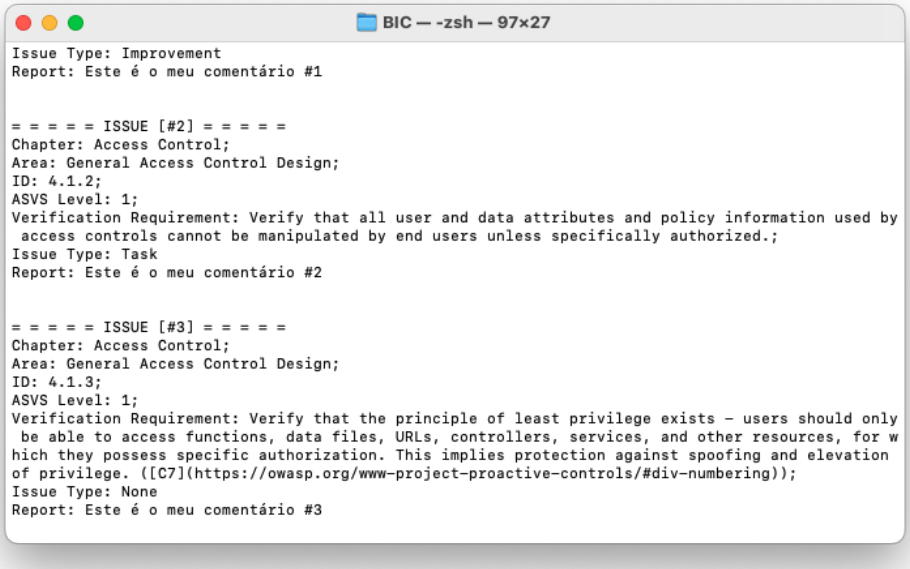

Figura 16: Output 3 - Parte 2

Figura 17: Output 3 - Parte 3

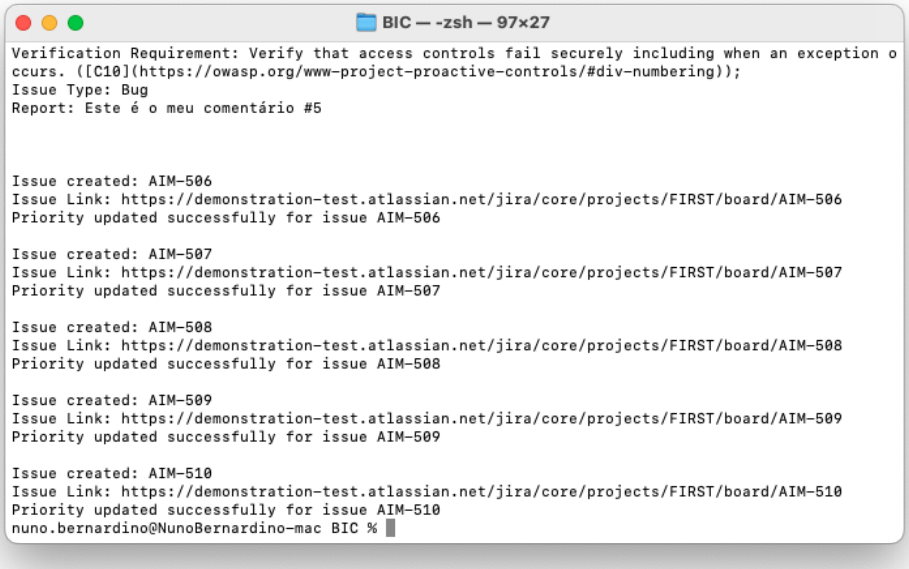

Figura 18: Output 3 - Parte 4

Dessa forma, no *Board* do projeto no *Jira*, é possível verificar que todos os *issues* reportados nos exemplos anteriores são encaminhados para a coluna "*TO_DO*" como mostra a Figura 19 .

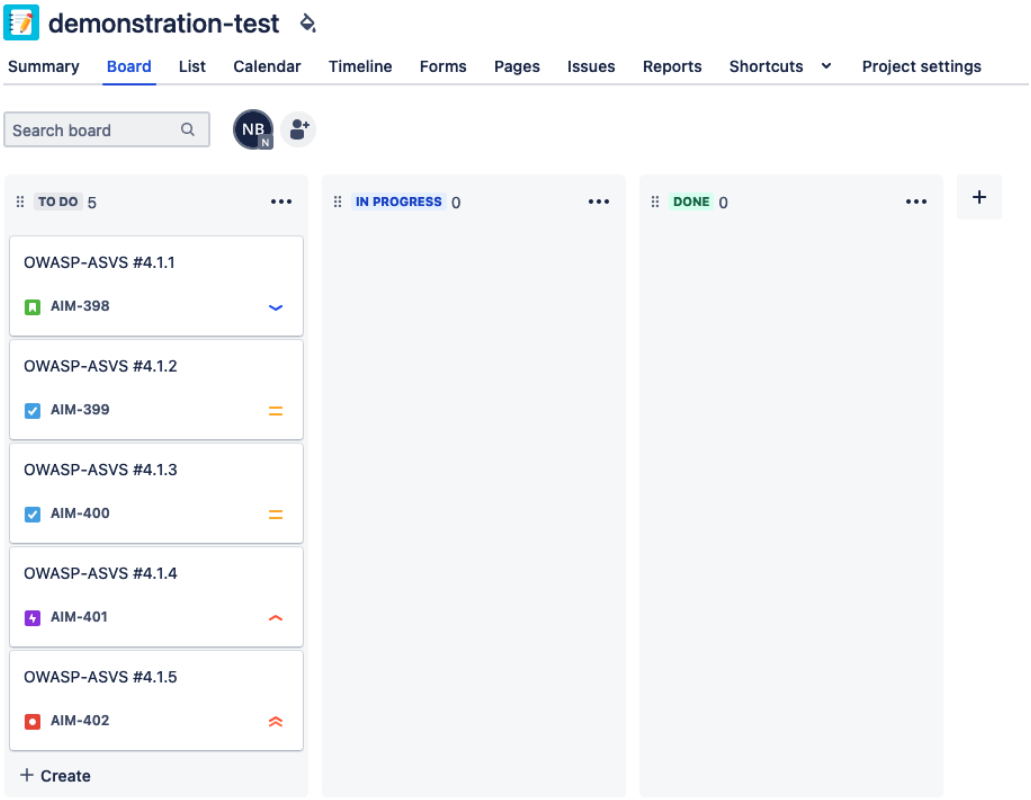

Figura 19: *Jira Board*

A Figura 20 apresenta um exemplo que permite visualizar a estrutura dos *issues*.

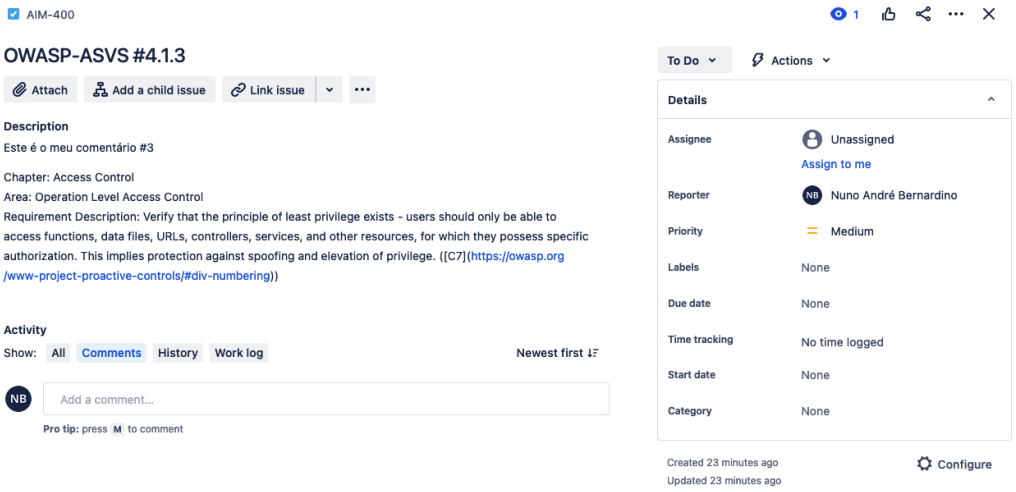

Figura 20: Estrutura dos *issues*

É também gerado um *report* (ficheiro .txt) com a estrutura da Figura 21:

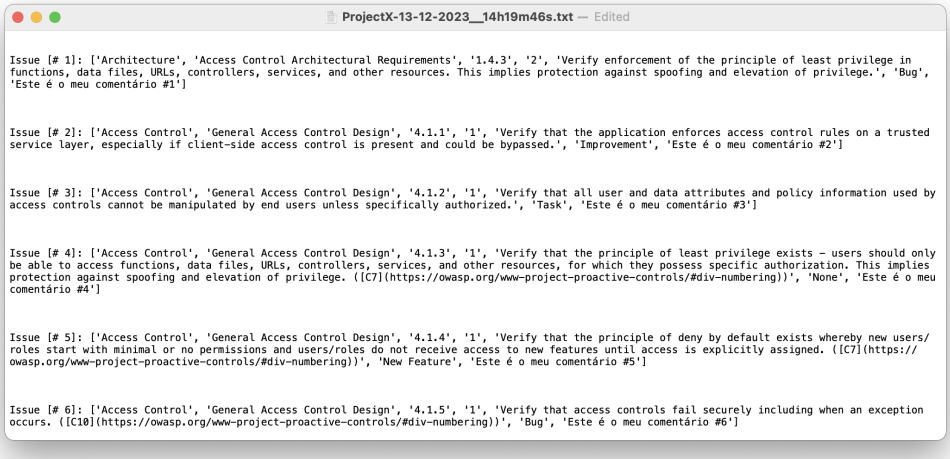

Figura 21: *Report* gerado pela ferramenta

4. Quando a ferramenta é executada e há *issues* que já se encontram no *Jira*: Foi acrescentado um novo *issue* (tipo: Bug) com o ID: #1.4.3 ao mesmo ficheiro utilizado anteriormente, conforme ilustrado na Figura 14.

   O output resultante pode ser observado nas Figuras 22 e 23.

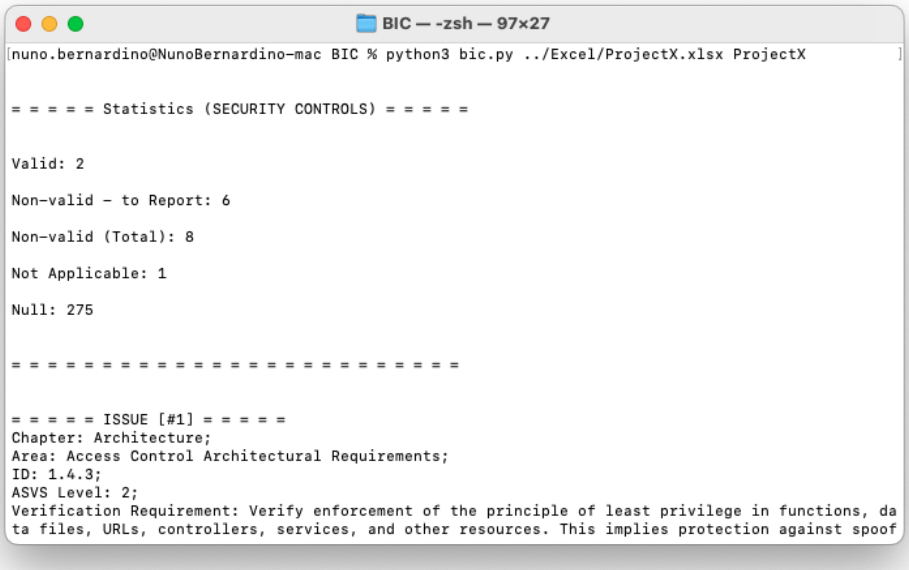

Figura 22: Output 4 - Parte 1

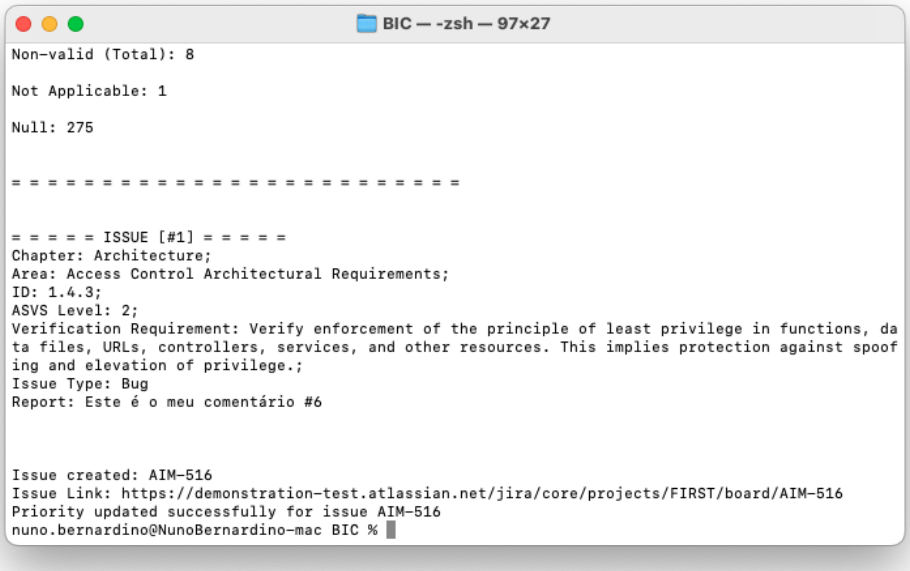

Figura 23: Output 4 - Parte 2

Conforme demonstrado na Figura 24, a ferramenta apenas cria os *issues* que ainda não foram reportados. No *Jira*:

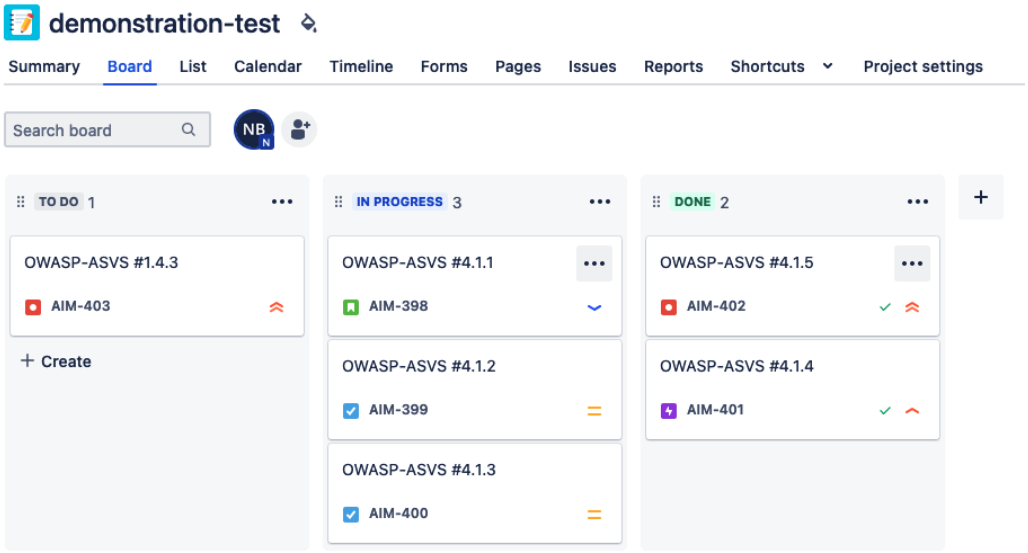

Figura 24: *Update* da *Board* do projeto

Neste capítulo foi apresentada a ferramenta BIC. Esta ferramenta é uma mais-valia para as organizações, pois permite a automatização do processo de *reporting* de vulnerabilidades, reduzindo erros manuais e aumentando a eficiência dos analistas de segurança.

Ao simplificar e acelerar a identificação e comunicação de riscos, as organizações conseguem melhorar a sua postura de segurança, garantir tempos de resposta mais

curtos e, assim, fortalecer a proteção dos seus ativos digitais, promovendo uma maior produtividade e resiliência perante ameaças de cibersegurança.

4

VERSION CHECKER

É muito importante uma organização ter os seus serviços internos atualizados de forma a não deixar vulnerabilidades que possam ser exploradas por atacantes.

Neste contexto, outra das atividades do estágio curricular foi a criação de uma ferramenta com o intuito de verificar se os softwares usados internamente numa organização se encontram na última versão estável dos mesmos.

Para isso foi desenvolvida em *Python*, a ferramenta *Version Checker*.

4.1 OBJETIVO

O principal objetivo desta ferramenta é facilitar o trabalho aos responsáveis pela área da segurança, ajudando-os (de forma automatizada) a perceber o estado das versões dos serviços implementados dentro da organização. Através de um *report* enviado por *e-mail* esta análise/comparação torna-se muito mais prática e simples, neutralizando quase por completo o tempo de execução desta tarefa.

4.2 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Esta secção apresenta os requisitos funcionais e não funcionais identificados para o desenvolvimento desta ferramenta.

4.2.1 *Requisitos Funcionais*

A ferramenta deve ser capaz de:

1. Garantir a segurança na comunicação com os servidores.
2. Visualizar as versões dos serviços implementados nos servidores.
3. Obter automaticamente as últimas versões estáveis de cada serviço através de *web scraping*.
4. Comparar as versões obtidas e gerar uma tabela de comparação clara e de fácil interpretação.

5. Enviar a tabela por *e-mail* para um endereço selecionado no início da execução da ferramenta.

4.2.2 *Requisitos Não Funcionais*

Em termos de requisitos não funcionais, é necessário que:

1. Seja fornecida uma lista de endereços de *e-mail* (num ficheiro de texto). Esta lista permitirá ao utilizador, através de uma Command-Line Interface (CLI), selecionar o endereço que irá receber o resultado da execução da ferramenta, na forma de uma tabela. Cada linha do ficheiro deve conter um único endereço de correio eletrónico.
2. Haja uma chave privada ativa para aceder aos servidores que alojam os serviços. Essa chave será utilizada no processo de autenticação *SSH*, permitindo a verificação das versões atuais dos serviços.

Os requisitos mencionados acima devem ser passados como argumentos na linha de comando.

A ligação à Internet é também um requisito indispensável para a ferramenta funcionar corretamente, pois é necessário obter a informação relativa às versões estáveis dos serviços (online).

É de referir que esta ferramenta está desenhada para a empresa em específico. As empresas não possuem todas os mesmos serviços implementados, e mesmo que possuam, os caminhos (*paths*) para obter a informação dentro dos servidores podem ser diferentes.

4.3 ARQUITETURA E DEFINIÇÃO DE TECNOLOGIAS UTILIZADAS

Na Figura 25 é mostrado o comportamento esperado da execução da ferramenta.

Figura 25: Diagrama C4 nível 1 da ferramenta *Version Checker*

A ferramenta é executada por um *SysAdmin*, que é normalmente o responsável por possuir as permissões necessárias para aceder aos servidores das organizações.

Ao executar a ferramenta, o *SysAdmin* gera um ficheiro Comma-Separated Values (CSV) que contém informações detalhadas sobre o estado dos serviços (em formato de tabela).

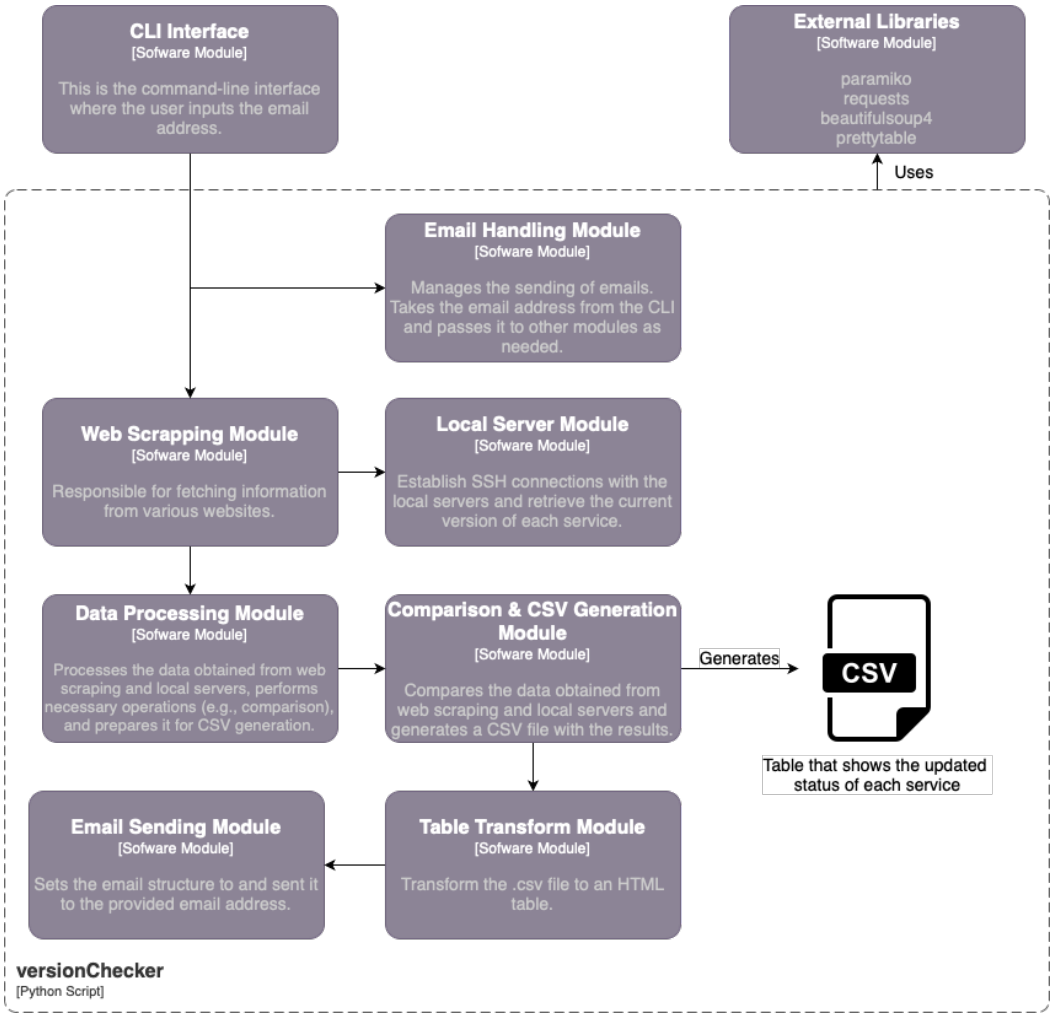

Figura 26: Diagrama C4 nível 2 da ferramenta *Version Checker*

Na Figura 26, podemos ver com mais detalhe os módulos que compõem esta ferramenta. São estes:

1. ***CLI Interface***: Este é o módulo onde o utilizador escolhe de forma interativa o endereço de *e-mail* a partir da lista fornecida como argumento.
2. ***External Libraries***: A Tabela 6 mostra as bibliotecas externas que foram utilizadas.

BIBLIOTECA	VERSÃO	DESCRIÇÃO
beautifulsoup4	4.12.2	Biblioteca para extrair dados de arquivos *HTML* e *XML*, utilizada para *Web Scraping*.
paramiko	2.11.0	Usada para fazer conexões *SSH* e executar comandos remotamente.
preetytable	3.7.0	Ferramenta para exibir dados em formato *tabular* de maneira legível e formatada.
PyInquirer	1.0.3	Biblioteca para criar *prompts* interativos no terminal, recolhendo respostas dos usuários.
requests	2.28.2	Utilizada para fazer pedidos *HTTP* de forma simples e intuitiva.
rich	13.4.1	Utilizada para alterar a fisionomia do terminal, com formatação avançada do mesmo (cores, estilos de letra, entre outros).
typer	0.7.0	*Framework* para criar aplicações de linha de comando CLI com *Python*.

Tabela 6: Bibliotecas externas utilizadas (*Version Checker*)

3. ***Email Handling***: Módulo que lê o *e-mail* escolhido do módulo *CLI Interface*;
4. ***Web Scrapping***: Responsável pela recolha de informação dos vários *websites*;
5. ***Local Server Module***: Estabelece ligações SSH com os servidores locais e obtém a versão atual de cada serviço.
6. ***Data Processing***: Processa os dados obtidos pelo método de extração de *Web Scrapping*, os dados retirados dos servidores locais e prepara os dados para para a criação automatizada do ficheiro CSV.
7. ***Comparison & CSV Generation***: Compara os valores das versões (última versão estável e a implementada) do mesmo serviço e gera o ficheiro CSV.
8. ***Table Transform***: transforma a informação do ficheiro CSV numa tabela HyperText Markup Language (HTML). Este código é escrito diretamente no corpo do *e-mail*;
9. ***Email Sending***: Define a estrutura do *e-mail* e envia para o endereço de *e-mail* recolhido no módulo "*Email Handling Module*".

4.4 DETALHES DA IMPLEMENTAÇÃO

Para entender melhor o funcionamento desta ferramenta vão ser introduzidos vários conceitos que são imprescindíveis para esse feito. Assim vão ser abordados os conceitos *Web Scrapping*, Concorrência e Paralelismo.

4.4.1 *Web Scraping*

Web Scraping é um método de extração de dados de uma página *web*. Este método foi aplicado em todas as páginas/repositório oficiais de cada serviço implementado dentro da organização. Envolvendo operações Input/Output (I/O) como pedidos HTTP e manipulação dos dados recebidos (página HTML) foi-se percorrendo ("raspando") os elementos que constituem esta página até chegar ao elemento final pretendido, a última versão disponível e estável do software.

Ao utilizar este método de extração de informação foram encontradas duas desvantagens que acabaram por ser resolvidas. Estas foram:

1. **Alteração da página HTML**: Pode acontecer uma organização responsável por um destes serviços, colocar o *website* em manutenção (é costume alterarem a página HTML para colocarem uma nova com a informação relativa ao acontecimento). Isto, ou podem simplesmente alterar a estrutura da página ou a mesma estar em baixo.

   De forma a ultrapassar este entrave inicial, sempre que a ferramenta é executada é gerado um ficheiro CSV com os valores dessa execução.

   Assim, nas próximas execuções, sempre que é consultada uma página *Web* e esta se encontra em baixo (ou alterada), vai ao *report* da última execução (se o ficheiro não tiver sido apagado) e vai buscar os valores em falta. Quando isto acontece, o *Status* do serviço em falta é atualizado. Independentemente das versões coincidirem ("*Latest Version*" e "*Current Version*"), o *Status* dessa entrada da tabela vai ficar a amarelo, com o intuito de alertar o utilizador de que o campo "*Latest Version*" não foi retirado com sucesso da página *web*.

   Caso o ficheiro CSV não exista, simplesmente imprime a String "???" na célula correspondente ao valor em falta. É também alterado o *Status* dessa linha (para a cor amarela) de forma a alertar o utilizador do sucedido.

   Quando o *Status* de alguma entrada do *Excel* se encontra a amarelo, é essencial o utilizador verificar o porquê e, caso a página HTML se tenha modificado, vai ser necessário alterar o código da função correspondente para esta ir aceder aos elementos pretendidos e retirar a informação da versão correta.

   Quando a aplicação completa a sua execução, todos os valores do *report* são atualizados, ou seja, são escritos por cima do *report* antigo.

   Nestes casos os elementos e as propriedades da página *web* são alterados, logo não vão conseguir ser acedidos os elementos (da página HTML) que foram definidos inicialmente.

2. **Tempo de execução da ferramenta**: Esta ferramenta consulta trinta e quatro *websites* de serviços distintos. É percetível que o tempo de execução da mesma seja ligeiramente longo devido ao grande número de páginas *web* a que extrai informação. Para combater este entrave, foram abordadas diferentes técnicas, que podem ser consultadas na secção 4.4.2. Na mesma secção é realizada uma comparação entre os tempos de execução consoante os diferentes tipos de abordagem por forma a combater esta limitação.

4.4.2 *Concorrência vs Paralelismo*

O paralelismo no contexto da programação, e no que toca às *threads*, refere-se à execução simultânea de várias operações ou tarefas. As *threads* são unidades de execução que podem ser executadas concorrentemente dentro de um processo.

A carga de trabalho de uma tarefa é, assim, dividida em partes menores, que podem ser executadas simultaneamente, o que acaba por melhorar o desempenho e a eficiência do programa.

Neste caso, em vez das trinta e quatro funções (trinta e quatro serviços) que vão buscar o valor da última versão disponível ("*Latest Version*") terem que esperar umas pelas outras, foram criadas seis *threads*, executando cada uma, um número diferente de funções (foi feito um estudo em relação às que demoram mais tempo a serem executadas). Assim, são executadas estas seis *threads* ao mesmo tempo.

À primeira vista, pode parecer que a concorrência e o paralelismo podem estar a referir-se aos mesmos conceitos. No entanto, estes termos são efetivamente diferentes.

Uma *CPU* executa uma tarefa de cada vez. Se lhe forem atribuídas várias, a *CPU* alterna entre a execução de cada tarefa. A concorrência no fundo é uma "ilusão" de múltiplas tarefas a correr em paralelo devido a uma comutação muito rápida por parte da *CPU*. Assim, a concorrência não é mais do que fazer "*pause*" e "*resume*" a tarefas distintas.

As Figuras 27 e 28 ilustram o processo de execução de três tarefas distintas:

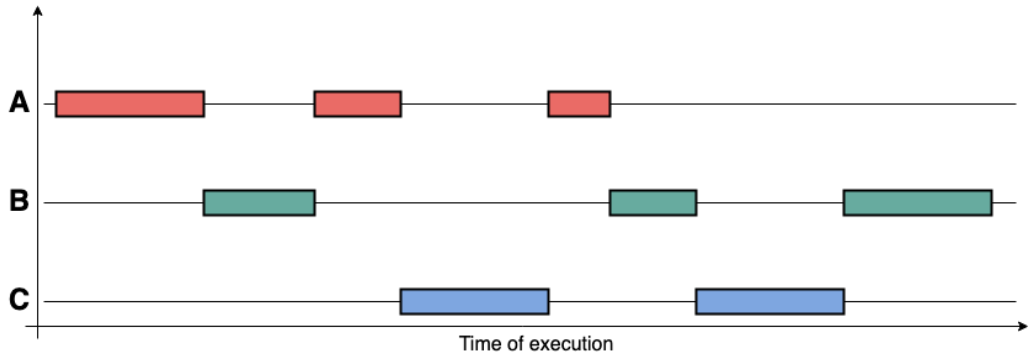

Figura 27: Concorrência #1

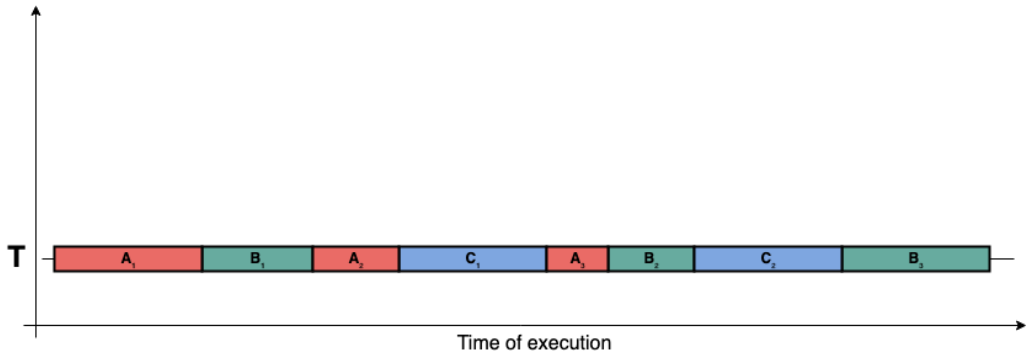

Figura 28: Concorrência #2

O Paralelismo é um tipo de computação em que vários núcleos da *CPU* (ou vários *CPU*) executam vários processos ao mesmo tempo.

A Figura 29 ilustra este tipo de computação. Em comparação com as Figuras 27 e 28 é de notar a diferença de tempo de execução.

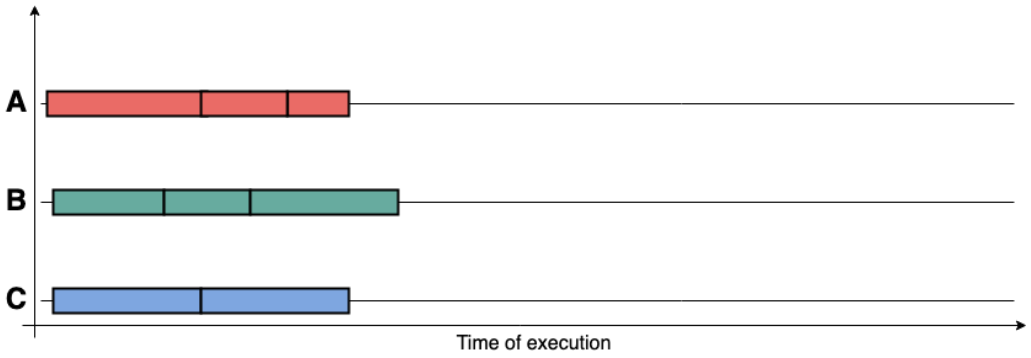

Figura 29: Paralelismo

No caso desta aplicação, o paralelismo foi utilizado com o intuito de ser possível descarregar o conteúdo das páginas HTML dos diferentes serviços e fazer a filtragem dos seus elementos de forma muito mais rápida e eficaz.

4.5 RESULTADOS

Como referido anteriormente, o tempo de execução da ferramenta originalmente era algo demorado. Assim, foram experimentadas duas abordagens para tentar colmatar esta falha.

4.5.1 *Biblioteca asyncio*

A biblioteca *asyncio* [38] é um módulo que fornece suporte para programação assíncrona, permitindo a execução concorrente de tarefas sem a necessidade de múltiplas *threads* ou processos.

Esta biblioteca é particularmente útil para lidar com operações de I/O intensivas, como requisições de rede, leitura/gravação de ficheiros e interações com BD.

Inicialmente parecia-me estar a ir na direção certa, mas após a conclusão da implementação do código da ferramenta, rapidamente percebi que não se notava grande diferença nos tempos de execução, como se pode ver mais à frente na Tabela 7.

4.5.2 *Threads*

No contexto desta ferramenta, o uso de *threads* oferece uma abordagem paralela, permitindo que várias tarefas ocorram simultaneamente (como referido anteriormente).

Ao invés de adotar uma abordagem sequencial, na qual cada função é executada uma após outra, as *threads* vão permitir-nos realizar várias operações em simultâneo. Esta característica é particularmente valiosa ao lidar com um grande número de sites.

Enquanto uma *thread* está à espera de uma resposta do servidor de um site, outras *threads* continuam a execução eficientemente preenchendo os períodos de inatividade. Esse paralelismo resulta numa utilização mais eficaz da capacidade computacional, reduzindo o tempo total de execução da ferramenta.

Para este efeito, a gestão de *threads* foi facilitada por meio da biblioteca "*threading*"[39], que oferece um conjunto de ferramentas para criar e controlar *threads* de maneira simples e eficaz.

Ao incorporar este método na ferramenta, acabamos por transformar uma abordagem linear e em série numa implementação paralela e ágil.

Na Tabela 7 podemos ver que o tempo de execução da ferramenta diminui de uma maneira muito significativa.

Método	Tempo médio (10 execuções)
código	45.814s
async.io	47.469s
threads	4.216s

Tabela 7: Tabela comparativa de tempos de execução

4.5.3 *Apresentação da Informação*

Esta ferramenta tem como resultado direto uma tabela (Tabela 8) as seguintes informações:

- ***ID***: número de identificação do serviço;
- ***Service***: nome do serviço;
- ***Type***: para identificar o serviço de forma mais específica, caso exista mais do que uma instância do mesmo implementado;
- ***Current Version***: versão do serviço que se encontra instalada na infraestrutura da empresa;
- ***Latest Version***: última versão estável do respetivo serviço (retirada online);
- ***Status***: valor que altera consoante os valores da versão implementada e a ultima versão estável disponível. Os valores desta coluna podem ser:
  1. 🟢 : "Serviço atualizado!"
  2. 🟡 : "Não foi possível recolher as informações diretamente da fonte. Versão recolhida a partir do report gerado da última execução da ferramenta."
  3. 🔴 : "Serviço desatualizado! Está disponível outra versão mais atual!"
- ***LATEST_UPDATED_AT***: coluna que informa a última vez que conseguiu ir ao *site* oficial de cada serviço retirar a informação correta da última versão.

ID	Service	Type	Current Version	Latest Version	Status	LATEST_UPDATED_AT
1	Service #1	N/A	v1.21.1	v1.21.1	🟢	2023-11-12 16:00:00
2	Service #2	N/A	v1.5.2	v1.5.8	🔴	2023-11-12 16:00:00
3	Service #3	1	v2.3.3	v2.3.3	🟡	2023-11-05 16:00:00
4	Service #3	2	v2.3.3	v2.3.3	🟡	2023-11-12 16:00:00
5	Service #4	Internal	v6.2.4	v6.2.4	🟢	2023-11-12 16:00:00
6	Service #4	External	v6.2.4	v6.2.4	🟢	2023-11-12 16:00:00
7	Service #5	N/A	v17.0	v18.0	🔴	2023-11-12 16:00:00
8	Service #6	N/A	v6.2	v6.2	🟢	2023-11-12 16:00:00

Tabela 8: Estrutura da tabela

A ferramenta de comparação de versões de serviços internos desenvolvida, representa um avanço significativo na gestão de TI dentro da organização. A sua capacidade de automatizar a verificação de atualizações, aumentar a segurança,

melhorar o desempenho e poupar tempo faz dela um recurso indispensável para os administradores de sistemas. Ao garantir que os serviços internos estejam sempre atualizados com as últimas versões disponíveis, a ferramenta não só poupa trabalho e tempo, mas também contribui para um ambiente de TI mais seguro e eficiente, beneficiando toda a organização.

5

CLOUD CLEANER

Num ambiente empresarial cada vez mais digitalizado e interligado, a partilha de informações a nível interno (entre colaboradores) e externo (com os clientes) torna-se essencial para evoluir de forma colaborativa.

Para facilitar esta partilha de informação, as organizações podem recorrer à implementação de serviços de *cloud*. As informações sensíveis e confidenciais armazenadas na *cloud* estão sujeitas a ameaças internas e externas, como *data leaks* ciberataques ou utilização indevida.

As organizações implementam políticas de acesso restrito e permissões baseadas em perfis, equipas, funções e Access Control List (ACL) (listas de controlo de acesso). Assim, podem reduzir significativamente o risco de exposição e proteger os seus ativos mais importantes.

5.1 OBJETIVO

O controlo das pastas partilhadas é essencial para preservar a integridade e a confidencialidade dos dados. Ao monitorizar estas pastas e quem tem acesso às mesmas, as organizações podem garantir que apenas a informação necessária/prevista é visível e partilhada.

O objetivo desta ferramenta, *Cloud Cleaner* é facilitar o controlo dos conteúdos que estão a ser partilhados dentro das pastas que se encontram na(s) *cloud*(s) da organização, fazendo com que cada utilizador seja responsável pelas suas próprias ações. Assim, cada utilizador da *cloud* receberá uma mensagem de correio eletrónico com **n** hiperligações, uma por cada pasta com *Shares* expirados. Consideram-se *Shares* expirados aqueles cuja partilha ultrapassa trinta dias desde a data de início.

5.2 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Esta ferramenta deve atender aos seguintes requisitos funcionais:

1. **Exportação de Utilizadores e Objetos *Share***: a ferramenta deve possibilitar a exportação dos utilizadores e dos objetos *Share* do serviço de *cloud*.

2. **Visualização de Objetos *Share***: a ferramenta deve permitir que os utilizadores da *cloud* vejam todos os objetos *Share* relacionados com as suas pastas (sejam estes expirados ou não).
3. **Autenticação**: implementação de autenticação através do serviço de autenticação utilizado dentro da empresa - *Keycloak*[40].
4. **Funcionalidades para todos os Utilizadores**:
   - **/folders**: ver todas as pastas da *cloud* em que o utilizador é dono.
   - **/folders/file_id**: mostrar todos os objetos *Share* por pasta.
5. **Funcionalidades exclusivas para *Admins***:
   - **/admin/users**: mostrar todos os utilizadores na instância da *cloud*.
   - **/admin/shares**: mostrar todos os *shares* na instância da *cloud*.
   - **/admin/folders**: mostrar todas as pastas de todos os utilizadores.
   - **/sendmails**: *trigger* para enviar os *e-mails* para os utilizadores designados.

De seguida são apresentados os requisitos não funcionais identificados para a ferramenta.

1. **Desenvolvimento para um serviço de *cloud* específico**: esta ferramenta foi desenvolvida para ser utilizada para com o serviço utilizado dentro da entidade de estágio - *Nextcloud* [41].
2. ***Share Listing***: o serviço da *Nextcloud* permite a instalação de aplicações/*plugins*. Para o correto funcionamento desta aplicação é necessária a instalação da aplicação "***Share Listing***" [42]. Esta aplicação permite gerar uma lista de *Shares* para apresentação na linha de comando.
3. **Execução do *Script***: o *script* deve ser executado com privilégios *sudo*[1] ou com o utilizador *www-data* (*default*) no servidor onde a instância do serviço *Nextcloud* está instalada.

   Esta medida garante que o *script* tenha as permissões necessárias para aceder e modificar os ficheiros e configurações exigidas pela ferramenta.
4. **Utilização de uma Base de Dados *MariaDB***: A BD utilizada pela ferramenta tem de ser *MariaDB*[43]. Serviço este também utilizado dentro da organização.

Estes requisitos não funcionais são essenciais para garantir que a ferramenta funcione de maneira eficiente.

[1] O comando *sudo*, utilizado em sistemas Linux, permite que um utilizador autorizado execute comandos como super utilizador ou outro utilizador autorizado.

5.3 ARQUITETURA E DEFINIÇÃO DE TECNOLOGIAS UTILIZADAS

É possível fazer um apanhado de toda esta informação e resumir a arquitetura desta ferramenta nos seguintes diagramas C4 nível 1 e 2, Figuras 30 e 31.

Na Figura 30 é mostrado o comportamento esperado da execução da ferramenta.

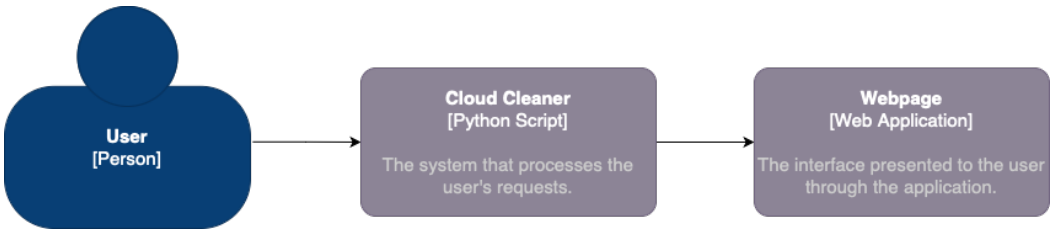

Figura 30: Diagrama C4 nível 1 da ferramenta *Cloud Cleaner*

A disposição do diagrama linear mostra o utilizador à esquerda a interagir com a aplicação *Web Flask* no meio, que por sua vez comunica com a página *Web* à direita, ilustrando o fluxo de pedidos e respostas entre estes componentes.

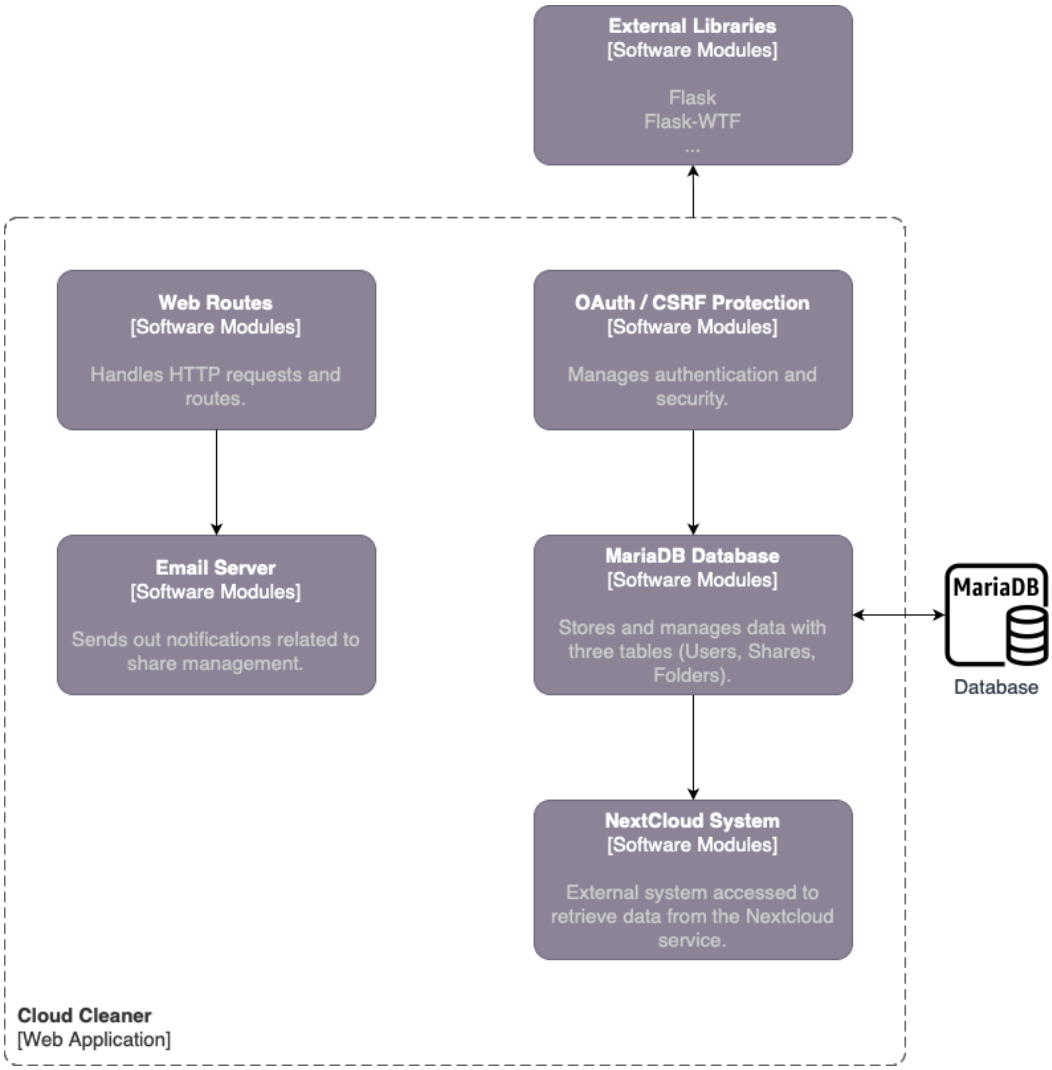

Figura 31: Diagrama C4 nível 2 da ferramenta *Cloud Cleaner*

Na Figura 31, podemos ver com mais detalhe os módulos que compõem esta ferramenta. São estes:

- ***Web Routes***: Lida com os *requests HTTP* e com as rotas da aplicação *Web*.
- ***Email Servers***: Envia notificações relacionadas com a gestão de partilhas para os respetivos utilizadores.
- **Bibliotecas Externas**: A tabela 9 mostra o nome da biblioteca, a versão utilizada e uma pequena descrição da mesma.

Biblioteca	Versão	Descrição
Flask	2.3.4	*Framework* para construir aplicações web em *Python*.
Flask-WTF	1.1.1	Extensão para *Flask* que integra o *WTForms*, oferece proteção *CSRF* para aplicações *Flask*.
mysql-connector	8.1.2	Serve para conectar e interagir com *BD MySQL* a partir de aplicações *Python*.
Authlib	1.1.0	Biblioteca para autenticação e autorização em aplicações web.
requests	2.31.0	Biblioteca para fazer pedidos *HTTP* de forma simples e intuitiva em *Python*.

Tabela 9: Bibliotecas externas utilizadas (Cloud Cleaner)

- ***OAuth/CSRF Protection*** - Faz a gestão da autenticação dos utilizadores.

- ***MariaDB Database*** - Armazena e faz a gestão dos dados com três tabelas (*Users*, *Shares* e *Folders*).
- ***NextCloud System*** - Este módulo faz a gestão dos dados recolhidos do serviço de *cloud*.

Como referido, foi utilizada uma BD (*MariaDB*) para armazenar alguns dados de forma a facilitar o processo de desenvolvimento da ferramenta. O Diagrama de Entidade e Relacionamento (DER) da BD está representado na Figura 32.

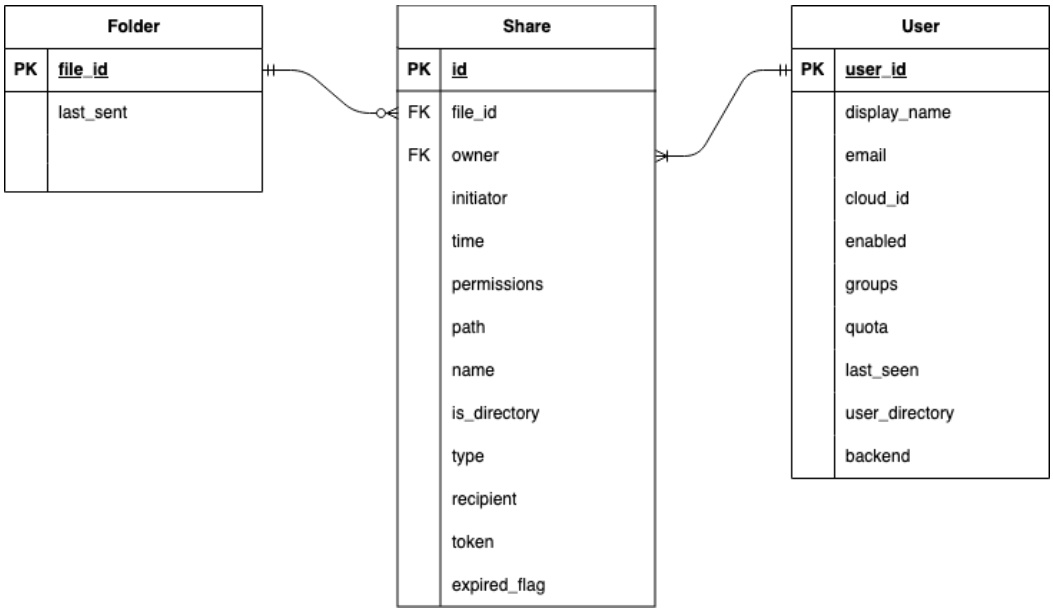

Figura 32: DER da ferramenta *Cloud Cleaner*

Nesta imagem podemos ver que a informação se encontra armazenada em três tabelas distintas. A tabela "*Folder*" armazena o *timestamp* da última vez que foi enviado um *e-mail* para o dono da pasta. As tabelas "*Share*" e "*User*" que mostram os campos, nos quais é aramazenada toda a informação relativa aos objetos *Share* e aos utilizadores do sistema, respetivamente.

5.4 DETALHES DA IMPLEMENTAÇÃO

Nesta secção, serão apresentados os detalhes de implementação da ferramenta, incluindo as decisões técnicas tomadas para o seu desenvolvimento e as tecnologias utilizadas para garantir eficiência e segurança.

5.4.1 *Docker*

O *Docker* [44] é uma plataforma de software (*open-source*) que permite criar, implementar e fazer a gestão de aplicações em *containers*, que são unidades leves e autónomas com tudo o que é necessário para a aplicação funcionar, como código, dependências ou serviços. Esta tecnologia garante que as aplicações funcionam de forma consistente em qualquer ambiente, desde o desenvolvimento até à produção.

Os *containers* oferecem várias vantagens, como portabilidade, permitindo que sejam executados em diferentes infraestruturas sem ajustes; escalabilidade, facilitando o ajuste à procura; e isolamento, prevenindo conflitos entre aplicações. Assim, o *Docker* simplifica o desenvolvimento e a gestão de aplicações, tornando os processos mais eficientes e reduzindo custos operacionais.

5.4.2 *Arquitetura dos containers*

Para explicar a arquitetura de um *container* no *Docker*, podemos dividi-la em três partes principais: Imagem, Automação (*Dockerfile*) e *Runtime*.

As imagens são como modelos de *containers* que contêm tudo o necessário para executar uma aplicação, desde sistemas operativos básicos até aplicações específicas. As imagens seguem uma estrutura hierárquica, onde elementos mais específicos derivam de bases mais genéricas.

Uma das grandes vantagens das imagens é a sua portabilidade, permitindo que sejam criadas num ambiente baseado em *Linux* e executadas noutras distribuições sem problemas, já que os *containers* partilham o *kernel* do sistema operativo, dispensando um sistema operativo específico para cada aplicação.

Para organizar e gerir as diversas imagens, são utilizados repositórios que as agrupam e permitem a sua gestão através de operações como *pull* e *push*. Esses repositórios estão alojados em "*Registries*", que são locais de armazenamento que podem ser públicos ou privados.

Para o desenvolvimento desta ferramenta foram utilizados três *containers*:

1. ***Nextcloud*** [45]: serviço de *cloud* utilizado dentro da organização.
2. ***Keycloak*** [46]: serviço de autenticação, também este utilizado na organização.
3. ***MariaDB*** [47]: serviço de BD designado.

5.5 RESULTADOS

Nesta secção, serão apresentados os resultados obtidos com o desenvolvimento da ferramenta anteriormente descrita. Vão ser exibidas as vistas atualmente existentes.

Primeiramente vão ser mostradas as vistas que todos os utilizadores, sejam administradores (*admin's*) ou não, podem visualizar.

1. **Página Inicial (sem *login* efetuado)**:

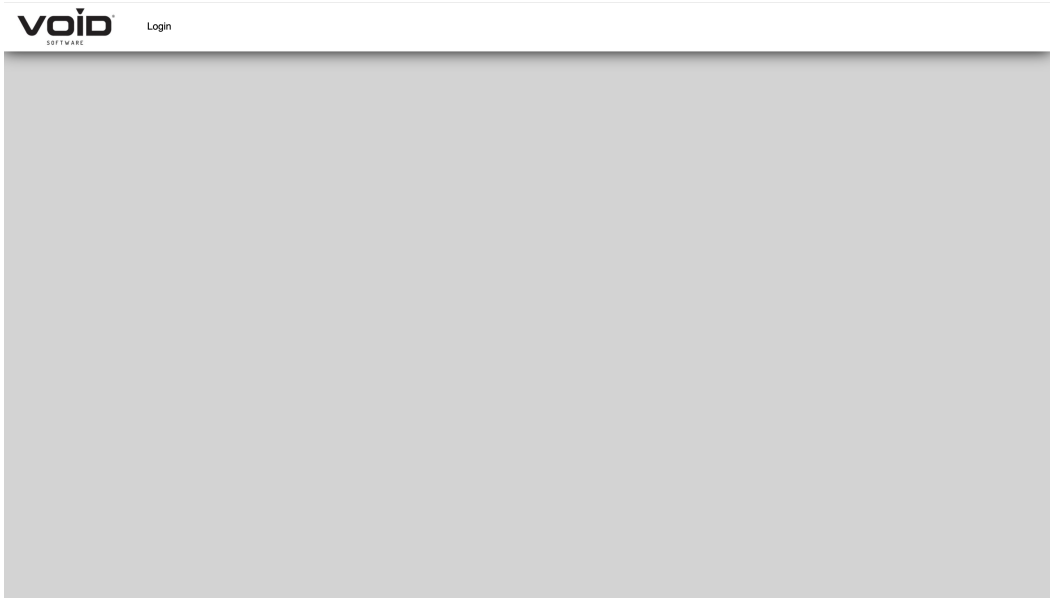

Figura 33: Página Inicial - sem *login* efetuado

   Ao clicar no botão "*Login*", o utilizador será redirecionado para a página do serviço de autenticação do *Keycloak*, conforme ilustrado na Figura 34.

2. **Redirecionamento para o serviço de autenticação**:

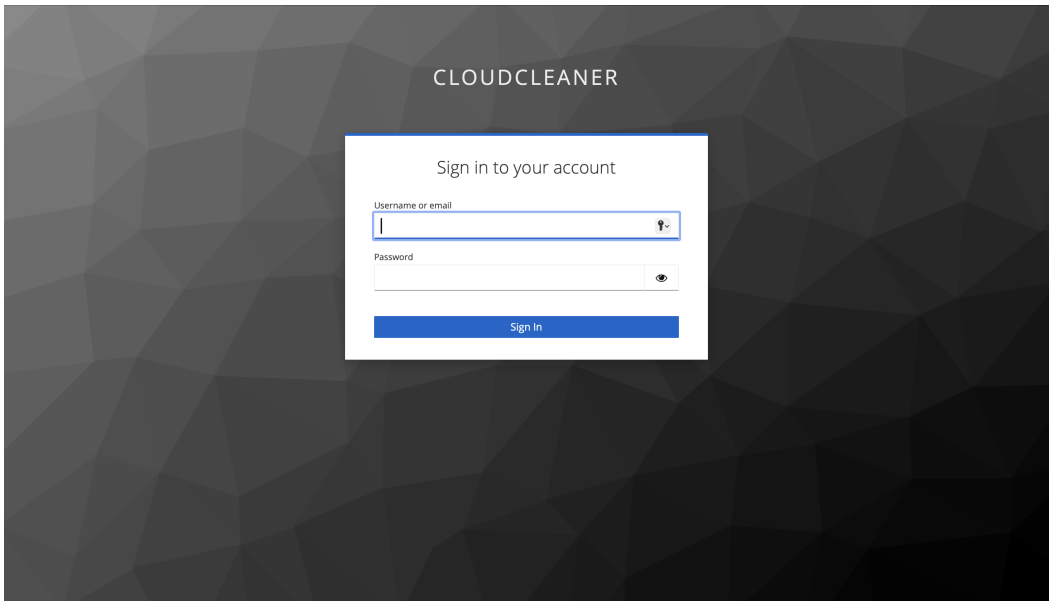

Figura 34: Autenticação do utilizador

3. **Pastas na *Cloud***:

   Assim que o utilizador (neste caso sem ser *administrador*) se autentica com sucesso é possível ver, na página principal, todas as pastas que este possui na *Cloud* (Figura 35).

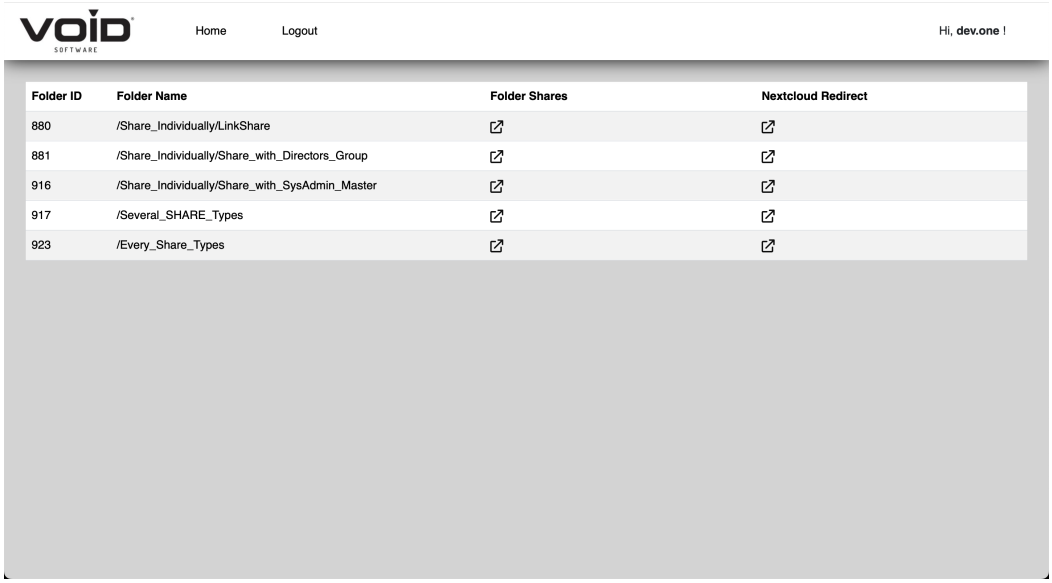

Figura 35: Pastas na *Cloud*

Esta tabela mostra:

- ***Folder ID***: identificador atribuído à pasta pelo serviço da *Nextcloud*.
- ***Folder Name***: nome da pasta
- ***Folder Shares***: ícone que redireciona o utilizador para uma outra vista que mostra todas as partilhas relacionadas com a respetiva pasta.

- ***Nextcloud Redirect***: botão que redireciona o utilizador diretamente para a pasta selecionada do serviço *Nextcloud*

4. ***Shares* por Pasta**

   Esta vista mostra ao utilizador os *Shares* que a pasta selecionada tem.

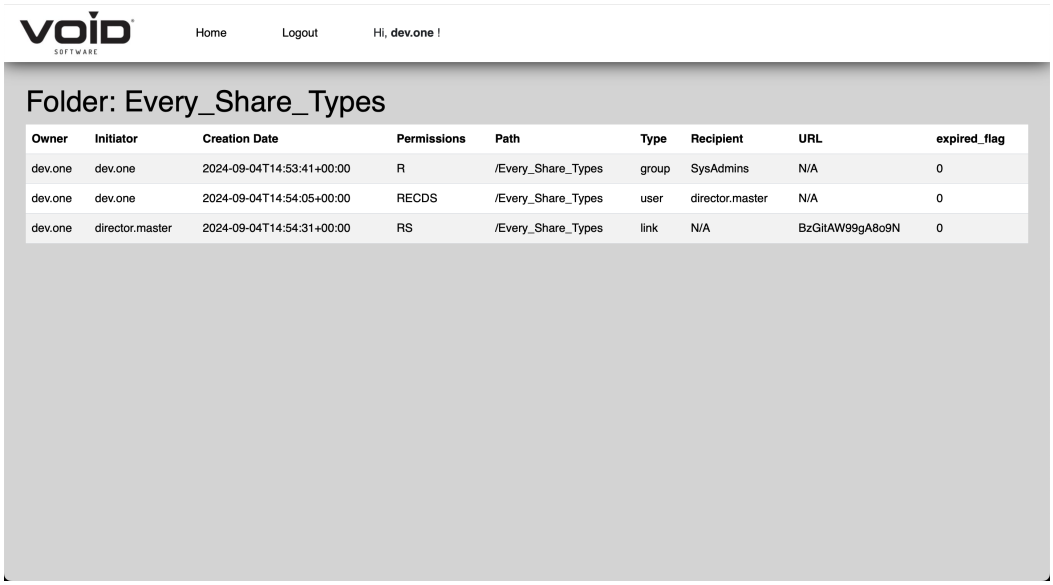

Figura 36: *Shares* na pasta selecionada

- ***Owner***: o dono da pasta.
- ***Initiator***: o nome de utilizador de quem iniciou o *Share*.
- ***Creation Date***: o *timestamp* que indica a data e hora em que o *Share* foi criado.
- **Permissions**: número inteiro que representa o tipo de permissão fornecida.

  A Tabela 10 apresenta como as permissões do serviço de *cloud* utilizado são geridas.

Read (1)	Edit (2)	Create (4)	Delete (8)	Share (16)	Total
1	0	0	0	0	1
1	1	0	0	0	3
1	0	1	0	0	5
1	1	1	0	0	7
1	0	0	1	0	9
1	1	0	1	0	11
1	0	1	1	0	13
1	1	1	1	0	15
1	0	0	0	1	17
1	1	0	0	1	19
1	0	1	0	1	21
1	1	1	0	1	23
1	0	0	1	1	25
1	1	0	1	1	27
1	0	1	1	1	29
1	1	1	1	1	31

Tabela 10: Tabela de Permissões em Binário

Os valores "0" e "1" nas colunas indicam, respetivamente, a ausência ou a presença da permissão correspondente. A coluna **Total** representa a soma dos valores binários das permissões ativas.

O valor apresentado na coluna **Total** corresponde ao código exibido no campo *permissions* do objeto. Para facilitar a análise, os códigos foram convertidos para mostrar apenas as iniciais de cada permissão.

- **Path**: caminho onde a diretoria/ficheiro partilhado se encontra.
- **Type**: Campo que define o tipo de *Share*. Estes valores podem ser:
  a) **User**: Quando o *Share* é criado para um utilizador individual
  b) **Group**: Quando um *Share* é partilhado com um Grupo de utilizadores
  c) **Link**: É gerado um URL onde qualquer pessoa que acede consegue visualizar o conteúdo partilhado. Este *Share* não é partilhado para nenhuma conta em específico.
- **Recipient**: Campo que mostra o nome do utilizador ou o grupo que o *Share* é partilhado.
- ***URL***: no caso do *Share* ser do tipo *link*, indica o *link* pelo qual a pasta/ficheiro pode ser acedido. Caso contrário, o valor é "*N/A*".

- ***expired_flag***: valor que indica se já passaram 30 dias desde que o *Share* foi criado. O valor pode ser 0 ou 1, indicando, respetivamente se já foram ultrapassados os 30 dias ou não.

5. ***Logout***: quando é realizado o *logout*, aparece novamente uma página do *Keycloak*:

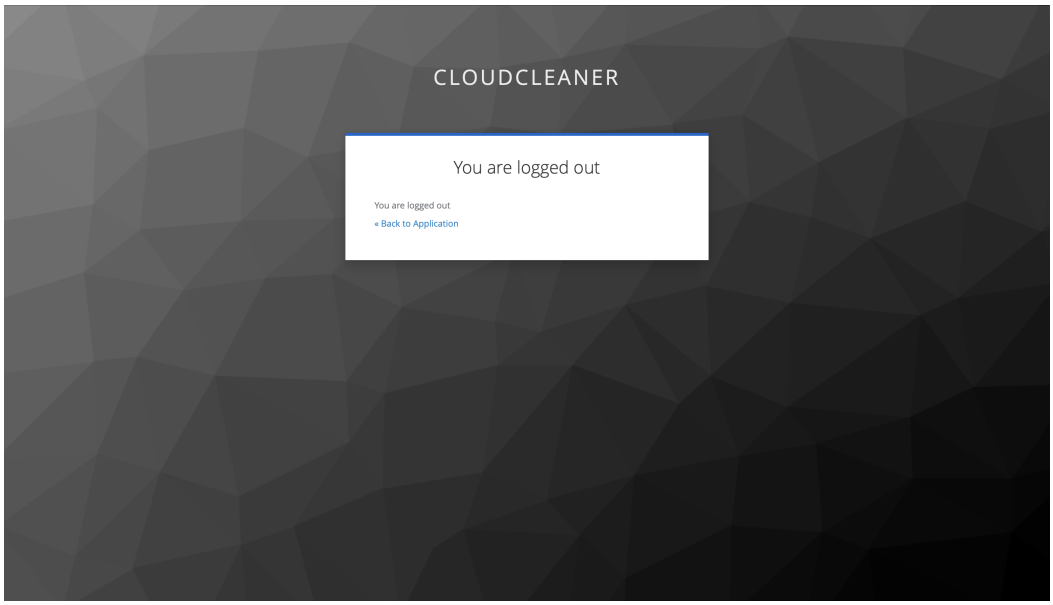

Figura 37: Página Inicial - sem *login* efetuado

Clicando em "*« Back to Application*", o utilizador é redirecionado para a vista mostrada anteriormente na Figura 34.

Quando é um *Admin* a fazer *login*, este consegue visualizar mais informação e realizar outro tipo de ações (Figura 38).

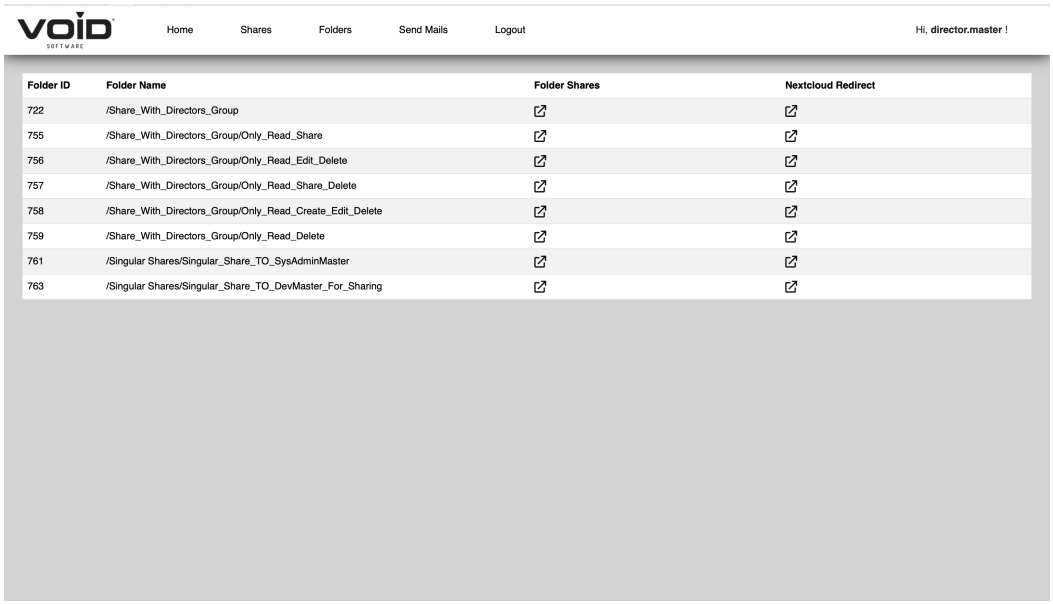

Figura 38: Página Inicial do *Admin*

Como se pode observar, ao contrário da página inicial de um utilizador normal, o *admin*, além de poder visualizar as vistas mencionadas anteriormente relacionadas com o seu conteúdo pessoal, tem acesso a diferentes menus na barra superior da página inicial.

1. ***Shares***: ao clicar na botão "*Shares*" este é presenteado com todos os *Shares* do sistema ordenados por ordem de criação.

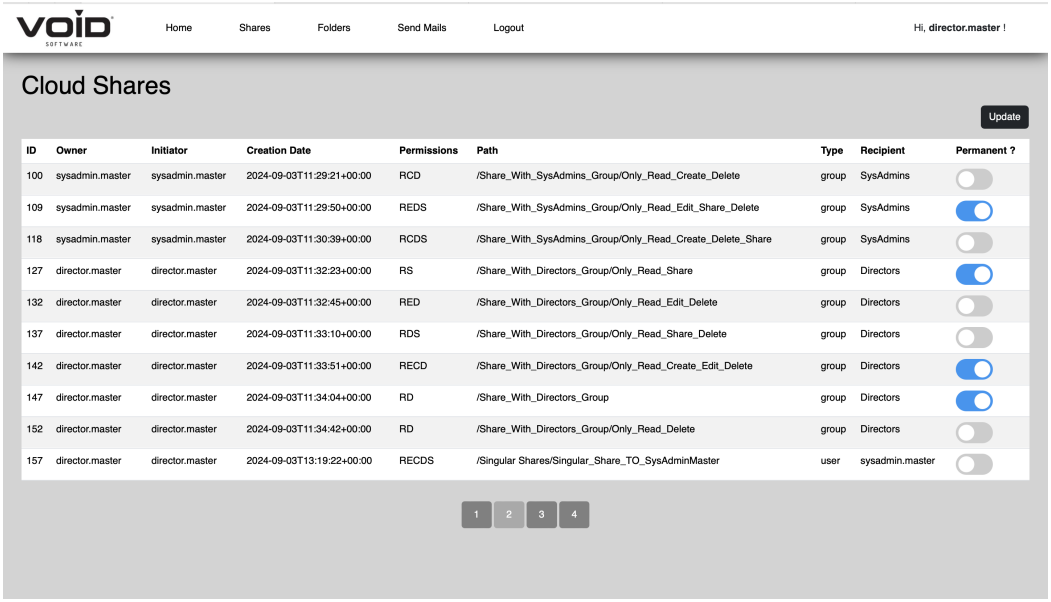

Figura 39: Todos os *Shares* do *Nextcloud*

2. ***Folders***: de forma semelhante, mas a informação apresentada é relativa a todas as Pastas do sistema.

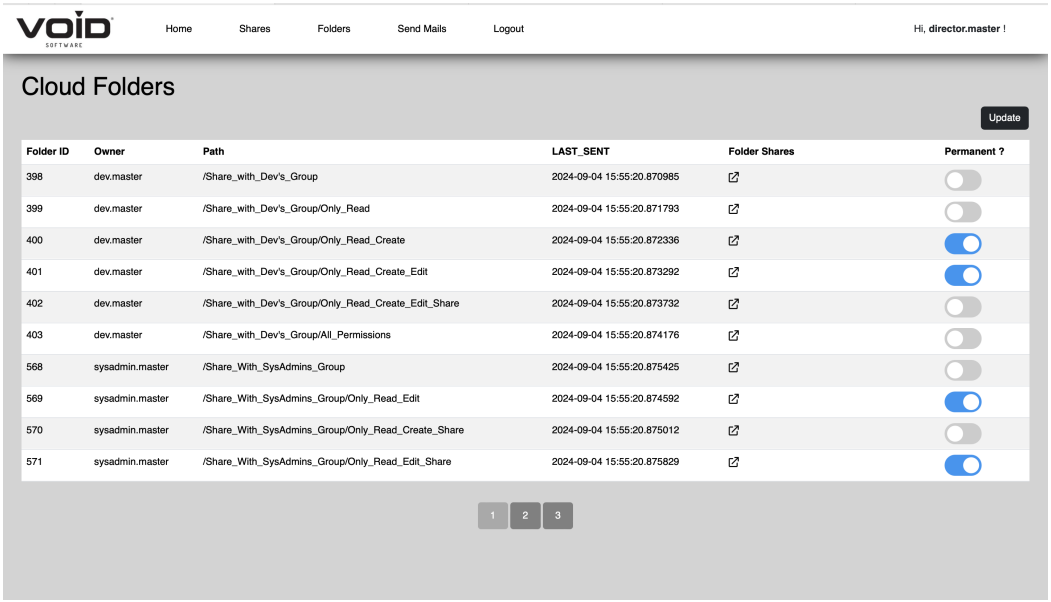

Figura 40: Todos as pastas do *Nextcloud*

3. ***Send Mails***: *endpoint* da aplicação que vai enviar os e-mails para cada dono das pastas. O conteúdo dos *e-mails*, como referido na secção 5.1 possui a estrutura da Figura 41:

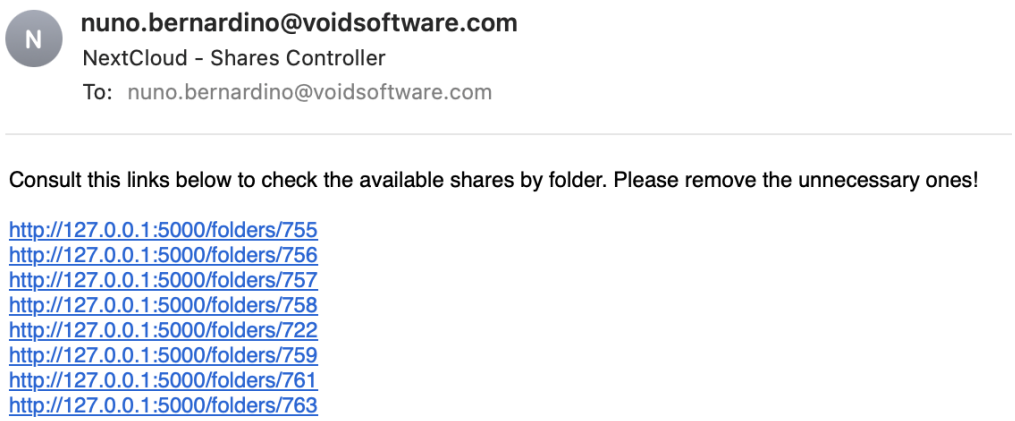

Figura 41: Estrutura do *email*

Importante reforçar, que cada utilizador normal (sem ser *admin*) só vai visualizar o conteúdo referente às suas pastas da *Cloud*.

A ferramenta desenvolvida melhora a segurança na organização ao proporcionar visibilidade e controlo sobre as partilhas de documentos e pastas no serviço de *Cloud*. Esta permite que os utilizadores vejam todas as partilhas ativas, ajudando a identificar acessos desnecessários ou desatualizados.

Com notificações semanais através do *e-mail* para revisões de partilhas expiradas (com mais de 30 dias) que não são permanentes, a ferramenta incentiva os colaboradores a avaliar continuamente a necessidade desses acessos, promovendo responsabilidade e reduzindo riscos de acessos indevidos. Isto, vai minimizar a exposição de dados sensíveis e fortalece a postura de segurança da organização ao manter apenas os acessos necessários ativos.

Em suma, a ferramenta *Cloud Cleaner* desempenha um papel essencial na melhoria da segurança da organização, ao oferecer uma plataforma robusta para a gestão de partilhas de documentos e pastas na *Cloud*. Ao proporcionar visibilidade, controlo e notificações periódicas para revisão de acessos, a ferramenta não só reduz a exposição de dados sensíveis, como também promove a responsabilidade dos colaboradores quanto à gestão de acessos. Com isso, a organização adota uma postura mais proativa e rigorosa em relação à segurança da informação.

6

CONCLUSÃO E TRABALHO FUTURO

As ferramentas desenvolvidas demonstraram ser ativos de valor inestimável para a organização, contribuindo significativamente para a melhoria do fluxo de trabalho e da produtividade. A sua funcionalidade, facilidade de utilização e adaptabilidade foram essenciais nesse processo. Com base nas descobertas apresentadas no capítulo 2 - Trabalho Relacionado -, e em resposta aos desafios e objetivos propostos no artigo (secção 2.1), bem como nas contribuições deste trabalho, retiram-se as seguintes conclusões:

- A ferramenta *BIC* responde parcialmente aos desafios D1, D2 e D3, ao garantir que os resultados de uma análise de segurança manual são automaticamente integrados numa ferramenta já existente, o *Jira*. Esta ferramenta é amplamente utilizada por programadores, contribuindo assim para o objetivo de desenvolver ferramentas de segurança centradas no programador (O1) e representando um primeiro passo na implementação da segurança das aplicações como um serviço (O2).
- A ferramenta *Version Checker* aborda o desafio D3, ao melhorar a avaliação de dependências, proporcionando resultados mais rápidos e alinhados com o ritmo acelerado de atualização das mesmas (normalmente atualizadas por razões de segurança). Esta ferramenta contribui diretamente para o objetivo O3 - Descoberta contínua de vulnerabilidades e gestão de dependências.
- A ferramenta *Cloud Cleaner* trata dos desafios D2 e D3, ao promover a descoberta rápida e interna de possíveis vulnerabilidades, que podem ser facilmente mitigadas. Com esta ferramenta, foram atingidos os objetivos O2 e O3, ao oferecer mais um serviço de testes de segurança e promover a descoberta contínua de vulnerabilidades e práticas de gestão.

Fazendo uma análise individual e concluindo o que foi descrito na secção 2.2:

No que diz respeito à ferramenta *BIC*, esta revelou-se uma solução eficiente e acessível para automatizar a criação de *issues* no *Jira* a partir de ficheiros *Excel* (com a estrutura do *OWASP ASVS*). Esta ferramenta destacou-se como uma alternativa sólida quando comparada a outras ferramentas, como o *Jira Automation*, que não permite a criação direta de *issues* a partir de ficheiros locais. Além disso, plataformas especializadas como o *OWASP DefectDojo* e o *Snyk* são mais complexas e difíceis de gerir, tornando a ferramenta *BIC* uma opção muito mais atraente.

Por sua vez, a ferramenta *Version Checker* destacou-se pela sua eficácia e facilidade de configuração na monitorização de versões de software. É uma solução prática para empresas que precisam garantir que as versões dos seus serviços ou sistemas estão sempre atualizadas. Ao contrário de soluções mais robustas e complexas, o *Version Checker* foca-se exclusivamente na verificação de versões de software, proporcionando uma abordagem leve e acessível, sem exigir configurações complicadas ou conhecimento técnico aprofundado. Assim, posiciona-se como uma alternativa ágil e eficaz para empresas que desejam assegurar a integridade das versões dos seus sistemas, sem os custos de complexidade adicional.

Em suma, ao analisar as diversas ferramentas estudadas na secção 2.2, tanto a BIC como o *Version Checker* ofereceram soluções que preenchem lacunas específicas nos processos de gestão e automação de tarefas empresariais. A simplicidade e a acessibilidade são características que diferenciam estas soluções das demais analisadas.

No que se refere à ferramenta *Cloud Cleaner*, foi evidente a escassez de trabalhos disponíveis, o que está certamente relacionado com a sua natureza muito objetiva e específica, explicando também a falta de artigos sobre o tema. No entanto considero que se alinha com as melhores práticas de segurança da informação o que acaba por promover um ambiente digital mais controlado e eficiente.

Relativamente a trabalho futuro, nas ferramentas desenvolvidas podem ser acrescentadas/alteradas certas funcionalidades.

1. **Na ferramenta BIC**:
    - Quando as *issues* depois de resolvidas forem movidas para coluna "Done", caso estes sejam removidos (apagados) da *Board* do projeto, a ferramenta torna a reportá-las, caso não seja alterado o valor da coluna *"Valid"* para "Valid" no *Excel*.
    - Em vez dos *issues* serem criados e só depois o valor do campo "Priority" ser atualizado consoante o tipo de *issue*, este pode ser logo criado com este valor definido.
    - Seria também interessante a ferramenta conseguir usar outros outros *templates* de *Excel* para além do OWASP ASVS.
    - O *Report* que é gerado com um resumo das *issues* criadas pode ser enviado para algum canal de comunicação (*e-mail*, serviço de *chat*, entre outros...).
2. **Na ferramenta *Version Checker***:
    - A escolha do endereço de *e-mail* pode ser realizada de outra maneira. De forma a conseguir escolher inúmeros endereços diferentes.

- A informação pode ser enviada de outra maneira sem ser o envio de um e-mail.

3. **Na ferramenta *Cloud Cleaner***:

   - Pode ser criada alguma *view* apenas disponível para o administrador que consiga ver dados estatísticos. Como por exemplo:
     - Que utilizador no último criou mais *shares* "externos".
     - Que utilizador teve mais *shares* expirados

BIBLIOGRAFIA

[1] *What Is a Data Leak? Causes & Prevention.* [Online; accessed 7. Feb. 2024]. Out. de 2023. URL: https://abnormalsecurity.com/glossary/data-leak.

[2] *VOID Software :: the software specialists.* [Online; accessed 29. Sep. 2023]. Mai. de 2023. URL: https://void.pt.

[3] *VOID Software Reviews | View Portfolios | DesignRush.* [Online; accessed 16. Oct. 2023]. Out. de 2023. URL: https://www.designrush.com/agency/profile/void-software.

[4] *0*rXGLIBdCrpI8F-6J.webp (WEBP Image, 1400 × 976 pixels) — Scaled (83%).* [Online; accessed 29. Jan. 2024]. Jan. de 2024. URL: https://miro.medium.com/v2/resize:fit:1400/format:webp/0*rXGLIBdCrpI8F-6J.

[5] 0xffccdd. «DevSecOps Definition, Best Practices and Tools - 0xffccdd - Medium». Em: *Medium* (jun. de 2023). URL: https://medium.com/@cloud_tips/devsecops-definition-best-practices-and-tools-1789587d165a.

[6] *The 11th IEEE International Conference on Cyber Security and Cloud Computing – IEEE CSCloud 2024.* [Online; accessed 27. Sep. 2024]. Abr. de 2024. URL: https://www.cloud-conf.net/cscloud/2024/cscloud/index.html.

[7] Nuno André Bernardino et al. «Enhancing DevSecOps: Three Custom Tools for Continuous Security». Em: *2024 IEEE 11th International Conference on Cyber Security and Cloud Computing (CSCloud).* IEEE, pp. 28–30. DOI: 10.1109/CSCloud62866.2024.00017.

[8] Michael Howard e Steve Lipner. *The security development lifecycle.* Vol. 8. Microsoft Press Redmond, 2006.

[9] Aleksandar Dimov e Vladimir Dimitrov. «Classification of software security tools». Em: *Information Systems and Grid Technologies* (2021).

[10] Mary Sánchez-Gordón e Ricardo Colomo-Palacios. «Security as culture: a systematic literature review of DevSecOps». Em: *Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops.* 2020, pp. 266–269.

[11] Thorsten Rangnau et al. «Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines». Em: *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC).* IEEE. 2020, pp. 145–154.

[12] Roshan N Rajapakse et al. «Challenges and solutions when adopting DevSecOps: A systematic review». Em: *Information and software technology* 141 (2022), p. 106700.

[13] *Atlassian Documentation*. [Online; accessed 29. Sep. 2024]. Set. de 2024. URL: https://confluence.atlassian.com/automation.

[14] *GitHub: Let's build from here*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://github.com.

[15] Atlassian. *Bitbucket*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://bitbucket.org/product.

[16] Slack. *AI Work Management & Productivity Tools*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://slack.com.

[17] *OWASP Defectdojo | OWASP Foundation*. [Online; accessed 29. Sep. 2024]. Set. de 2024. URL: https://owasp.org/www-project-defectdojo.

[18] *Getting started | Snyk User Docs*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://docs.snyk.io/getting-started.

[19] *Overview | Nagios Open Source*. [Online; accessed 28. Sep. 2024]. Set. de 2024. URL: https://www.nagios.org/about/overview.

[20] *Introduction to Ansible — Ansible Community Documentation*. [Online; accessed 28. Sep. 2024]. Set. de 2024. URL: https://docs.ansible.com/ansible/latest/getting_started/introduction.html.

[21] *What is MDM? A Guide to Mobile Device Management | Miradore*. [Online; accessed 28. Sep. 2024]. Jul. de 2024. URL: https://www.miradore.com/blog/mdm-mobile-device-management.

[22] *Google Académico*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://scholar.google.com.

[23] *IEEE Xplore*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://ieeexplore.ieee.org/Xplore/home.jsp.

[24] *ACM Digital Library*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://dl.acm.org.

[25] *Home | SpringerLink*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://link.springer.com.

[26] *BIC*. [Online; accessed 16. Jan. 2024]. Jan. de 2024. URL: https://github.com/nunobernas/BIC.

[27] Tes. «Implementing QA in a CI/CD Pipeline - Parasoft». Em: *Parasoft* (fev. de 2024). URL: https://www.parasoft.com/blog/implementing-qa-in-a-ci-cd-pipeline.

[28] *Jenkins*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://www.jenkins.io.

[29] *Get started with GitLab CI/CD | GitLab*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://docs.gitlab.com/ee/ci.

[30] *Simple, Flexible, Trustworthy CI/CD Tools - Travis CI*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://www.travis-ci.com.

[31] *Continuous Integration and Delivery*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://circleci.com.

[32] *Pipeline as Code*. [Online; accessed 12. Aug. 2024]. Ago. de 2024. URL: https://www.jenkins.io/doc/book/pipeline/pipeline-as-code//#the-jenkinsfile.

[33] *About the OWASP Foundation | OWASP Foundation*. [Online; accessed 2. Oct. 2023]. Out. de 2023. URL: https://owasp.org/about.

[34] *CWE - Common Weakness Enumeration*. [Online; accessed 9. Oct. 2023]. Out. de 2023. URL: https://cwe.mitre.org.

[35] *Special Publication 800-63 | NIST*. [Online; accessed 11. Oct. 2023]. Fev. de 2022. URL: https://www.nist.gov/special-publication-800-63.

[36] *OWASP Top Ten | OWASP Foundation*. [Online; accessed 10. Jan. 2024]. Jun. de 2023. URL: https://owasp.org/www-project-top-ten.

[37] *wordlists/wordlists/passwords at main · kkrypt0nn/wordlists*. [Online; accessed 15. Jan. 2024]. Jan. de 2024. URL: https://github.com/kkrypt0nn/wordlists/tree/main/wordlists/passwords.

[38] *asyncio — Asynchronous I/O*. [Online; accessed 1. Feb. 2024]. Fev. de 2024. URL: https://docs.python.org/3/library/asyncio.html.

[39] *threading — Thread-based parallelism*. [Online; accessed 1. Feb. 2024]. Fev. de 2024. URL: https://docs.python.org/3/library/threading.html.

[40] Keycloak Team. *Keycloak*. [Online; accessed 16. Sep. 2024]. Set. de 2024. URL: https://www.keycloak.org.

[41] *Nextcloud - Open source content collaboration platform*. [Online; accessed 16. Sep. 2024]. Set. de 2024. URL: https://nextcloud.com.

[42] The Nextcloud Community. *Share Listing - Apps - App Store - Nextcloud*. [Online; accessed 4. Sep. 2024]. Set. de 2024. URL: https://apps.nextcloud.com/apps/sharelisting.

[43] *Open Source Database (RDBMS) for the Enterprise | MariaDB*. [Online; accessed 16. Sep. 2024]. Set. de 2024. URL: https://mariadb.com.

[44] *Docker: Accelerated Container Application Development*. [Online; accessed 30. Sep. 2024]. Jul. de 2024. URL: https://www.docker.com.

[45] *nextcloud - Official Image | Docker Hub*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://hub.docker.com/_/nextcloud.

[46] *bitnami/keycloak - Docker Image | Docker Hub*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://hub.docker.com/r/bitnami/keycloak.

[47] *mariadb - Official Image | Docker Hub*. [Online; accessed 30. Sep. 2024]. Set. de 2024. URL: https://hub.docker.com/_/mariadb.

[48] The Postman Team. «What Is an API Endpoint? | Postman Blog». Em: *Postman Blog* (fev. de 2024). URL: https://blog.postman.com/what-is-an-api-endpoint.

[49] Vk. *AWUS1900*. [Online; accessed 15. Apr. 2024]. Abr. de 2024. URL: https://alfa-network.eu/awus1900.

[50] admin. «4-Way Handshake - WiFi». Em: *WiFi* (fev. de 2022). URL: https://www.wifi-professionals.com/2019/01/4-way-handshake.

[51] *Wireshark · About*. [Online; accessed 22. Sep. 2024]. Set. de 2024. URL: https://www.wireshark.org/about.html.

[52] hashcat. *hashcat*. [Online; accessed 22. Sep. 2024]. Set. de 2024. URL: https://github.com/hashcat/hashcat.

[53] *hashcat_utils [hashcat wiki]*. [Online; accessed 23. Apr. 2024]. Abr. de 2024. URL: https://hashcat.net/wiki/doku.php?id=hashcat_utils#cap2hccapx.

APÊNDICES

A

ENUNCIADO DE ESTÁGIO

Este apêndice apresenta o enunciado inicial da proposta de estágio apresentado pela empresa *VOID SOFTWARE S.A.*

Título: Avaliação de segurança de aplicações e infraestrutura

Departamento: DevSecOps

Criador(es): Bernardo Sequeira, Fábio Antunes, Eduardo Piza

Supervisor(es): Fábio Antunes, Eduardo Piza

Referência: INTERNSHIP.P3.2023

Contexto

No cenário digital de hoje, a segurança é uma preocupação primordial. Duas questões prementes que exigem atenção imediata são a segurança da infraestrutura e o número crescente de explorações e ataques direcionados a aplicações [1]. Proteger a infraestrutura e as aplicações tornou-se imperativo à medida que as organizações se esforçam para proteger dados confidenciais, manter a integridade operacional e preservar a confiança dos utilizadores. “Uma avaliação de segurança de aplicações é uma parte crucial do ciclo de vida de desenvolvimento de software das empresas.” [2]

Ao nível da infraestrutura, as vulnerabilidades podem expor sistemas sujeitos a violações e acesso não autorizado. O teste de intrusão (pentest) desempenha um papel vital na identificação e tratamento dessas vulnerabilidades [3]. Ao simular ataques do mundo real, o teste de intrusão avalia a resiliência dos componentes da infraestrutura, ajudando as organizações a prioritizar os esforços de remediação e fortalecer os seus sistemas, contra possíveis ameaças. Enquanto isso, medidas proativas de segurança são essenciais para proteger as aplicações contra *exploits* e ataques. Práticas robustas de segurança de aplicações, como codificação segura e avaliações regulares de vulnerabilidade, reduzem os riscos e garantem a integridade de aplicações [4].

Objetivo

Este estágio foca-se na avaliação de segurança de aplicações e infraestrutura. Nomeadamente, visa realizar pentesting à infraestrutura de uma rede (interna e externamente) e, ainda, familiarizar-se com ferramentas conhecidas como Wazuh, MobSF e WSAP. O processo de reportar vulnerabilidades de segurança para a equipa de desenvolvimento e integrar esta etapa no SDLC deve ser concebido e validado.

Atividades

1. Visão geral das tecnologias e ferramentas utilizadas (Pentesting, Wazuh, MobSF e WSAP)
2. Visão geral do processo de desenvolvimento e padrões de segurança em prática dentro da empresa
3. Pentest na infraestrutura (interna e externa)
4. Configuração de ferramentas (Wazuh, MobSF e WSAP)
5. Conceção do processo de reporting de problemas de vulnerabilidade e integração no SDLC

Referências

[1] Lew, P., Olsina, L., & Zhang, L. (2010). Quality, quality in use, actual usability and user experience as key drivers for web application evaluation. In Web Engineering: 10th International Conference, ICWE 2010, Vienna Austria, July 5-9, 2010. Proceedings 10 (pp. 218-232). Springer Berlin Heidelberg.

[2] React. (2023). React. https://react.dev/ [Available online: accessed June 19 2023]

[3] JavaScript With Syntax For Types. (2023). TypeScript: JavaScript With Syntax for Types. https://www.typescriptlang.org [Available online: accessed June 19 2023]

[4] Documentation - React. (2023). TypeScript: Documentation - React. https://www.typescriptlang.org/docs/handbook/react.html [Available online: accessed June 19 2023]

[5] Malhotra, R., & Chug, A. (2016, March). Comparative analysis of agile methods and iterative enhancement model in assessment of software maintenance. In 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1271-1276). IEEE.

Versão do Documento

V0.9 - Draft - June 2023 - Author: Bernardo Sequeira - VOID Labs - VOID Software

B

ENHANCING DEVSECOPS: THREE CUSTOM TOOLS FOR CONTINUOUS SECURITY

Este apêndice apresenta o artigo publicado na 11ª edição da conferência "*2024 IEEE International Conference on Cyber Security and Cloud Computing (CSCloud).*

Enhancing DevSecOps: Three Custom Tools for Continuous Security

Nuno André Bernardino
VOID Labs - VOID Software
ESTG - Polytechnic of Leiria
Leiria, Portugal
nuno.bernardino@voidsoftware.com

Bernardo Sequeira
VOID Labs - VOID Software
Leiria, Portugal
bernardo.sequeira@voidsoftware.com

Eduardo Piza
VOID Labs - VOID Software
Leiria, Portugal
eduardo.piza@voidsoftware.com

Fábio Henriques
VOID Labs - VOID Software
Leiria, Portugal
fabio.henriques@voidsoftware.com

Filipe Neves
ESTG - Polytechnic of Leiria
Leiria, Portugal
fneves@ipleiria.pt

Catarina I. Reis
VOID Labs - VOID Software
Leiria, Portugal
catarina.reis@voidsoftware.com

***Abstract*—With the rapid evolution of technology and the adoption of agile methodologies like DevOps, the integration of security, known as DevSecOps, becomes imperative to ensure software integrity and safety. This paradigm shift emphasizes the incorporation of security testing throughout the development lifecycle, promoting Continuous Security Testing (CST) rather than deferring it to later stages. In response to this need for integrated security in software development, we propose three custom tools designed for specific stages of the development cycle. The first tool generates JIRA issues for identified vulnerabilities during the "Build" and "Test" phases. The second ensures alerts for service updates for infrastructure security in the "Monitor" phase. Lastly, the third tool identifies unused Cloud Service files and folders for monitoring in "Operate" and "Monitor" stages. By incorporating these tools, organizations can bolster security practices and maintain software integrity throughout development.**

***Index Terms*—DevSecOps, security, Continuous Security Testing, automation, vulnerabilities**

I. Introduction

DevSecOps is a combination of development, security and operations. A DevSecOps team is multidisciplinary and is responsible for ensuring security through practices and decisions at the same level as development and operation tasks [1].

Thus, DevSecOps integrates security practices into all stages of the Software Development Life Cycle (SDLC). Instead of being treated as a separate and dedicated stage, that only happens after application development, DevSecOps seeks to incorporate these practices, early from the very beginning of an application's development up until its deployment.

The tools developed and presented in this document are applied precisely in this context. They are custom-made software tools that are: (a) used to ensure that security procedures are applied to staged applications before they go into production, and (b) specific for managing and monitoring the organization's internal infrastructure.

This research work was supported by *VOID SOFTWARE, S.A.*, to whom the authors gratefully acknowledge the generous support.

The document is organized as follows. Chapter II presents the state-of-the-art, providing an overview of the basic concepts related to the CI/CD pipeline and the DevSecOps toolchain. It also includes a section that highlights current research related to this study. Chapters III-V will present the tools developed. Each tool's usage is described, a section ("Setup") describes the tool's requirements regarding the environment, and finally, a section ("Usage and Execution Results") provides details on how to use the tools and presents the results of their execution. The last chapter (VI) concludes the paper and presents future work.

II. State of the Art

Traditional software development methodologies are not enough to fulfill nowadays business requirements. Adaptation of Agile practices enables flexibility, efficiency and speed of the SDLC, which is attractive to software development companies [2]. The *Agile Manifesto* [3] sets out the twelve fundamental principles of agile software development. By combining Continuous Integration (CI) and Continuous Delivery (CD) (later becoming the CI/CD pipeline) software delivery is much more efficient and productive in agile processes.

A CI/CD pipeline is a software automation approach that allows developers to deliver high-quality code more efficiently and quickly. It consists of a series of automated steps that the code goes through from the moment it is developed until it is deployed in production [4]. The automation provided by a CI/CD pipeline not only increases the efficiency of the development process, but also helps to reduce human errors, improves code quality and enables more frequent and reliable software releases. Several tools compile and run these pipelines automatically (*e.g.* Jenkins, CircleCI, Bamboo). With continuous deployment, changes to the code - continuous integration - are automatically promoted to production as soon as they succeed in the following pre-defined stages of the pipeline (namely a mandatory Testing stage).

The aim is to further reduce time between writing the code and making it available to end users/customers, allowing for fast and frequent deployments of new features and bug fixes.

With the advance of time and the emergence of new technologies, new software development methodologies such as DevOps, which promotes speed and agility, need to incorporate security testing during this rapid development cycle sooner rather than later [5], thus achieving Continuous Security Testing (CST). This approach, also known as 'Shift left,' moves testing to earlier in the software development lifecycle.

DevSecOps activities can be framed in a infinity tool-chain as the one presented in Figure 1.

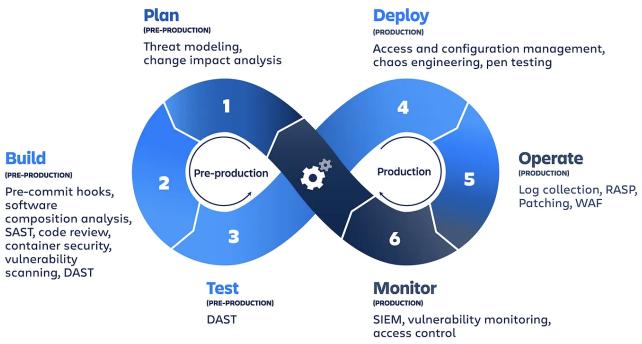

Fig. 1. DevSecOps Tool-chain [5].

A. Related Work

Enforcing security along the SDLC is crucial for maintaining the integrity and safety of software applications. New threats appear everyday and they evolve with technology. Microsoft, as early as 2006, addressed the need to ensure security since "everything is connected" and introduced its Security Development Lifecycle [6]. Since then, various tools and practices have been introduced and used throughout the SDLC to ensure security at different stages of development. Static Application Security Testing (SAST) are used to identify security vulnerabilities within source code. Dynamic Application Security Testing (DAST) test running applications in real-time. Interactive Application Security Testing (IAST) combine SAST with DAST. Software Composition Analysis (SCA) identify and track dependency-related vulnerabilities. These are the most widely known and used categories amongst other categories and typologies [7].

However, and despite the popularity and perceived benefits, software security aspects of DevOps remain a concern for most organizations. To deal with some security issues, culture is an essential element that needs to be adequately addressed in DevSecOps. DevSecOps culture is an emerging topic that goes beyond moving some security practices to an earlier phase of the software lifecycle. Recent reviews of the literature show that there are several needs to be addressed and there is a long pathway to reach the ultimate level of Continuous Security Testing [8]–[10].

Specifically, [10] identified several challenges that need to be addressed and future directions or goals to achieve. The following short-list presents the most relevant challenges we considered as high priority to tackle in our work:

- C1: Security issues resulting from tool complexity and integration challenges
- C2: Inability to fully automate traditionally manual security practices to integrate into DevSecOps security practices to integrate into DevSecOps
- C3: Inability to carry out rapid security requirements assessment.

And we highlight the following goals:

- G1: The need for security tools that target developers and not security experts (*e.g.* developer-centered security)
- G2: Application security testing as a service
- G3: Continuous vulnerability discovery and management practices.

Based on the revision of the state of the art and the related work, and as far as we were able to uncover, no tooling could be found that provided an adequate answer to the raised issues. We propose 3 new custom-made tools, identified for specific stages of the DevSecOps cycle, namely the Build, Test, Operate and Monitor phases. They aim to address specific security concerns and internal practices, namely:

- a tool that automates the creation of issues in a JIRA project that were identified as security vulnerabilities through the manual execution of a checklist [11], [12]. This tool sits within stages 2 and 3 ("Build" and "Test");
- a tool that verifies if services running within an organisation are up to date according to their latest stable version. The purpose is to monitor the organization's infrastructure, addressing possible sources of attacks. It will fit on stage 6 ("Monitor");
- and, a tool that lists all the "unused" shared folders/files in the Cloud Service of a company's infrastructure. It will monitor the sharing of unnecessary information, and can be included in steps 5 and 6 ("Operate" and "Monitor").

III. Bulk Issue Creator (BIC)

One of the many tasks of a Security Analyst is to analyze the code and detect poor implementation of security controls.

There are several ways to analyze the correct implementation of these security controls. There is an automatic approach in which applications are used that provide final reports on the overall security posture of an application. These reports point to security flaws and present possible ways to exploit the application, as well as the solution that should be implemented to mitigate them.

The OWASP-ASVS is a checklist that is handled by a security expert that exercises each entry manually. He follows a checklist that can be found in different spreadsheet formats (*e.g.* Excel) and contains best practices/standards that an application should follow. The Security Analyst should check (manually) those that are validated and those that are not.

A common procedure that follows the above mentioned manual validation is to register new JIRA issues in the corresponding JIRA board's project, one per each vulnerability detected. After obtaining the results it is necessary to go to JIRA and enter each issue manually, a process that is time-consuming, repetitive and a potential source of errors (human factor).

It was in this context that the BIC tool was developed. Now, Security Analysts can report these issues to JIRA in a simpler and more automatic way. After each execution of the tool, a report is generated (.txt file) where it is possible to see a listing of the issues that were created on the JIRA board. This tool and technical manual are available here [13].

The architecture of the tool is presented on Figure 2.

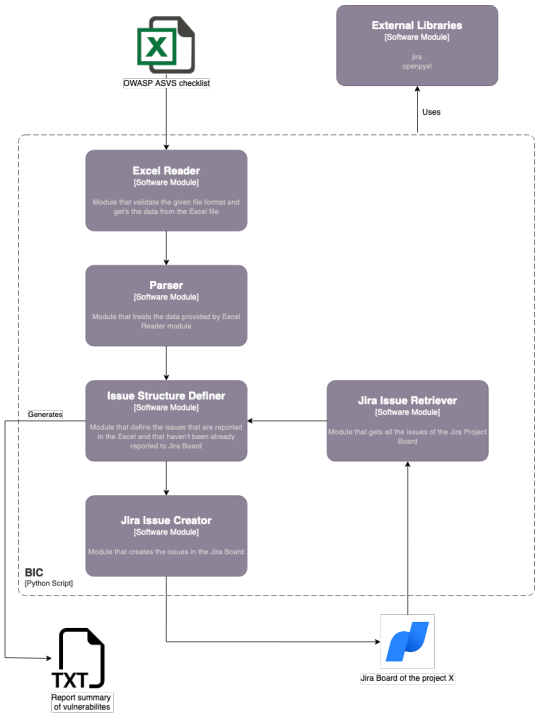

Fig. 2. BIC Architecture

A. Setup

The first step in the functioning of the tool corresponds to the creation of a new spreadsheet structure that is very similar to the original (Figure 3).

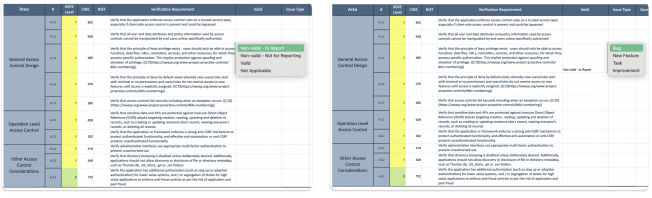

Fig. 3. Updated spreadsheet structure

The new "Valid" column has the following values {Valid, Not Applicable, Non-valid - to Report, Non-valid - not for Reporting}. This allows the Security Analyst to distinguish between security controls that are not validated, and those that should or should not be reported. Only the issues signalled with "Non-valid - to Report" will be reported.

On the other hand, a new "Issue Type" column was created, allowing the user to define the type of issue that will be reported to JIRA. Possible values include: Bug, New Feature, Task (default), and Improvement. Regarding the JIRA project that will serve as destination, it is required to ensure that the referenced issue types are correctly configured.

B. Usage and Execution Results

The correct use of the tool is summarized in the following three steps:

- Step 1 - Create updated spreadsheet (run BIC.py with the original OWASP-ASVS excel).
- Step 2 - Manual validation of vulnerabilities (fill in the updated OWASP-ASVS checklist).
- Step 3 - Create JIRA issues (run BIC.py with the updated and filled in OWASP-ASVS checklist).

Figure 4 illustrates the JIRA Board after the tool execution. The summary and description of the issues were reused from the best practices of the original checklist (eg. "OWASP-ASVS #4.1.4").

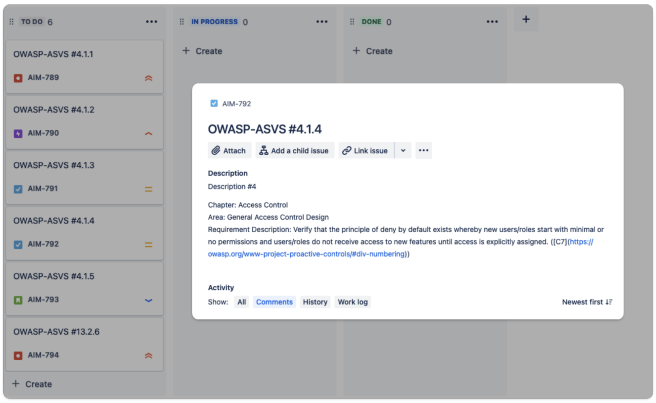

Fig. 4. JIRA Board with vulnerability issues - automatically generated

IV. Version Checker

A company should ensure that all of its internal services are up to date in order to avoid leaving any openings for potential attackers to exploit. These days, this is a relevant concern that should be tackled. On the list of the top ten dangers to application security [14], "Vulnerable and outdated components" is so widely used for attacks, that it comes in at number six.

This is why "Version Checker" was created. This tool's objective is to determine whether the services that are being utilized in an organization are current, regarding the most recent stable versions publicly available.

Figure 5 shows the architecture of this tool. The user runs the tool and is presented with a prompt to choose the email address he wishes to send a version status report to. Another component searches for the most recent stable versions for each service and, finally, a list with the status for each service is created and sent by email.

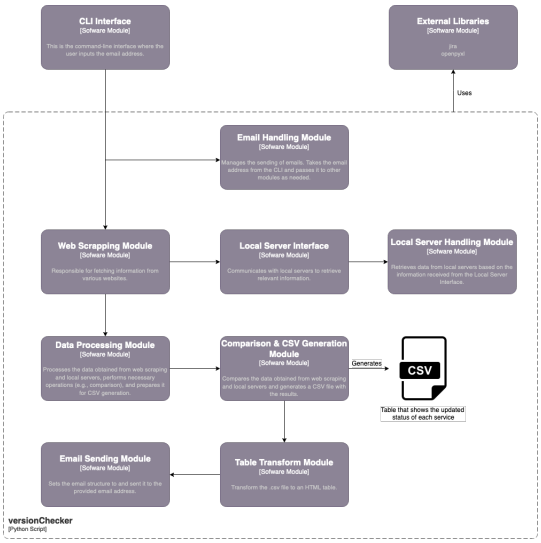

Fig. 5. Version Checker Architecture

In order to obtain the latest available version of each service, an information extraction technique/process called ”Web Scrapping” was used. Using the HTML code of each original service website/repository, it was possible to retrieve the latest stable version of each service.

To obtain the versions of existing services operating on the servers, SSH connections are established, and many commands are issued to obtain a list of services and their related versions (current version).

Version Checker searches several different websites. It is noticeable that its execution time is slightly long due to the large number of webpages that it extracts information from. We have optimized the process using different techniques. Table I shows the results for an average of ten runs of these different approaches. We have opted for the implementation of threads to aim for the most efficient approach.

TABLE I
EXECUTION TIME COMPARISON

Method	Average Time (10 runs)
source-code	45.814 s
async.io	47.469 s
threads	4.216 s

Upon execution, results are presented in a table format (Table II). This table’s main column refers to the status. This column is color-coded to show if the service is up to date. The approach employed was a function that checks both the current version and the latest stable version available; if they match, the service is up to date; otherwise, action should be taken to update it.

The values in this column can be:

1) 🟢: Updated service!
2) 🟡: It was not possible to collect the information directly from the source. Version taken from the last run of Version Checker.
3) 🔴: Service out of date! A newer version is available!

When the value of the ”Status” is yellow, it is because it wasn’t possible to get the information from the corresponding site. Whenever a site is down or the structure of the page has changed, values to update the ”Status” are taken from the (csv) file resulting from the last execution of this tool, if it exists. Otherwise, it prints the string ”???”. Therefore, the color yellow informs the tool user that something is unusual with the websites and that further steps to correct the situation should be taken (eg. refactoring the web scrapping component to read through the new site structure).

A. Setup

Version Checker requires that: (1) an email list is provided, and (2), there is an active private key to access to the servers that are hosting the services ((1) and (2) are both command-line arguments).

The email list is a text file that contains all the email addresses of those interested in receiving the results provided by the tool e.g. the Computer Security Incident Response Team - CSIRT). The file must have one email address per line.

The private key received as an argument is later used in the process when Version Checker establishes SSH connections to each server to discover the services current versions.

On a side note, this tool is custom-made and specific to a set of predefined services. However, it is possible to have a variant and adapt it to other environments. First, it is necessary to list all the services that the organization wants to monitor. Then it is necessary to have two distinct functions per service: (1) a function to retrieve the latest version of the service available, and, (2) a function that accesses the servers to retrieve the current version.

B. Usage and Execution Results

The user simply has to run the tool (version_checker.py).

Version Checker outputs the following information in a table format (see Table II):

- **ID**: unique identifier of the service.
- **Service**: name of the service.
- **Type**: specifically identifies the service, if there is more than one instance available (*e.g.* the internal or external chat service).
- **Current Version**: version of the service installed.
- **Latest Version**: latest stable version of the corresponding service (information retrieved online).
- **Status**: based on the values of the current version *versus* the latest version available.

- **LATEST_UPDATED_AT**: the most recent date from the official website of each service corresponding to the latest version.

V. Cloud Cleaner

In an increasingly digitalized and interconnected business environment, information shared internally (between employees) and externally (with customers) becomes essential to evolve collaboratively. In order to facilitate this sharing of information, organizations have implemented instances of cloud services. Sensitive and confidential information stored in the cloud is subject to internal and external threats, such as data leaks, cyberattacks or misuse by malicious employees.

Organizations implement restricted access policies and permissions based on profiles, teams, roles and access control lists (ACLs). Thus, they can significantly reduce the risk of exposure and protect their most important assets (the shared information).

Control of shared folders is essential to preserve data integrity and confidentiality. By monitoring these folders and who has access to them, organizations can ensure that only the necessary/supposed information is visible and shared.

Cloud Cleaner's main purpose is to list all the shares that exist within each instance of the cloud services so that it is easier to control these shares.

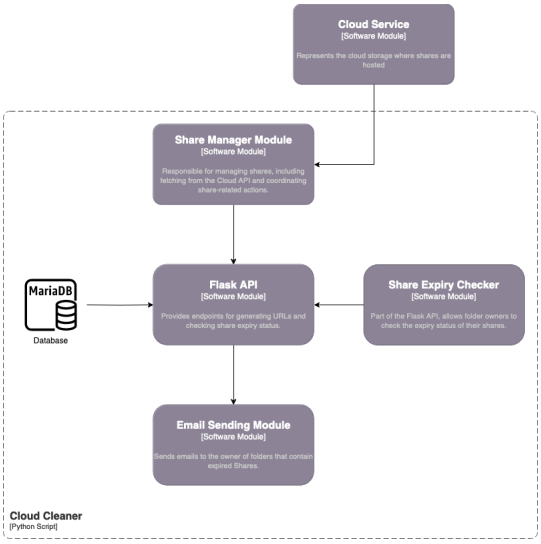

Fig. 6. Cloud Cleaner Architecture

This tool is designed so that the owner of each folder with "Shares" that exist for more than 30 days receives an e-mail message with one link per folder. The goal is to make each user accountable for his own actions. Thus, each cloud user will receive one e-mail with ***n*** links, one per each folder with "expired" shares.

The link corresponds to the folder's URL and will allow the owner of the folder to check the "expired" Shares.

E-mails are always sent every 30 days. To ensure this, a database was used (MariaDB) to store the Shares and also the timestamp of the last email sent ("LAST_SENT" field).

A. Setup

This tool was developed for a specific cloud service. It can easily be used as an administrative plugin to check all the unnecessary shares. Installing the tool directly on the server's hosting the cloud services is the first prerequisite that has to be set up.

It is necessary to have a database (as referred before). The tool will detect its existence and upon its first usage will create the required table structure. For each execution of the tool, the database will only be updated (append mode only).

B. Usage and Execution Results

The user should start the tool by running **cloud_cleaner.py**.

A link then appears on the terminal, taking the user directly to the platform's homepage. For admins, a second URL is provided, where they see all the "Expired Shares".

Results are presented on a browser and are distinct for both user and admin profiles. They share the same structure. Table III illustrates the format of the information supplied. For each URL that will be supplied via e-mail to the owner of the share, it will link to the "Expired Shares" detailed information and just on a specific folder. The Admin profile will have access to all the information of the "Expired Shares".

- **Owner**: owner of the folder.
- **Folder Path**: the path of the folder in the owner's quota.
- **Permissions**: type of permissions per share: **(R)**ead; **(C)**reate; **(E)**dit; **(S)**hare; **(D)**elete.
- **Expiration Date**: the expiration date of the Share. If not specified, it will be "None".
- **Type**: (U)ser - individual account. (G)roup - group (Administrators, Developers, RH, ...) with several accounts. (L)ink - shared through a link with people without accounts ("External Share").
- **Shared Users**: name of the holder of the share (User, Group or None, in case of a link).

VI. Conclusions and Future Work

The tools developed have proven to be invaluable assets to the organization.

Their functionality, ease of use, and adaptability have enhanced our workflow and productivity. Expanding upon the discoveries delineated in the related work section, particularly those outlined by the article [10], alongside our own contributions, we draw the subsequent conclusions.

The BIC tool partially addresses challenges C1, C2 and C3 ensuring that the results of a manual security analysis are automatically integrated with an existing tool (JIRA). This tool is mainly used by developers, thus aiming for the goal to start developing security tools centered in the developer (G1) and an initial step in the application security as a service infrastructure (G2).

The Version Checker tool addresses challenge C3 enhancing the assessment of dependencies that produces faster results as well as aligning them with the short timespan of dependencies evolution (usually updated due to security reasons). This

TABLE II
VERSION CHECKER RESULTS TABLE

ID	Service	Type	Current Version	Latest Version	Status	LATEST_UPDATED_AT
1	Service #1	N/A	v1.21.1	v1.21.1	🟢	2024-02-20 13:32:50
2	Service #2	N/A	v1.5.2	v1.5.8	🔴	2024-02-20 13:32:50
3	Service #3	1	v2.3.3	v2.3.3	🟢	2024-02-20 13:32:50
4	Service #3	2	v2.3.3	v2.3.3	🟢	2024-02-03 17:02:32
5	Service #4	Internal	v6.2.2	v6.2.4	🔴	2024-02-20 13:32:50
6	Service #4	External	v6.2.4	v6.2.4	🟢	2024-02-20 13:32:50
7	Service #5	N/A	v17.0	v18.0	🔴	2024-02-20 13:32:50
8	Service #6	N/A	v6.2	v6.4	🟡	2024-02-03 17:02:32

TABLE III
CLOUD CLEANER RESULTS TABLE

Owner	Folder Path	Permissions	Expiration Date	Type	Shared Users
User 1	/ProjectX	R C	N/A	User	User 3
User 1	/ProjectY	R E S	N/A	User	User 4
User 2	/Images	R C E S	25/4/2024 00:00:00	User	User1
User 3	/Documents	R C D	2/2/2024 00:00:00	User	User1
User 3	/ProjectZ/Production	R	9/10/2030 00:00:00	Link	None
User 4	/	R C E S D	N/A	Group	Group One

has the direct aim to achieve G3 - Continuous vulnerability discovery and management practices.

The Cloud Cleaner tool addresses challenges C2 and C3 promoting a rapid internal discovery of possible vulnerabilities that can easily be mitigated. This aims for achieving G2 and G3 by providing yet another security testing service while continuously discovering vulnerabilities and management practices.

As we reflect on the journey so far, it becomes evident that this is merely the beginning of a long and promising path. We need to further enhance these tools, by adding new and useful features. Their integration into existing pipelines, automating their execution and the corresponding visualization of the results on real-time dashboards becomes one of the major steps of development for the future.

REFERENCES

[1] Z. Ahmed and S. C. Francis, "Integrating security with devsecops: Techniques and challenges," in *2019 International Conference on Digitization (ICD)*, IEEE, 2019, pp. 178–182.

[2] S. Arachchi and I. Perera, "Continuous integration and continuous delivery pipeline automation for agile software project management," in *2018 Moratuwa Engineering Research Conference (MERCon)*, IEEE, 2018, pp. 156–161.

[3] "Agile manifesto - principles." (2001), [Online]. Available: https://agilemanifesto.org/principles.html (visited on 03/11/2024).

[4] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. IT Revolution, 2021.

[5] "Devsecops toolchain." (2024), [Online]. Available: https://miro.medium.com/v2/resize:fit:1400/0*RX5kPOb6pyf_k5pX (visited on 03/11/2024).

[6] M. Howard and S. Lipner, *The security development lifecycle*. Microsoft Press Redmond, 2006, vol. 8.

[7] A. Dimov and V. Dimitrov, "Classification of software security tools," *Information Systems and Grid Technologies*, 2021.

[8] M. Sánchez-Gordón and R. Colomo-Palacios, "Security as culture: A systematic literature review of devsecops," in *Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops*, 2020, pp. 266–269.

[9] T. Rangnau, R. v. Buijtenen, F. Fransen, and F. Turkmen, "Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines," in *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, IEEE, 2020, pp. 145–154.

[10] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting devsecops: A systematic review," *Information and software technology*, vol. 141, p. 106 700, 2022.

[11] "Owasp application security verification standard." (2024), [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/ (visited on 03/11/2024).

[12] "Owasp asvs checklist for audits." (2024), [Online]. Available: https://github.com/shenril/owasp-asvs-checklist (visited on 03/11/2024).

[13] "Bulk issue creator." (2024), [Online]. Available: https://github.com/nunobernas/BIC (visited on 03/11/2024).

[14] "Owasp top 10: 2021." (2021), [Online]. Available: https://owasp.org/Top10/ (visited on 03/11/2024).

C

ANÁLISE DE SEGURANÇA EM REDES WI-FI

Realizar *pentests* (neste caso internos) de forma recorrente dentro de uma organização é uma prática essencial para garantir a segurança da informação e proteger os ativos da empresa. Um *pentest* (teste de penetração) é uma metodologia de avaliação de segurança em que os analistas, com acesso a toda a documentação pertinente (por exemplo, arquitetura do sistema, código-fonte, manuais, entre outros), tentam identificar e explorar vulnerabilidades nos sistemas de informação, respeitando determinadas restrições impostas (no caso dos *pentests* externos)[48].

Durante o estágio curricular, uma das tarefas solicitadas pela *VOID SOFTWARE, S.A.* foi a realização de testes de complexidade das palavras-passe em duas das redes internas da organização. Por motivos de confidencialidade, os nomes reais das redes não serão revelados e, para efeitos deste relatório, serão referidas como **Rede 1** e **Rede 2**.

Para a execução destes testes, foi utilizada uma máquina com a distribuição Kali Linux, versão 2024.1, preparada especificamente para estas atividades.

As redes alvo da avaliação apresentavam as seguintes características:

1. **Rede 1**: Nesta rede, não era permitido desautenticar dispositivos conectados, o que impôs algumas limitações no processo.
2. **Rede 2**: Nesta rede, não foram impostas restrições, o que possibilitou a realização integral do teste sem impedimentos.

A secção C.2 detalha o procedimento adotado para identificar a palavra-passe da **Rede 1**, enquanto a secção C.3 descreve o método utilizado para obter a palavra-passe da **Rede 2**.

C.1 ALPHA AWUS 1900

Foi necessário utilizar um adaptador *wireless*, uma vez que as ferramentas empregues exigiam acesso direto à placa de rede (Network Interface Card (NIC)) para executar tarefas como captura de pacotes, injeção e monitorização de redes. A máquina virtual utiliza uma placa de rede (NIC) virtualizada, o que limita estas capacidades.

Dessa forma, com um adaptador de rede físico, as ferramentas podem interagir diretamente com o NIC para capturar, analisar e injetar pacotes na rede. Além disso, o desempenho de um adaptador de rede "virtualizado"fica muito aquém do necessário para essas operações. Foi também essencial recorrer a um dispositivo (NIC) que suportasse o modo "Monitor", como abordado na secção C.1.1, uma vez que muitos adaptadores virtualizados não oferecem esse recurso.

O dispositivo escolhido, que atende a todos estes requisitos, foi o "Alfa AWUS1900", ilustrado na Figura 42.

Figura 42: Dispositivo *Alfa AWUS1900* [49]

Este dispositivo permite fazer *scan* tanto a redes com largura de banda de 2.4 GHz como de 5GHz.

C.1.1 *Modos Wireless (Managed & Monitor)*

Existem dois modos de operação em redes *wireless*: *Managed* e *Monitor*. O modo predefinido é o *Managed*.

O primeiro passo para iniciar o processo de descoberta da palavra-passe de uma rede consiste em configurar a interface *wireless* no modo *Monitor*.

O modo *Monitor* permite que o adaptador intercete todos os pacotes transmitidos no ar. Normalmente, a placa de rede apenas "escuta" os pacotes que lhe são diretamente endereçados. No entanto, ao ativar este modo, é possível capturar todos os pacotes, o que nos permite, posteriormente, obter a captura do *4-Way Handshake* (secção C.1.2).

As figuras 43 e 44 mostram os dois modos no dispositivo referido anteriormente na secção C.1:

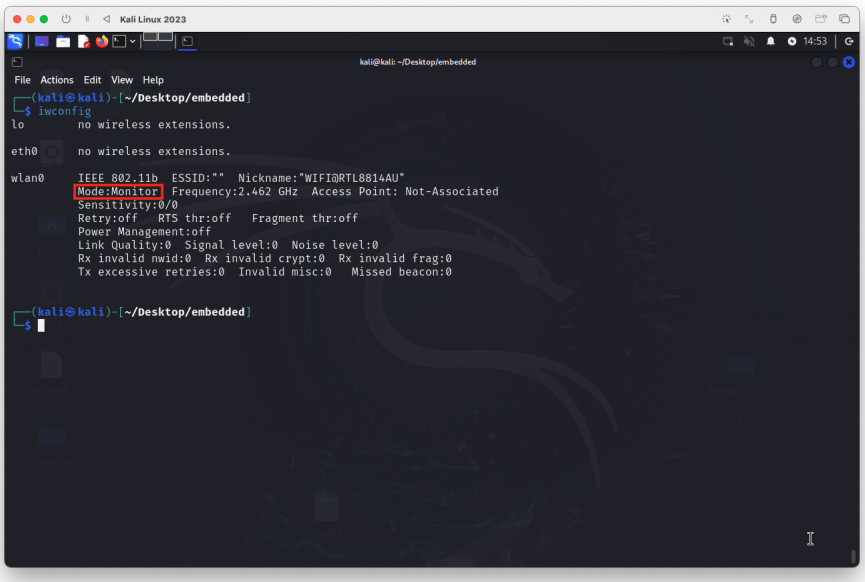

Figura 43: Dispositivo em modo *Monitor*

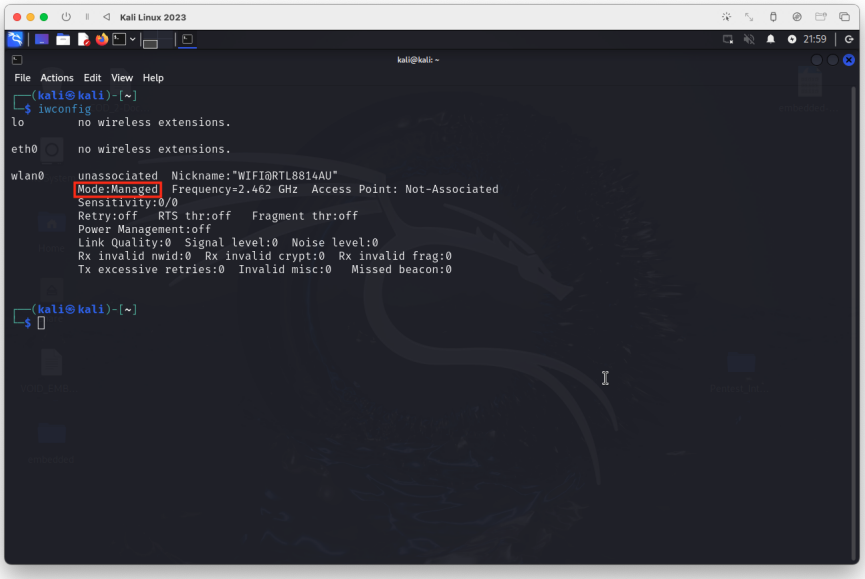

Figura 44: Dispositivo em modo *Managed*

C.1.2 *4-Way Handshake*

Nesta secção do documento, será explicado o que é o *4-Way Handshake* e como funciona.

O *4-Way Handshake* é um protocolo de autenticação de rede que permite que um autenticador e um cliente estabeleçam uma ligação encriptada sem terem de revelar diretamente a palavra-passe (ou *passphrase*). Em vez disso, é utilizada uma chave

derivada da palavra-passe ou de outros métodos de autenticação, conhecida como Pairwise Master Key (PMK).

Na fase inicial da autenticação, que vai desde a associação até à validação da segurança, ocorre o *4-Way Handshake*. Em vez de enviar a palavra-passe para os Access Point (AP) realiza-se uma troca de mensagens Extensible authentication protocol over LAN (EAPOL). Como o nome sugere, há quatro mensagens distintas.

O 4-Way Handshake funciona da seguinte forma:

1. ***EAPOL-Key* - Mensagem 1**: O AP envia o primeiro frame para o cliente ("supplicant") contendo o ANonce (Authenticator Nonce) - um número aleatório que será utilizado como *input* para o cálculo da Pairwise Transient Key (PTK), que será usada para a encriptação da sessão. Neste ponto, o cliente já calculou o seu *SNonce* (*Supplicant Nonce*). Como ambos os dispositivos estão em comunicação, conhecem o endereço MAC um do outro: (1) Supplicant Address (SA) (o endereço MAC do cliente) e (2) Authenticator Address (AA) (o endereço MAC do AP).
2. ***EAPOL-Key* - Mensagem 2**: O cliente cria a sua PTK e responde com o seu *SNonce*, necessário para o AP gerar a sua PTK. Esta resposta é enviada juntamente com o Message Integrity Code (MIC) para garantir que a mensagem não foi modificada durante a comunicação. Ao receber o *SNonce*, o AP pode calcular a PTK e validá-la utilizando o MIC.
3. ***EAPOL-Key* - Mensagem 3**: O AP envia uma mensagem para o cliente contendo a Group Temporal Key (GTK) (Group Temporal Key), que será usada para proteger o tráfego *multicast* e *broadcast*. Juntamente com este valor, é também enviado o MIC para garantir a integridade da mensagem.
4. ***EAPOL-Key* - Mensagem 4**: O cliente confirma que as chaves (PTK e GTK) foram instaladas com sucesso, finalizando o processo de autenticação.

A Figura 49 ilustra o processo descrito anteriormente, o funcionamento deste protocolo de autenticação.

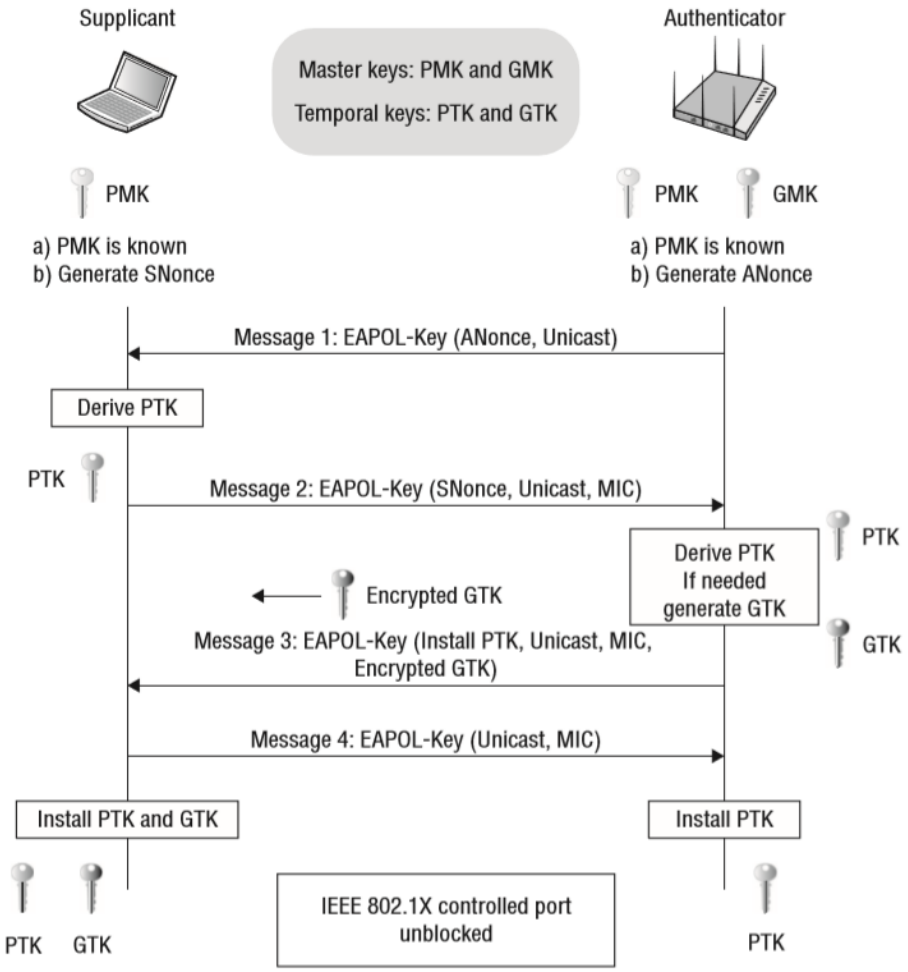

Figura 45: *4-Way Handshake*[50]

O processo de descobrir a palavra-passe de uma rede Wi-Fi, envolve a captura de um *handshake* entre um dispositivo cliente e o AP, seguido pela tentativa de quebrar essa palavra-passe através de *brute-force* ou ataques utilizando *wordlists*.

Os passos a seguir, resumidamente são:

1. Configuração da placa de rede: Configurar a placa de rede para o modo Monitor.
2. Capturar o *Handshake*
3. Utilizar um método de *Brute-Force* para tentar descobrir a password

Os passos 2 e 3 variam nas diferentes abordagens que foram tomadas (devido à limitação referida no início do capítulo).

C.2 WI-FI DEAUTHENTICATION ATTACK

Nesta secção, descreve-se o ataque realizado à **Rede 1**, utilizando um método conhecido como "*Wi-Fi deauthentication attack*". O objetivo principal deste ataque é forçar um dispositivo a desligar-se do AP da rede, permitindo capturar o *4-Way Handshake* quando o dispositivo se volta a ligar.

C.2.1 *Descrição e Objetivo do ataque*

Um *deauthentication attack* tem como objetivo desconectar temporariamente um cliente da rede Wi-Fi, através do envio de pacotes de *deauth* (*deauth*) para o AP e o dispositivo alvo. Forçando esta desconexão, quando o cliente tenta religar-se, o *4-Way Handshake* necessário para a autenticação WPA/WPA2 é transmitido, permitindo a sua captura. Este *handshake* é fundamental, pois contém informações que podem ser usadas para descobrir a palavra-passe da rede, especialmente se for utilizado um ataque de *brute-force*.

C.2.2 *Análise e Identificação da Rede Alvo*

Inicialmente é feita uma análise às redes que foram identificadas com o equipamento referido anteriormente na secção C.1, correndo o comando representado na Listagem 1.

```
1 sudo airodump-ng wlan0
```

Listagem 1: Descobrir redes Wi-Fi com o airodump-ng

A Figura 46 mostra algumas destas redes que foram identificadas no instante em que a o comando foi executado.

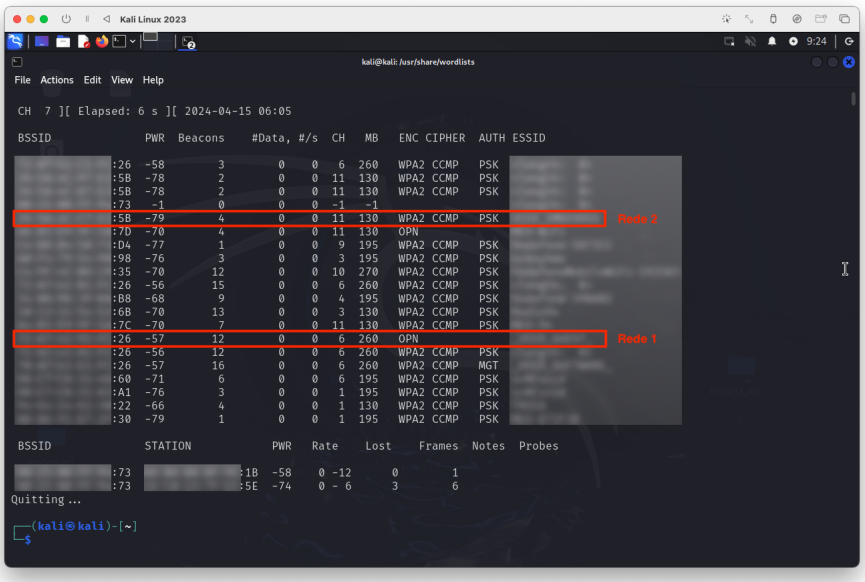

Figura 46: Identificação da rede alvo

Na figura anterior, os campos foram ocultados por razões de confidencialidade. No entanto, a informação exibida pode ser dividida em duas secções distintas, correspondentes às duas tabelas apresentadas:

A tabela superior contém informação referente aos AP detetados. Cada linha representa uma rede individual (SSID) e inclui as seguintes colunas:

- ***BSSID***: O endereço MAC do AP.
- **PWR**: Força do sinal (medida em dBm - *Decibel Milliwatts*). Valores negativos mais elevados indicam sinais mais fracos (por exemplo, -50 dBm é mais forte que -90 dBm).
- ***Beacons***: Número de *frames beacon* transmitidos pelo AP.
- ***#Data***: Quantidade de pacotes de dados capturados a partir do AP.
- ***#/s***: Número de pacotes de dados capturados por segundo.
- ***CH***: Canal de operação do AP no espectro sem fios.
- ***MB***: Velocidade máxima suportada pelo AP, medida em Mbps.
- ***ENC***: Tipo de encriptação utilizado pelo AP.
- ***CIPHER***: Algoritmo de cifra utilizado na encriptação.
- ***AUTH***: Método de autenticação em uso.
- ***ESSID***: Nome da rede sem fios (SSID).

A tabela inferior apresenta informação relativa aos dispositivos conectados ou a sondar redes sem fios. Cada linha corresponde a um único "cliente" e as colunas contêm os seguintes dados:

- ***BSSID***: Endereço MAC do AP ao qual o cliente está associado. Se o cliente não estiver associado a nenhum ponto de acesso, aparecerá a indicação "(não associado)".
- ***STATION***: Endereço MAC do dispositivo cliente.
- ***PWR***: Força do sinal do dispositivo cliente.
- ***Rate***: Taxas de transmissão suportadas pelo dispositivo cliente.
- ***Lost***: Número de pacotes perdidos na comunicação entre o dispositivo e o AP.
- ***Frames***: Número de *frames* (pacotes) capturados do dispositivo.
- ***Notes***: Observações adicionais relevantes sobre o dispositivo (campo opcional, geralmente vazio).
- ***Probes***: Redes que o cliente está a sondar, caso aplicável.

C.2.3 *Execução do Ataque: Captura de Pacotes e Deauth*

Durante o ataque, toda a comunicação na rede é capturada e registada num ficheiro (através do comando da Listagem 2).

```
1 sudo airodump-ng wlan0 -c <numero_do_canal> -d <AP_MacAddress> -w "output"
```

Listagem 2: Comando para capturar pacotes da rede

Após iniciar a captura dos pacotes, numa nova instância do terminal foi executado o comando apresentado na Listagem 3. Este comando injeta cinquenta pacotes de *deauth* no dispositivo com o endereço MAC especificado através da opção "`-a`".

```
1 sudo aireplay-ng -0 50 wlan0 -a <AP_MacAddress> -c <target_MacAddress>
```

Listagem 3: Injeção de pacotes de *deauth*

Este processo força o cliente a se desconectar do AP. Quando o dispositivo tenta se reconectar, o *4-Way Handshake* é transmitido entre o AP e o cliente, permitindo a captura dos pacotes necessários para o próximo passo do ataque.

C.2.4 *Ataque de Brute Force e Recomendações de Segurança*

Com o *handshake* capturado, o próximo passo consiste em realizar um ataque de força bruta (*brute-force*) para tentar descobrir a palavra-passe da rede. Para isso, foi utilizada a ferramenta aircrack-ng, que compara o *handshake* capturado com uma *wordlist* de possíveis palavras-passe, conforme mostrado na Listagem 4.

```
sudo airecrack-ng -w wordlists.txt output.cap
```

Listagem 4: Comando para executar o ataque de *brute-force* com a *wordlist*

O processo de força bruta percorre todas as palavras-passe contidas na *wordlist*, até que a correta seja encontrada. A Figura 47 mostra o resultado quando a palavra-passe é descoberta com sucesso.

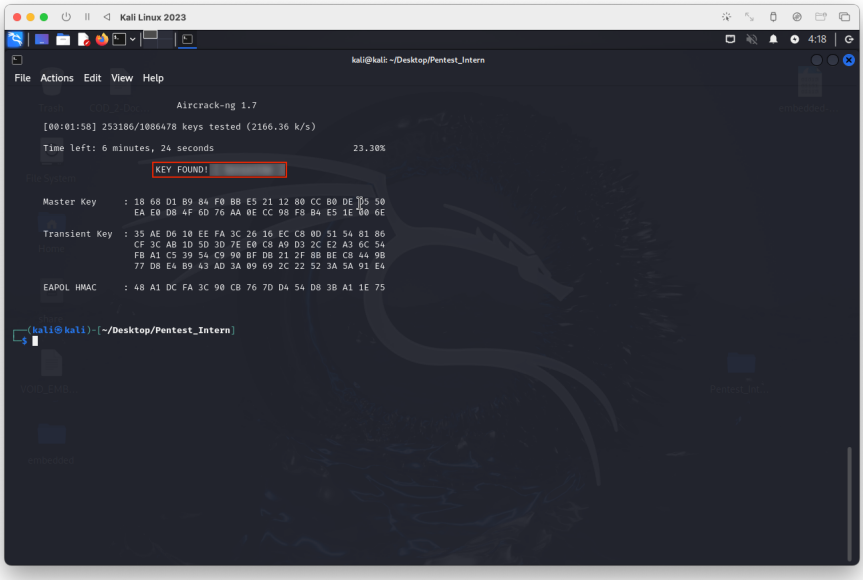

Figura 47: Palavra-passe encontrada!

Após a descoberta da palavra-passe, recomendou-se a implementação de medidas de segurança adicionais, como a utilização de credenciais rotativas ou autenticação mais forte, para prevenir ataques semelhantes no futuro. Além disso, sugeriu-se a revisão periódica das políticas de segurança da rede, visando mitigar vulnerabilidades conhecidas.

A Figura 48 mostra a nova forma de autenticação a esta rede.

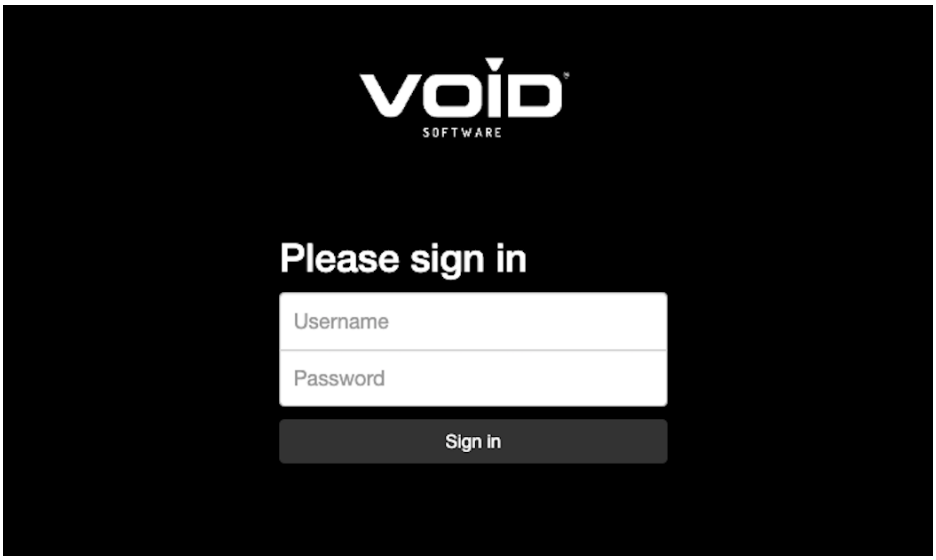

Figura 48: Nova forma de autenticação da Rede 1

C.3 BRUTEFORCE COM *HASHCAT*

O processo descrito nesta secção refere-se ao ataque realizado à rede **Rede 2**. Ao contrário da **Rede 1**, aqui não foi possível forçar uma *deauthentication*, devido à presença de equipamentos críticos que não podiam ser desconectados da rede.

A abordagem, contudo, manteve-se semelhante à anterior. Os passos principais incluíram a configuração da NIC para o modo "Monitor", a captura do *4-Way Handshake* e a tentativa de descoberta da palavra-passe.

Como não era viável induzir a desconexão de nenhum dispositivo, foi necessário monitorizar passivamente o tráfego da rede alvo e aguardar até que algum dispositivo se conectasse por conta própria. Este processo, embora mais lento, foi fundamental para garantir que a operação não interferisse no funcionamento da rede. Após aproximadamente seis horas de captura contínua, foi exibida no terminal a notificação de sucesso, indicando que o *handshake* havia sido obtido, conforme ilustrado na Figura 49.

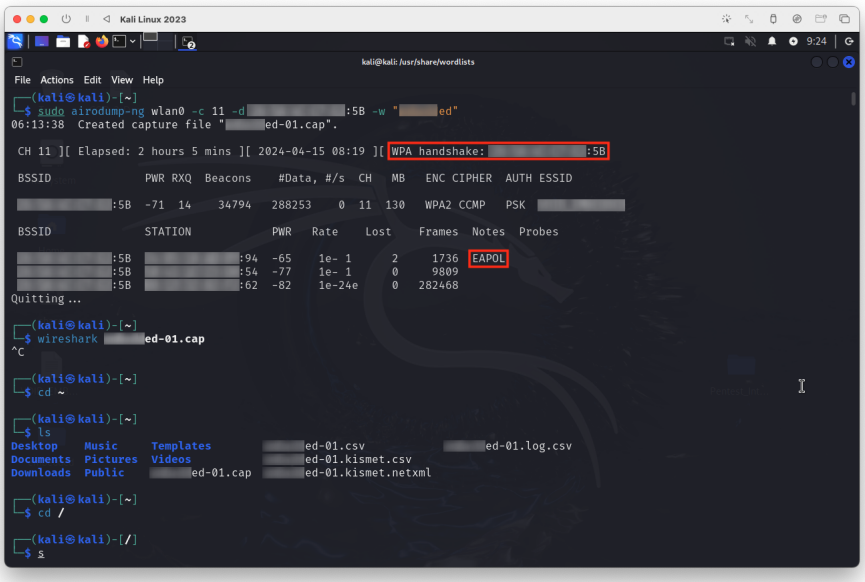

Figura 49: Captura do *4-Way Handshake*

A Figura 50 apresenta as mensagens EAPOL do ficheiro que contém o *handshake* no Wireshark. O Wireshark é uma ferramenta de análise de protocolos de rede que permite capturar e inspecionar pacotes de dados em tempo real[51]. Esta ferramenta é essencial para a análise de segurança, pois possibilita a visualização dos detalhes das trocas de mensagens, e neste caso, o *4-Way Handshake*, como ilustra a Figura 50.

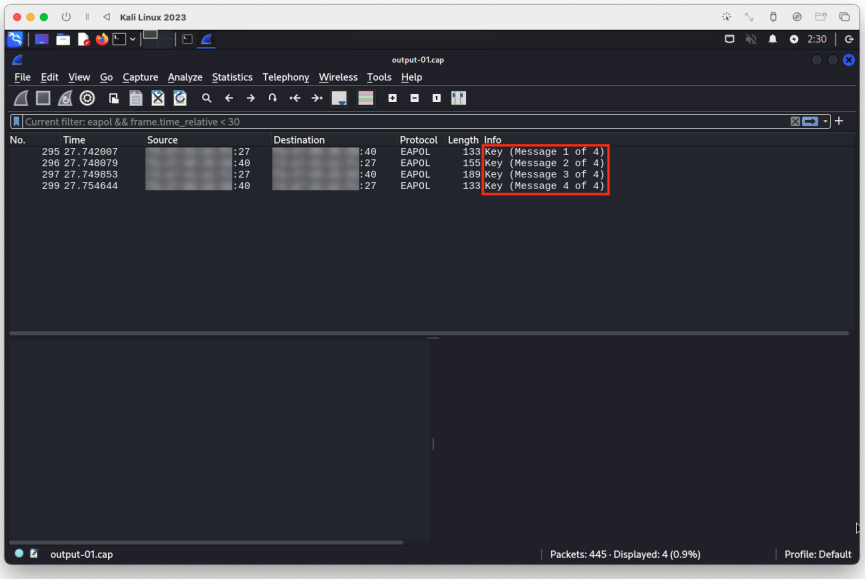

Figura 50: Mensagens *EAPOL*

Com o *handshake* capturado, inicialmente foram utilizadas algumas *wordlists* simples para tentar descobrir a palavra-passe, mas sem sucesso. Como alternativa,

optei por utilizar a ferramenta *hashcat*. O *Hashcat* é o utilitário de recuperação de passwords mais rápido e avançado do mundo, suportando cinco modos de ataque distintos para mais de 300 algoritmos de *hashing* altamente otimizados. Atualmente, o *hashcat* suporta *CPU*, *GPU* e outros aceleradores de hardware e possui funcionalidades que permitem a quebra distribuída de passwords[52].

Primeiramente, foi necessário converter o *handshake* capturado para um formato compatível com o *hashcat*. Para isso, utilizei um utilitário específico do *hashcat*, chamado **cap2hccapx**[53].

```
sudo cap2hccapx.bin input.cap output.hccapx
```

Listagem 5: Conversão do ficheiro *.cap* para *.hccapx*

Este comando gera um ficheiro com a extensão ".hccapx" a partir do *handshake* capturado (output.hccapx).

Em seguida, para iniciar o ataque com o *hashcat*, executei o seguinte comando:

```
hashcat -m 2500 -a 3 -1 ?l?u?d /path/to/file.hccapx ?1?1?1?1?1?1?1?1
-i --increment-min=8 --deprecated-check-disable
```

Listagem 6: Comando executado para iniciar o *hashcat*

O comando acima, significa:

- **-m 2500**: Especifica o tipo de *hash* como WPA/WPA2.
- **-a 3**: Especifica o modo de ataque como *brute force*.
- **-1 ?l?u?d**: Especifica o conjunto de caracteres para a palavra-passe.
  - **?l** representa letras minúsculas
  - **?u** representa letras maiúsculas
  - **?d** representa dígitos
- **?1?1?1?1?1?1?1?1?1**: A máscara para a password mínima de 8 caracteres.
- **-i**: Ativa o modo incremental.
- –**increment-min=8**: Define o comprimento mínimo da palavra-passe para 8 caracteres.
- –**deprecated-check-disable**: ativar *plugins* obsoletos (tipo de hash 2500 é obsoleto)

O comando inicia a tentativa de todas as combinações possíveis de palavras-passe com um comprimento mínimo de oito caracteres, aumentando gradualmente.

Este processo demonstrou ser demorado. A Figura 51 ilustra a estimativa de tempo necessária para a conclusão do comando apresentado na Listagem 5:

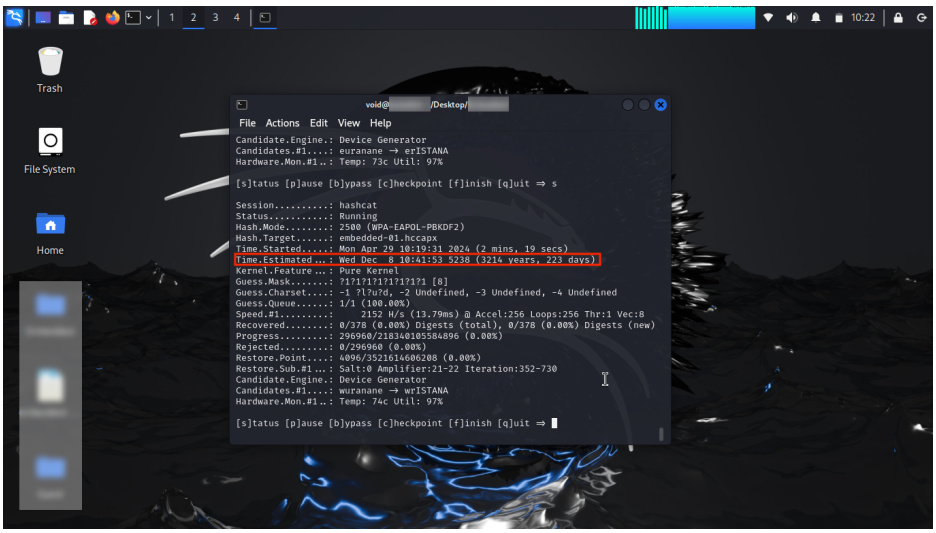

Figura 51: Duração da execução do comando

Devido a esses tempos totalmente impraticáveis, decidi explorar outras abordagens.

Experimentei usar a ferramenta *hydra* para tentar realizar o ataque de *brute force*, mas a estimativa de tempo era absurdamente longa, chegando a ser de anos. Apesar de conhecer o tamanho das credenciais (que não eram muito extensas), o uso do *hydra* revelou-se também ele impraticável nesta situação.

É importante notar que, caso tivesse utilizado uma máquina com especificações superiores, os tempos de execução poderiam ter sido significativamente reduzidos.

Após diferentes tentativas de ataque, conclui-se que a palavra-passe da rede permanece segura, pelo menos a este nível de ataque.

D

OWASP ASVS - EXEMPLO DE UMA *WORKSHEET*

Este apêndice contem uma que imagem mostra como se encontra estruturada uma folha de Excel da *checklist* OWASP ASVS.

+ | Session Management | Access Control | Input Validation | Cryptography at Rest | Error Handling and Logging | Data Protection

Area	#	ASVS Level	CWE	NIST	Verification Requirement	Valid	Source Code Reference	Comment	Tool Used
Data Classification	6.1.1	2	311		Verify that regulated private data is stored encrypted while at rest, such as Personally Identifiable Information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR.				
	6.1.2	2	311		Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records.				
	6.1.3	2	311		Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records.				
Algorithms	6.2.1	1	310		Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.				
	6.2.2	2	327		Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography. ([C8](https://owasp.org/www-project-proactive-controls/#div-numbering))				
	6.2.3	2	326		Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice.				
	6.2.4	2	326		Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks. ([C8](https://owasp.org/www-project-proactive-controls/#div-numbering))				
	6.2.5	2	326		Verify that known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility.				
	6.2.6	2	326		Verify that nonces, initialization vectors, and other single use numbers must not be used more than once with a given encryption key. The method of generation must be appropriate for the algorithm being used.				
	6.2.7	3	326		Verify that encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party.				
	6.2.8	3	385		Verify that all cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information.				
Random values	6.3.1	2	338		Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker.				
	6.3.2	2	338		Verify that random GUIDs are created using the GUID v4 algorithm, and a Cryptographically-secure Pseudo-random Number Generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable.				
	6.3.3	3	338		Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.				
Secret Management	6.4.1	2	798		Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets. ([C8](https://owasp.org/www-project-proactive-controls/#div-numbering))				
	6.4.2	2	320		Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations. ([C8](https://owasp.org/www-project-proactive-controls/#div-numbering))				

Figura 52: Exemplo de uma folha de Excel da *checklist* OWASP ASVS

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Desenvolvimento de Ferramentas e Práticas para uma Implementação Efetiva de DevSecOps"*, é original e foi realizado por Nuno André Faustino Bernardino (2230461) sob orientação de Professor Doutor Filipe Santos Neves (fneves@ipleiria.pt).

Leiria, Junho de 2024

Nuno André Faustino Bernardino