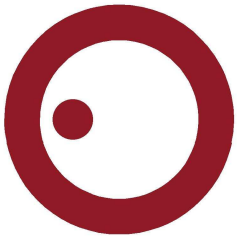


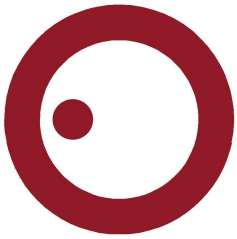
IPL

escola superior de tecnologia e gestão
instituto politécnico de leiria

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

WEB SECURITY APPLICATION PROJECT

JOÃO PEDRO BORGES FERREIRA CASTANHEIRA MARQUES



escola superior de tecnologia e gestão
instituto politécnico de leiria

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

WEB SECURITY APPLICATION PROJECT

JOÃO PEDRO BORGES FERREIRA CASTANHEIRA MARQUES
Número: 2192637

Relatório de estágio realizado sob orientação do Professor Doutor Marco António de Oliveira Monteiro (marco.monteiro@ipleiria.pt).

AGRADECIMENTOS

Gostaria de agradecer à *VOID SOFTWARE, S.A.* pela oportunidade da realização deste estágio num ambiente empresarial. Em especial consideração ao Marco Cova por supervisionar o desenvolvimento e a todos os membros da *VOID SOFTWARE, S.A.* que me acompanharam durante todo o percurso.

Queria também dar uma salva ao orientador Professor Marco Monteiro pela sua excelência em todo o processo de construção deste documento, que esteve sempre disponível e pronto para sugerir novas alterações.

Para a redação deste documento foi usado *Latex*. pelo que também gostaria de agradecer ao Professor Miguel Monteiro de Sousa Frade por providenciar o *template* e dispor do seu tempo para responder a questões sobre esta *framework*.

Por fim, gostaria de agradecer a família e amigos que me acompanharem ao longo desta jornada e estiveram presentes em todos os momentos.

RESUMO

Este relatório descreve o trabalho realizado durante o estágio na *VOID SOFTWARE, S.A.* no âmbito da conclusão do Mestrado de Cibersegurança e Informática Forense.

A *VOID SOFTWARE, S.A.* é uma empresa multifacetada de desenvolvimento de software, que aposta na implementação de segurança e, atualmente, emprega-a no seu processo atual de desenvolvimento.

Na decurso deste estágio, que teve duração de 9 meses, foi lançado o desafio de reforçar os processos existentes na *VOID SOFTWARE, S.A.* ao providenciar uma camada adicional de segurança, o que culminou no desenvolvimento da solução *Web Security Application Project*. Esta solução pretende aplicar o mecanismo de segurança *Security Automation*, ser fácil de integrar no processo da empresa e aplicar diversos paradigmas para encontrar vulnerabilidades.

Todo o processo de desenvolvimento foi seguido segundo as práticas da empresa, através da aplicação de uma metodologia ágil, sendo optada a metodologia *Kanban* por permitir realizar o planeamento, desenvolvimento e seguimento do trabalho realizado.

ABSTRACT

This report describes the work done during the internship at *VOID SOFTWARE, S.A.* as part of the Master's degree in Cybersecurity and Computer Forensics.

VOID SOFTWARE, S.A. is a multifaceted software development company that is committed to implementing security and employs it in it's current development process.

During this internship, which lasted 9 months, the company challenged me to strengthen their existing processes by providing an additional layer of security, which culminated in the development of the solution *Web Security Application Project*. This solution aims to implement the security mechanism *Security Automation*, be easy to integrate in the company's process and be able to identify vulnerabilities using multiple paradigms.

The development process that was followed applied the company's practices, using the agile methodology *Kanban* in order to plan, develop, and follow-up all the work.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AJAX	Asynchronous JavaScript And XML.
CI/CD	Continuous Integration and Continuous Delivery.
CLI	Command Line Interface.
CWE	Common Weakness Enumeration.
CWRAF	Common Weakness Risk Analysis Framework.
CWSS	Common Weakness Scoring System.
DAST	Dynamic Application Security Testing.
HPA	Hybrid Page Application.
HTML	Hyper Text Markup Language.
HTTP	Hyper Text Transfer Protocol.
IC	Integração Contínua.
ITU-T	International Telecommunication Union.
JSON	JavaScript Object Notation.
MPA	Multi Page Application.
NIST	National Institute of Standards and Technology.
OSI	Open System Interconnection.
OWASP	Open Web Application Security Project.

SAMATE	Software Assurance Metrics and Tool Evaluation.
SAST	Static Application Security Testing.
SCM	Source Code Management.
SDLC	Software Development Life Cycle.
SO	Sistema Operativo.
SPA	Single Page Application.
SSH	Secure Shell.
TCP	Transmission Control Protocol.
WSAP	Web Security Application Project.
X.800	X.800, Security Arquiteture for OSI.
XML	eXtended Markup Language.

1 INTRODUÇÃO

Garantir e inovar com segurança é um requisito essencial, para quem pretende ser líder do mercado. A *VOID SOFTWARE, S.A.* com a sua equipa de excelência não é exceção, com a sua versatilidade, capacidade de inovar e de aceitar riscos tornou possível, no âmbito do Mestrado de Cibersegurança e Informática Forense, a realização deste estágio curricular.

Neste documento será descrito todo o trabalho desenvolvido, no estágio curricular, realizado na *VOID SOFTWARE, S.A.* No contexto do estágio foi proposto, para a implementação da segurança, o desenvolvimento de uma solução automática de análise de vulnerabilidades *Web* que possa ser integrada no pipeline *Continuous Integration and Continuous Delivery (CI/CD)* da empresa e desta forma encontrar as vulnerabilidades antes que cheguem ao consumidor.

Será documentado todo o processo de desenvolvimento realizado. Primeiramente serão explorados alguns dos conceitos tidos como fundamentais para o entendimento da solução. No estado de arte será realizado um estudo exploratório das várias ferramentas disponíveis, assim como a seleção das mesmas. Por fim, no desenvolvimento, será apresentado, a metodologia de trabalho, estrutura da solução e escolhas tomadas.

1.1 MOTIVAÇÃO

O desenvolvimento das *Web Applications* requer todo um processo que começa no planeamento e termina na fase de distribuição.

A implementação da segurança ao longo do *pipeline* de desenvolvimento, permite endereçar o mais rapidamente possível as falhas do software, reduzindo esforço e custos adicionais bem como atestar a qualidade e segurança da aplicação.

Para tal, devem existir processos, metodologias e mecanismos que permitam eliminar as vulnerabilidades, ou se não for possível, minimizar o risco das mesmas. Caso contrário, se as vulnerabilidades não forem devidamente tratadas, estas podem

ser usadas para afetar negativamente a organização, desde extorsão de dinheiro até manchar a própria reputação.

A *Security Automation* é um dos possíveis mecanismos de segurança, que segundo Hoffman (2020) destaca-se pelo custo, eficiência e longevidade.

Face a estas vantagens, é proposta a implementação de uma solução *Security Automation* que engloba várias ferramentas de carácter *open-source* e as integra no pipeline de desenvolvimento.

1.2 ENTIDADE DE ACOLHIMENTO

A *VOID SOFTWARE, S.A.*, é uma empresa multifacetada e flexível de desenvolvimento de software, fundada em Leiria no ano 2006. Atualmente emprega mais de 30 trabalhadores e pretende continuar a crescer ao apostar no mercado europeu e americano.

No planeamento das suas soluções de software, a *VOID SOFTWARE, S.A.* aplica principalmente as metodologias ágeis, mas está recetiva a adotar outras metodologias com base nas necessidades do cliente.

Em relação às soluções desenvolvidas, estas são adequadas aos requisitos do cliente podendo ser aplicações: *Mobile*, *Web* ou *Desktop*.

A *VOID SOFTWARE, S.A.* disponibiliza ainda serviços e soluções mais avançadas, para satisfazer as novas necessidades do mercado, explorando as seguintes áreas:

- *BlockChain*;
- *Machine Learning / Data Science*;
- Realidade Aumentada / Realidade Virtual;
- Cibersegurança / Forense digital.

1.3 OBJETIVOS

No mestrado de Cibersegurança e Informática Forense foram abordados vários mecanismos, processos e metodologias de cibersegurança, sendo *Security Automation* um dos mecanismos utilizados pelas *Blue Teams* na defesa dos ativos.

No âmbito do estágio curricular foi proposto a implementação de uma solução de *Web Vulnerability Scanning*, sendo recomendado que esta englobe as várias ferramentas de análise de vulnerabilidades já existentes.

Esta solução, deve ter a capacidade de realizar análises estáticas e dinâmicas e ser integrável no pipeline de desenvolvimento da *VOID SOFTWARE, S.A.*

Para responder a estes requisitos, foi necessário realizar um levantamento das ferramentas de análise estática e dinâmicas, conhecer as vulnerabilidades mais usuais em *Web* e adaptar as ferramentas de forma a terem a capacidade para analisar as aplicações em desenvolvimento na *VOID SOFTWARE, S.A.*

Por fim, a solução final deve ser integrada com a ferramenta *Jenkins*, que se encontra atualmente integrada no pipeline de desenvolvimento da *VOID SOFTWARE, S.A.*

1.4 ESTRUTURA DO RELATÓRIO

Em relação à estrutura deste documento, encontra-se subdividido em vários capítulos.

No **Capítulo 1**, "Introdução", é realizado o enquadramento do estágio, da entidade empregadora e a estrutura do documento.

No **Capítulo 2**, "Conceitos Fundamentais", irão ser abordados os conceitos principais, sendo primeiramente introduzidas as *Web Applications* e a *Segurança de Computadores*. Será realizado um enquadramento entre elas. Introduzir-se-ão os vários fatores que contribuem para o risco. Serão explorados em mais detalhe estudos de referência de mercado que identificam e classificam as vulnerabilidades mais recorrentes em *Web Aplications*.

Apresentar-se-ão as várias recomendações para um desenvolvimento de software seguro e introduzir-se-á o tema de *Security Automation*, que abrange: *Static Aplication Security Testing* e *Dynamic Application Security Testing*.

No **Capítulo 3**, "Estado de Arte", realizar-se-á uma introdução à ferramenta de desenvolvimento de *Continuous Intragration/Continuos Delivery*, *Jenkins*. Será também feito um levantamento e análise comparativa das ferramentas de análise dinâmica e de análise estática.

No **Capítulo 4**, "Desenvolvimento", abordar-se-á todo o processo de desenvolvimento da solução WSAP. Neste será explorado a metodologia de desenvolvimento, a solução desenvolvida e os testes efetuados para assegurar a sua qualidade.

No **Capítulo 5**, "Conclusões", será realizado uma retrospetiva do trabalho desenvolvido, possíveis melhorias e realçado a importância do mesmo.

2

CONCEITOS FUNDAMENTAIS

Neste capítulo serão explorados os conceitos necessários para o entendimento do trabalho desenvolvido. Na **Secção 2.1** (Web Applications) serão abordadas as *Web Applications*, sendo estudadas: as suas arquiteturas, os componentes que as definem e as diferentes implementações destas.

De seguida, na **Secção 2.2** (Segurança de Computadores), serão explorados os conceitos de segurança de computadores, onde se introduzirão os pilares de segurança, a arquitetura de segurança e os princípios a ter em conta na sua implementação.

Devido às particularidades das *Web Applications*, na **Secção 2.3** (Segurança Aplicada a Web), serão aprofundados conceitos de segurança relativos a este tipo de soluções.

Na **Secção 2.4** (Ameaças, Vulnerabilidades e Riscos) serão explorados os diferentes fatores que podem afectar um sistema e algumas *frameworks* que auxiliam na deteção, resolução ou mitigação das falhas de segurança.

Na **Secção 2.5** (Segurança no Software Development Life Cycle), será reforçado, a importância da implementação da segurança ao longo de todo o processo de desenvolvimento e explorar-se-ão as soluções Static Application Security Testing (SAST) e Dynamic Application Security Testing (DAST).

2.1 WEB APPLICATIONS

As *Web Application* são a base deste projeto, pelo que antes de se estudar os scanners de segurança é importante entender um pouco da sua origem e os potenciais pontos de falha presentes neste tipo de solução.

De acordo com Hoffman (2020), o aparecimento da *Internet*, uma rede a nível mundial de computadores ligados entre si, impulsionou a partilha de informação através da tecnologia *Web* com a evolução desta tecnologia surgiram as *Web Applications* da atualidade, que segundo Laurent (2014), ao separarem as responsabilidades entre utilizadores e o sistema permitem ser facilmente distribuídas, mantidas e

escaladas com base na necessidade. Através desta flexibilidade, as *Web Applications* tornaram-se a opção predileta para endereçar o problema de aplicações distribuídas. Este tipo de aplicações caracterizam-se pelo uso da arquitetura *client-servidor*, no qual um cliente faz um pedido e uma máquina dedicada fica encarregue de dar resposta.

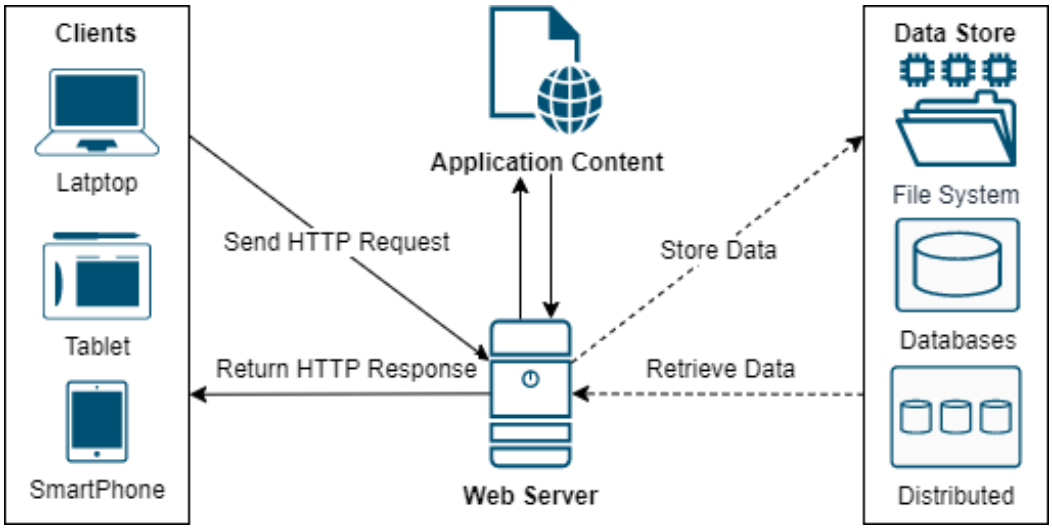


Figura 1: *Web Application - Architecture*

Na **Figura 1**, *"Web Application - Architecture"*, é possível consultar um exemplo focado para a vertente de segurança da arquitetura de uma *Web Application* que serve diversos clientes. São apresentados os componentes que constituem a *Web Application*, as interações que são realizadas no decorrer de um pedido e algumas das formas de armazenamento que podem estar em uso no decorrer do processo.

De acordo Cross et al. (2007), os componentes que se destacam nesta arquitetura são:

- ***Web Server*** - infraestrutura responsável por receber/responder aos clientes e comunicar com o(s) Data Store(s);
- ***Application Content*** - aplicação responsável por processar os pedidos e parâmetros enviados pelo cliente e realizar ações;
- ***Data Store*** - infraestrutura, solução ou ficheiro responsável por armazenar os dados da aplicação de forma persistente.

A compreensão dos diferentes componentes que constituem as *Web Application* assim como as interações realizadas entre eles são essenciais na deteção de potenciais falhas, que se não forem tratadas, podem ser exploradas para fins maliciosos.

2.1.1 *Componentes das Web Applications*

A perceção da arquitetura e de como estes componentes interagem entre si é essencial para a segurança. Permite às equipas defensoras (*Blue Teams*) tornar o sistema mais compacto e protegido, e às equipas atacantes (*Red Team*) aumentar a precisão dos ataques.

Na secção anterior foram abordados os diferentes componentes que definem a arquitetura das *Web Applications*, sendo estes: *Web Server*, *Application Content* e *Data Stores*. Segundo Cross et al. (2007) também é necessário identificar os componentes de software que podem apresentar falhas de segurança, nomeadamente:

- ***Login*** - processo de autenticação do utilizador na aplicação realizado através de credenciais (**e.g.** Bypass da Autenticação...);
- ***Session Tracking Mechanism*** - mecanismo usado para identificar e manter a sessão do utilizador autenticado, usualmente *Session Cookies* (**e.g.** Session Hijacking...);
- ***User Permissions Enforcement*** - coexistência de utilizadores com diferentes níveis de privilégios;
- ***Role Level Enforcement*** - coexistência de grupos de utilizadores com diferentes níveis de privilégios;
- ***Data Access*** - acesso aos *Data Store* da aplicação é garantido e salvaguardado (**e.g.** *SQL Injections*...);
- ***Application Logic*** - falhas na lógica da aplicação em si que comprometam o sistema ou os dados (**e.g.** Valores Inválidos, Problemas de concorrência, Erros de Execução...);
- ***Logout*** - processo que garante que o término da sessão do utilizador é refletida no servidor.

2.1.2 *Tipos de Web Applications*

Com a evolução da tecnologia, surgiram diferentes tipos de *Web Applications* para endereçar os diversos casos de uso, inclusive os presentes na **Secção 2.1.1** (Componentes das Web Applications). Devido às alterações significativas que os diferentes tipos de *Web Applications* apresentam, torna-se importante compreender o seu paradigma assim como o seu funcionamento.

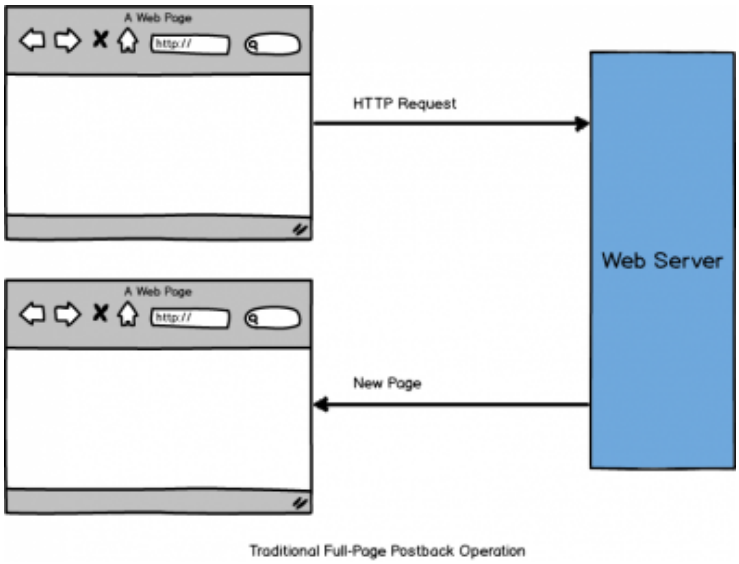


Figura 2: *Multi Page Application* (A. Team, 2013)

As primeiras *Web Applications* baseavam-se no modelo Multi Page Application (MPA), também conhecidas como *Traditional Web Applications*. Este modelo, apresentado na **Figura 2**, é caracterizado por utilizar múltiplas páginas no processo de interação com o utilizador contudo, a falta de responsividade e o elevado tempo de espera entre interações, levou ao surgimento das Single Page Application (SPA), também conhecidas como *Modern Applications*.

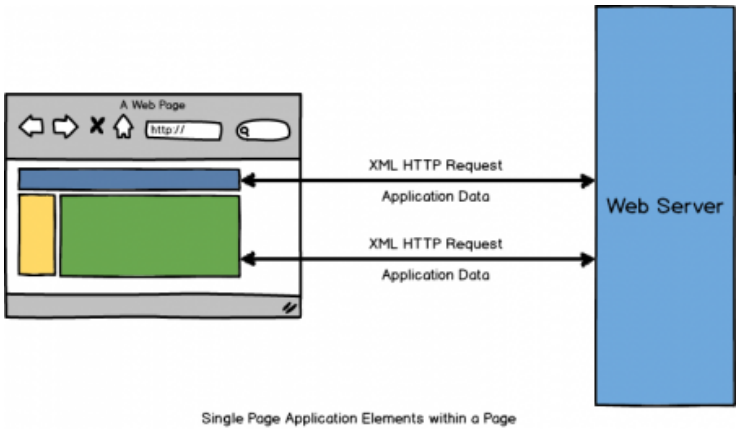


Figura 3: *Single Page Application* (A. Team, 2013)

O modelo SPA, apresentado na **Figura 3**, ao invés de usar múltiplas páginas como o seu predecessor, apenas carregada uma única página e futuras interações são realizadas a partir da mesma.

Existe ainda o modelo Hybrid Page Application (HPA). Pretende ser um compromisso entre uma razoável experiência de navegação e o modelo tradicional, mas por incorporar dois paradigmas apresenta um maior esforço de manutenção.

Segundo Skólski (2016), todos os modelos são atualmente empregues e apresentam vantagens, por um lado, as MPA demonstram um grande grau de maturidade e robustez, sendo recomendadas para aplicações de grandes dimensões, enquanto que as aplicações SPA proporcionam uma excelente experiência de navegação.

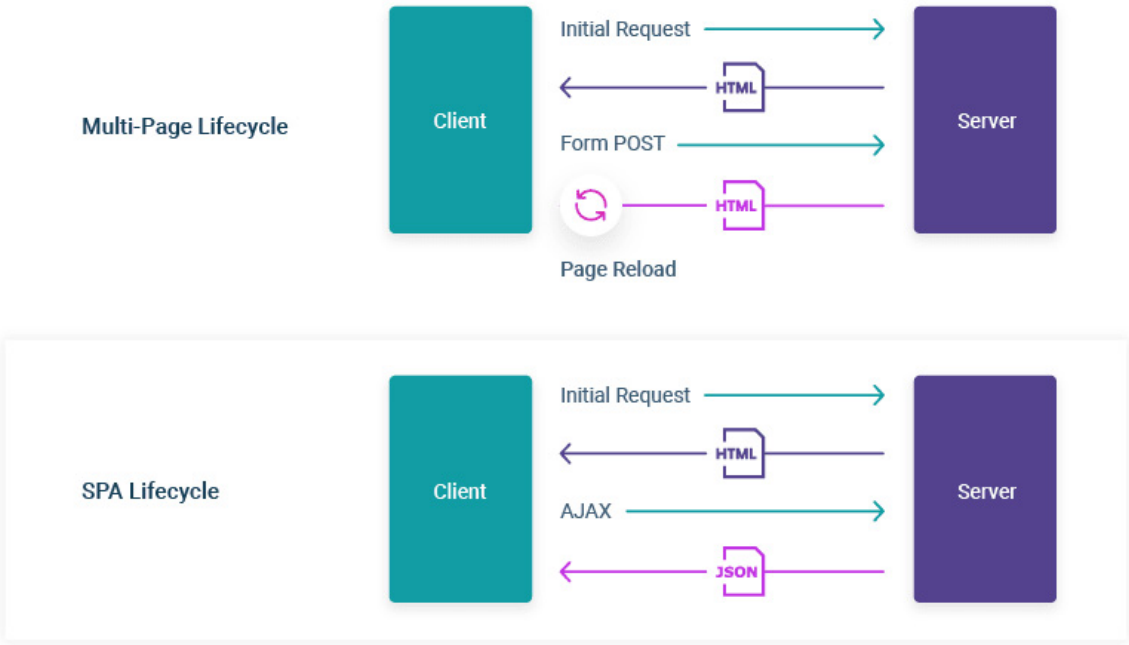


Figura 4: Comparação - *MPA* e *SPA* (L. Team, 2020)

Na **Figura 4**, *"Comparação - MPA e SPA"*, é evidenciado algumas das diferenças existentes nos modelos de *Web Applications* previamente introduzidos. No modelo MPA sempre que o cliente interage com a página HTML, esta efetua um pedido HTTP para o servidor. Após receber e processar este pedido, o servidor responde com uma nova página HTML que cliente tem de carregar.

No modelo SPA a interação com o cliente é realizada através de uma linguagem como *Javacript*. Quando este interage com a página, o *Javascript* é usado para gerar e enviar um pedido HTTP para o servidor. O servidor após receber e processar este pedido, envia apenas os dados requeridos pela acção. O *browser* do cliente, através do *Javascript* processa os dados e atualiza a página de acordo com os mesmos.

Devido às disparidades existentes entre os modelos MPA e SPA, é essencial a compreensão do paradigma, das tecnologias empregues e do próprio processo

de comunicação, visto que, no âmbito das ferramentas de análise de segurança, as disparidades presentes nestes modelos podem implicar um processo de análise diferenciado, ou até mesmo a aplicação de diferentes ferramentas para contemplar cada modelo.

2.2 SEGURANÇA DE COMPUTADORES

Segundo a Organização National Institute of Standards and Technology (NIST), o termo de **segurança de computadores** pode ser descrito como a proteção conferida a um sistema de informação automatizado, a fim de, atingir os objetivos aplicáveis de preservação da integridade, disponibilidade e confidencialidade dos recursos do sistema de informação (inclui hardware, software, firmware, informações / dados e telecomunicações) (JTFTI, 2011).

Nesta definição são abordados os objetivos principais de segurança, que de acordo com Kaspersky (2017), originaram a criação de diversas áreas, nomeadamente:

- ***Network security*** - salvaguarda a infraestrutura de rede contra intrusos;
- ***Application security*** - assegura a segurança de dispositivos e *software*;
- ***Information security*** - garante que o armazenamento e a transferência de dados é realizado de forma segura;
- ***Operational security*** - define os processos e decisões aplicados no tratamento e proteção de dados;
- ***Disaster recovery and business continuity***, define a resposta a um incidente de segurança salvaguardando a continuidade das operações;
- ***End-user education*** - sensibiliza as pessoas para as boas práticas, procedimentos seguros, entre outros.

No âmbito da solução proposta, esta enquadra-se principalmente na área da *Application Security* visto pretender resguardar a segurança das *Web Applications*, mas também afeta outras áreas como *Information Security* e *Operation Security* no tratamento e processamento de dados, respetivamente.

Nas seguintes sub-secções serão explorados, em mais detalhe, os pilares da segurança mencionados na definição de segurança de computadores. Será realizado um enquadramento da solução pretendida com a arquitetura Open System Interconnection (OSI), que define os requisitos de segurança.

2.2.1 *Pilares de Seguranças*

Na definição de segurança de computadores, previamente definido na **Secção 2.2** (Segurança de Computadores) foram abordados os objetivos fundamentais de segurança, também conhecidos como *CIA triad*, sendo estes:

- ***Confidencialidade:*** preservar as restrições de autorização que delegam o acesso à informação e divulgação, incluindo meios para proteger a privacidade e informação proprietária;
- ***Integridade:*** proteger contra a modificação ou destruição indevida da informação, incluindo o não repúdio e autenticidade;
- ***Disponibilidade:*** garantir a fiabilidade e o acesso atempado da informação e uso da mesma.

A compreensão destes objetivos fundamentais de segurança torna-se crucial, porque ao indicar os pontos de um sistema, que podem ser comprometidos, permitem facilmente e intrinsecamente caracterizar as diversas vulnerabilidades.

2.2.2 *Arquitetura de Segurança Open System Interconnection*

Para assegurar e cumprir com as necessidades de segurança da entidade, o gestor responsável pela segurança necessita de um processo sistemático que defina os requisitos de segurança e medidas para os cumprir (Stallings, 2005).

Devido à necessidade de coerência na definição de tal processo a International Telecommunication Union (ITU-T) publicou o *standard* X.800, Security Arquiteture for OSI (X.800) que aborda as temáticas de segurança, boas práticas e auxílio ao gestor de segurança.

Neste *standard* X.800, são abordados inúmeros conceitos, que de acordo com Stallings (2005) podem ser enquadrados nos seguintes tópicos:

- ***Security attack*** - qualquer ação que compromete a segurança da informação;
- ***Security mechanism*** - processo que é desenhado para detetar, prevenir ou recuperar de um ***Security attack***;
- ***Security service*** - serviço que realça um objetivo fundamental de segurança, através da aplicação de um ou mais ***Security mechanisms***.

Tendo como base a arquitetura de segurança OSI, pode afirmar-se que a solução pretendida corresponde a um mecanismo de segurança, que irá detetar e sugerir resoluções para as vulnerabilidades presentas nas *Web Applications* analisadas.

2.2.3 *Segurança Aplicacional e Princípios de Resiliência*

Na **Secção 2.2.1** (Pilares de Seguranças) foram abordados os serviços fundamentais de segurança. Também existem outros serviços importantes que exploram diferentes vertentes de segurança.

A resiliência é um desses serviços de segurança, que segundo Rieger et al. (2009), assegura a capacidade de um sistema manter um estado de consciência e a nível operacional aceite em resposta a distúrbios, incluindo ameaças de natureza inesperadas e maliciosas.

Baseados nos riscos de segurança identificados no Open Web Application Security Project (OWASP), ***OWASP - Top Ten***, Merkow (2019) formalizou o conjunto de princípios de resiliência, no qual engloba as boas práticas, padrões e metodologias; adaptados à vertente *Web*, sendo estes:

1. **Aplicar defesa em profundidade** - os sistemas devem ter múltiplos níveis de defesa desde: hardware, software, etc;
2. **Usar um modelo de segurança positivo** - deve ser aplicado uma metodologia *whitelist* nos parâmetros processados na aplicação;
3. **Falhar em segurança** - os erros da aplicação devem ser devidamente tratados para impedir que o sistema entre num estado anómalo;
4. **Executar com o mínimo de privilégios** - os utilizadores devem apenas ter os privilégios básicos para o seu processo de negócio, o que inclui os seus direitos e recursos;
5. **Evitar a segurança através da obscuridade** - a implementação de controlos de segurança num sistema deve ser aberta e transparente;
6. **Manter a segurança simples** - a implementação da segurança não deve aumentar a complexidade da solução ou sistema;
7. **Detetar intrusões** - devem existir controlos implementados para a monitorização dos sistemas;
8. **Não confiar na infraestrutura** - as aplicações devem cumprir com requisitos de segurança pré-estabelecidos;

9. **Não confiar nos serviços** - devem existir controlos adicionais de segurança para validar interação com serviços externos;
10. **Definir *defaults* de segurança** - as pré-definições da aplicação devem ser as configurações mais seguras.

2.3 SEGURANÇA APLICADA A *WEB*

As *Web Applications* são soluções de software, estudadas na **Secção 2.1**, que apresentam múltiplas características incluindo uma arquitetura e requisitos funcionais próprios.

Devido às diversas particularidades das *Web Applications*, torna-se necessário aprofundar alguns dos conceitos de segurança característicos destas soluções. Para tal, é introduzido o conceito de *Attack Surface*, assim como algumas das técnicas usadas para o seu mapeamento.

2.3.1 *Introdução à Attack Surface*

A *Attack Surface* no âmbito da Web, representa todos os possíveis pontos de entrada que uma entidade mal intencionada pode explorar para atacar uma aplicação ou sistema (Merkow, 2019).

Nas secções anteriores foram abordados os diferentes componentes das *Web applications* que podem estar sujeitos a falhas, contudo o conceito de *Attack Surface* vai para além disso, abrangendo toda a infraestrutura da aplicação, que de acordo com Merkow (2019), representa:

- Cada ponto em que o atacante pode interagir com a aplicação (todos os *inputs*, *inputs* ocultos, *cookies*, ou variáveis de *URL*);
- Cada funcionalidade que a aplicação dá resposta.

Podemos sumarizar que a *Attack Surface*, de um sistema, é o conjunto de formas que um adversário pode usar para entrar no sistema e potencialmente causar danos. Quanto maior for a *Attack Surface* mais inseguro é o sistema (Jajodia et al., 2012).

Esta realidade é especialmente relevante nas *Web Applications*. Devido à sua dimensão, nível de complexidade e exposição irão sempre apresentar vulnerabilidades de segurança, sendo necessário aceitar o potencial risco das vulnerabilidades futuras e atuais.

A dimensão da *Attack Surface* depende também do nível de acesso do atacante malicioso, que pode ser:

- **Externo** - restringe-se apenas as áreas da aplicação que não necessitam de autorização;
- **Interno** - o utilizador malicioso explora funcionalidades não autorizadas.

2.3.1.1 *Mapeamento da Attack Surface nas Web Applications*

O mapeamento da *Attack Surface*, também conhecido como *Reconnaissance*, é a primeira etapa do processo de testes de penetração sobre um sistema. São aplicadas técnicas de reconhecimento para enumerar e identificar os alvos vulneráveis que podem ser vitimas de futuros ataques. (Jajodia et al., 2012)

A *Attack Surface*, não se restringe apenas à arquitetura e ao *software* desenvolvido. O seu mapeamento facilmente se pode tornar num processo exaustivo. Este processo na vertente das *Web Applications*, segundo Merkow (2019), engloba um variado número de técnicas, sendo estas:

- Rastrear cada página da aplicação (usando uma ferramenta automatizada);
- Identificar todas as funcionalidades disponíveis:
  - Seguir todos os links;
  - Preencher todos os formulários com dados válidos/inválidos e submetê-los;
- Procurar os pontos no qual o utilizador pode fornecer dados para a aplicação:
  - Pedidos *GET* de *query* com parâmetros de strings;
  - Pedidos *POST* gerados por formulários;
  - HTTP *headers*;
  - *Cookies*;
  - Parâmetros Ocultos.

A duração da fase de *Reconnaissance* sobre a aplicação está proporcionalmente dependente da dimensão da *Attack Surface* exposta, sendo aconselhado a pré-selecção do software essencial à aplicação, reduzindo desta forma o risco associado e potenciais vulnerabilidades.

2.4 AMEAÇAS, VULNERABILIDADES E RISCOS

Na publicação *NIST 800-39: Managing Information Security Risk* os sistemas de informação são definidos como um conjunto de recursos que recolhem, processam, mantém, usam, distribuem, ou eliminam informação. (JTFTI, 2011)

Estes sistemas, estão continuamente sujeitos a ameaças, eventos com drásticas repercussões que têm o potencial de afectar negativamente a organização, e que segundo Watts (2020) podem-se enquadrar nos seguintes tipos:

- Ameaças Naturais (inundações, terramotos, etc.);
- Ameaças Não Intencionais (erros humanos, etc.);
- Ameaças Intencionais (malware, funcionário mal-intencionado, etc.).

Os recursos que compõem um sistema de informação, também podem apresentar vulnerabilidades de segurança, pequenos defeitos nos sistemas, que podem ser explorados por uma entidade mal intencionada.

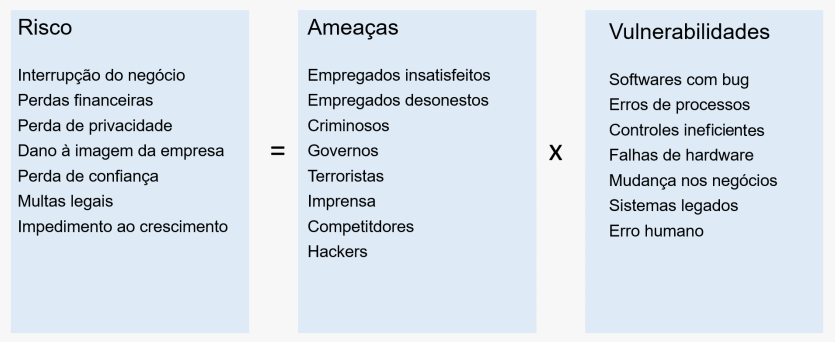


Figura 5: Formula do Risco - Adaptado (Programatic, 2015)

A exploração desses pequenos defeitos, em conjunto com uma ameaça, traduzem-se num potencial risco de informação, ao qual a organização está sujeita a sofrer de perdas significativas, como apresentado na **Figura 5**.

Segundo Watts (2020), um risco de informação pode ser descrito com uma perda ou prejuízo quando uma ameaça é explorada através da utilização de uma vulnerabilidade.

Com necessidade de minimizar o risco de informação, surgiram várias metodologias, *frameworks* e processos que afetam as várias condicionantes do risco, nomeadamente: ameaça, vulnerabilidade ou o próprio risco.

Nos seguintes sub-capítulos será abordada a *framework Common Weakness Enumeration*, utilizada para a identificação das vulnerabilidades. De seguida, devido

a natureza da solução proposta, será analisado o *NIST 500-269: Web Application Security Scanner* que contempla os requisitos funcionais de uma solução de *Web Scanning* juntamente com as vulnerabilidades cruciais que devem ser endereçadas. E por fim, analisado o *OWASP Top Ten - Web Application Security Risks*, que contém as vulnerabilidades de segurança das *Web Applications* mais cruciais.

2.4.1 Common Weakness Enumeration

A necessidade de existir uma linguagem universal para a identificação e descrição dos pontos fracos do sistema originou a criação da lista Common Weakness Enumeration (CWE). Nesta consta a segurança a nível do hardware e software, abrangendo assim: falhas, *crashes*, *bugs*, vulnerabilidades, erros, etc.

O CWE ao descrever as potenciais vulnerabilidades permite aos responsáveis pela segurança comunicá-las facilmente, e ao mesmo tempo auxiliar na validação da implementação de segurança. As *frameworks* Common Weakness Scoring System (CWSS) e Common Weakness Risk Analysis Framework (CWRAF) podem ser usadas em conjunção com CWE na classificação do nível de severidade das vulnerabilidade encontradas.

No âmbito das *Web Applications* o CWE revela-se de extrema importância por, permitir num sistema que se destaca pela complexidade do software, identificar as potenciais falhas de segurança para que possam ser endereçadas. Para tal, MITRE (2020) enumera as vulnerabilidades de carácter de software como pertencentes aos seguintes grupos:

- *buffer overflows*, formatações de string, etc;
- problemas de estrutura e validação;
- manipulação de caracteres especiais;
- erros nos canais de comunicação e caminhos;
- incorrecto tratamento de erros;
- erros de *user interface*;
- erros de *pathname traversal* e equivalentes;
- falhas na autenticação;
- erros de gestão de recursos;
- verificação de dados insuficiente;

- injecção de código;
- aleatoriedade e previsibilidade.

2.4.2 NIST 500-269: Web Application Security Scanner

O projeto Software Assurance Metrics and Tool Evaluation (SAMATE) desenvolvido pela National Institute of Standards and Technology (NIST) pretende auxiliar os engenheiros de software e clientes, ao providenciar métricas que atestam o grau de confiança das ferramentas usadas nos testes de qualidade de software. Para tal, descreve os requisitos funcionais dos softwares de garantia de qualidade, conhecidos por *Web Application Security Scanners*.

No âmbito trabalho proposto, a publicação *NIST 500-269: Web Application Security Scanner* revela-se de extrema importância, ao definir o que é um ferramenta de deteção de vulnerabilidades e de estipular os requisitos necessários para que esta tenha a capacidade de detetar as vulnerabilidades presentes nas *Web Applications*.

Segundo a NIST, um *scanner* é uma ferramenta automatizada que examina as potenciais vulnerabilidades de segurança presentes numa *Web Application*, podendo analisar erros no código.

De acordo com a publicação *NIST 500-269: Web Application Security Scanner*, um software de garantia de qualidade, para ser considerado um *Web Application Security Scanner*, tem de cumprir com os seguintes requisitos:

1. Identificar um conjunto de vulnerabilidades definidas pela NIST, como *Cross-Site Scripting*, *SQL Injection*, entre muitas outras que estão identificadas no **Anexo I**;
2. Informar de um possível ataque que use a vulnerabilidade encontrada;
3. Especificar o ataque descrevendo a localização do *script*, inputs e contexto;
4. Identificar a vulnerabilidade com um nome semanticamente equivalente aos conjunto de vulnerabilidades previamente referidos;
5. Ter a capacidade de autenticar na aplicação e manter o estado de autenticado;
6. Ter uma taxa de falsos positivos baixa.

As ferramentas de *Web Application Security Scanner* podem suportar funcionalidades opcionais que, se existirem, devem ser testadas, sendo estas:

1. **WA-RO-1:** Encontrar vulnerabilidades nos mecanismos de defesa definidos pela **NIST**, como *Token Validation*, *Typecasting*, entre muitas outras que estão listadas no **Anexo II**;
2. Ter a capacidade de suprimir as vulnerabilidades não aplicáveis à solução;
3. Ter a capacidade de providenciar pelo menos uma solução para a correcção da vulnerabilidade;
4. Produzir um relatório no formato eXtended Markup Language (XML);
5. Classificar a gravidade das vulnerabilidades identificadas.

2.4.3 *Top Ten - Web Application Security Risks*

A fundação Open Web Application Security Project (OWASP) é uma comunidade internacional, sem fins lucrativos formada por pessoas, organizações e instituições entusiastas pela segurança. Através da publicação regular de artigos, metodologias, estudos, documentos e ferramentas, a fundação OWASP pretende auxiliar as organizações no cumprimento e implementação da segurança.

A publicação *OWASP - Top Ten* define um risco de segurança como um potencial ponto de falha numa aplicação. Este risco, segundo o documento, pode-se enquadrar na lista *Top Ten*, que descreve, enumera e classifica os diversos riscos de segurança presentes nas *Web Applications* à data atual, sendo estes:

- **A1: Injection** - injecção de dados inválidos com intenção maliciosa;
- **A2: Broken Authentication** - exploração de falhas do mecanismo de autenticação;
- **A3: Sensitive Data Exposure** - acceso a informações confidenciais;
- **A4: XML External Entities (XXE)** - exploração de falhas do interpretador XML;
- **A5: Broken Access Control** - acesso indevido a conteúdos não protegidos;
- **A6: Security Misconfiguration** - exploração de configurações incorrectas, contas default, mensagem de erro expostas, etc;
- **A7: Cross-Site Scripting (XSS)** - injeção de código na aplicação com o intuito de atacar o utilizador;
- **A8: Insecure Deserialization** - injeção de dados inválidos com intenção maliciosa no processo de *desserialização*;

- **A9: Using Components with Known Vulnerabilities** - exploração de falhas nas dependências;
- **A10: Insufficient Logging Monitoring** - devem existir registos físicos sobre as operações afetas ao sistema.

2.5 SEGURANÇA NO *SOFTWARE DEVELOPMENT LIFE CYCLE*

Ao compreender as metodologias utilizadas pelos *hackers* na intrusão de aplicações, a **programação defensiva** tenta derivar as melhores práticas de segurança por forma a salvaguardar a arquitetura e lógica da aplicação.

Para tal, utiliza engenharia de software em paralelo com técnicas de *hacking* para reduzir a probabilidade de sucesso de um ataque, mitigar potenciais danos, e gerir danos ativos ou passados.

O Software Development Life Cycle (SDLC), na sua essência, descreve a sequência de etapas aplicadas ao longo do desenvolvimento de uma solução de software, desde o planeamento à distribuição do mesmo.

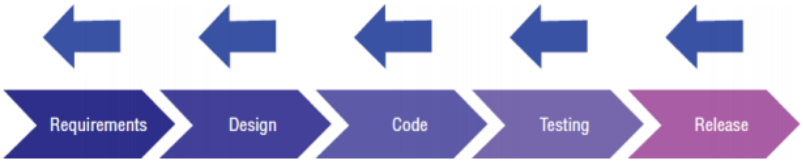


Figura 6: *Shifting Left Model* (Janca, 2020)

Para a implementação da segurança no SDLC é recomendado o modelo *Shifting Left*, apresentado na **Figura 6**, que defende que quanto mais cedo for aplicada a segurança no *pipeline* de desenvolvimento de software, menos propensa será a solução a falhas de segurança.

Segundo Janca (2020), este modelo ao endereçar as falhas do software desde o ínicio, impede que estas prossigam para as seguintes etapas de desenvolvimento, minimizando assim o esforço e possíveis custos de retificação.

No âmbito do pipeline de desenvolvimento de **Web Applications**, Hoffman (2020) recomenda que a implementação deste modelo deve ser acompanhado por um conjunto de medidas que devem ser aplicadas ao longo das várias fases do SDLC, sendo estas:

- **Arquitetura de Aplicação Segura** - aplicada na fase de levamento de requisitos e estendo-se ao longo da fase de design. Tem como objetivo identificar as variáveis, nomeadamente: local de armazenamento dos dados, dependências de terceiros, linguagem fonte, etc;
- **Revisões de Código para Segurança** - efetuado após a arquitetura estar bem consolidada. Tem como objetivo garantir a qualidade, reduzir o débito técnico e eliminar erros de programação ao incentivar o uso de boas práticas;
- ***Vulnerability Discovery*** - devido aos processos anteriores serem de caráter humano, é possível que existam falhas, pelo que é necessário que exista um reforço adicional que assegure que nenhuma vulnerabilidade passe despercebida;
- ***Vulnerability Management*** - após as vulnerabilidades serem encontradas, deve ser cuidadosamente planeado todo um processo de monitorização e análise de risco, que avalie e priorize a resolução das vulnerabilidades, dependendo do seu grau de criticidade.

2.5.1 *Security Automation*

Na fase de *Vulnerability Discovery* existem múltiplas técnicas que podem ser aplicadas para a deteção de vulnerabilidades que não foram previamente encontradas ao longo do SDLC.

O mecanismo de segurança *Security Automation* é uma desses técnicas, que segundo Hoffman (2020), apesar de apresentar algumas limitações na identificação de vulnerabilidades complexas, destaca-se pelo custo, eficácia e longevidade.

As limitações presentes nas soluções de *Security Automation* devem-se ao facto das ferramentas automáticas empregues não terem a capacidade de encontrar vulnerabilidades lógicas especificas da aplicação, e não cruzarem múltiplas vulnerabilidades simples numa mais complexa (Hoffman, 2020).

Nas seguintes sub-secções serão explorados em mais detalhe os diferentes tipos de análises que as ferramentas de *Security Automation* efetuam para descobrir as vulnerabilidades presentes nas *Web Application*.

2.5.1.1 *Static Application Security Testing*

As Static Application Security Testing (SAST) são ferramentas que pretendem atestar a segurança da aplicação através da metodologia *white-box*. Realizam uma

análise *white-box* estática, que tem acesso intrínseco ao design da aplicação, modelo de ameaças, e documentação.

A análise é estática porque, através da análise do código da aplicação, tenta detetar problemas de segurança e falhas de design software que podem variar desde código morto a violações das boas práticas.

A implementação da SAST usualmente emprega-se na fase de desenvolvimento como ferramenta auxiliar (*linter*) ao programador, ou diretamente aplicada no repositório onde o código da aplicação está alojado. (Hoffman, 2020)

Comparando com a análise estática manual, a análise SAST destaca-se pela velocidade de análise, baixo custo, capacidade de analisar grandes aplicações e maior eficiência na realização de tarefas repetitivas. (Merkow, 2019)

Apesar das vantagens das SAST, segundo Merkow (2019) estas apresentam uma grande taxa de falsos positivos pelo que requerem um esforço adicional e contínuo do seu tratamento. Têm dificuldades a avaliar falhas na lógica de negócio, e não conseguem verificar certo tipo de vulnerabilidades (fugas de informação, condições de corrida, etc...). De acordo com Hoffman (2020), as ferramentas SAST também têm dificuldade a avaliar linguagens *dynamically typed* como o *javascript*, que devido a sua natureza permite a alteração do tipo das variáveis a qualquer altura no ciclo de vida das aplicações.

2.5.1.2 *Dynamic Application Security Testing*

As ferramentas Dynamic Application Security Testing (DAST) pretendem simular um ataque do ponto de vista do atacante. Para tal, tentam encontrar as vulnerabilidades da aplicação através do uso de testes automatizados de penetração *black-box*.

Por estas ferramentas se basearem na metodologia *black-box* têm a capacidade de encontrar vulnerabilidades nas *Web Applications* comerciais e proprietárias, onde usualmente não é possível ter acesso ao código fonte para análise ou revisão. (Merkow, 2019)

De forma a simular as tentativas de ataque, estas enviam múltiplos *playloads* maliciosos para o servidor e analisam a resposta de forma a encontrar possíveis erros. Também, têm a capacidade de validar se o servidor está corretamente configurado.

Muitas destas DAST adicionalmente empregam um *proxy*, que ao intercetar todos os pedidos, permite diretamente atestar as validações realizadas pelo servidor, ao mesmo tempo que, aumentam o nível de controlo dos pedidos enviados e recebidos.

Por realizarem a análise dinâmica diretamente sobre as *Web Application*, podem apresentar um custo mais elevado e serem mais demoradas. Ao contrário das ferramentas SAST, são mais fáceis de manter, apresentam menor custo a longo prazo e têm uma maior taxa de acerto.

3

ESTADO DE ARTE

No capítulo anterior foi introduzido o processo de desenvolvimento de software e a importância da segurança no mesmo. Atualmente, a *VOID SOFTWARE, S.A.* aplica um processo de CI/CD através do *Jenkins*, pelo que se torna necessário compreender como este funciona.

Na **Secção** 3.1 (Jenkins) será introduzido *Jenkins* e o processo de desenvolvimento contínuo que esta ferramenta proporciona. De seguida, serão explorados os conceitos base e as suas formas de utilização.

Tendo em vista objetivo do estágio, na **Secção** 3.2 (Ferramentas de Web Security) serão realizados dois estudos de recolha e seleção das várias ferramentas de *Web Security* de forma a inserir-las no processo de integração contínuo.

3.1 JENKINS

Segundo Heller (2020), o *Jenkins* foi criado pela necessidade de automatizar parte do desenvolvimento de software. Ao endereçar este problema, revolucionou o SDLC tornando-o num processo de Integração Contínua (IC).

O *Jenkins*, para além do caráter *open-source*, destaca-se por suportar diferentes paradigmas e linguagens de programação; ser extremamente extensível, fácil de integrar e distribuir, entre muitas outras vantagens que o tornam líder do mercado. (Kulshrestha, 2020)

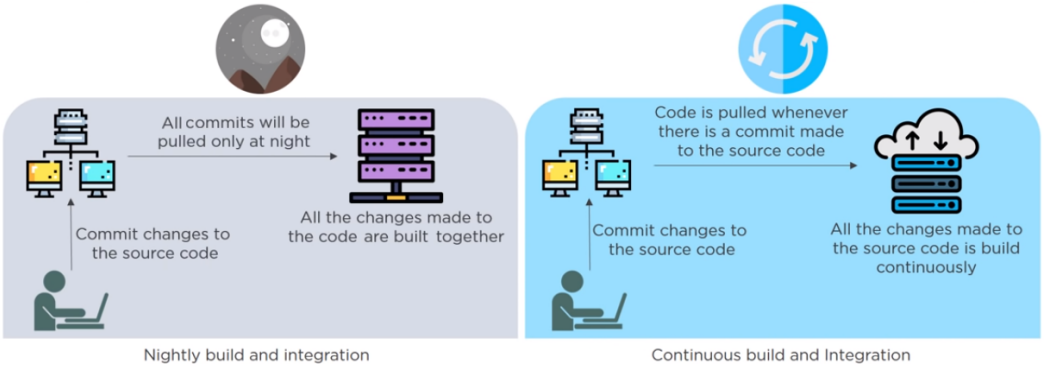


Figura 7: Desenvolvimento de software - Integração Contínua (Simplilearn, 2018)

Na **Figura 7**, é apresentado o processo antes e depois da integração do *Jenkins* no desenvolvimento de software.

No primeiro cenário onde o *Jenkins* não foi utilizado, as alterações realizadas ao longo do dia apenas são validadas e integradas durante o período da noite. Se existir alguma falha, todas estas alterações serão rejeitadas, tornando o desenvolvimento pouco produtivo.

Com o *Jenkins*, ao invés das alterações serem aplicadas todas de uma só vez, sempre que existe uma nova alteração, um *pipeline* é invocado e através deste a solução é validada e de seguida adicionada ao ambiente de produção. Caso existam problemas, apenas aquela alteração é rejeitada e um relatório com as falhas é gerado.

3.1.1 *Introdução ao pipeline*

Para automatizar o processo de SDLC o paradigma do *Jenkins* associa a cada projeto um *pipeline*. Um pipeline é constituído por um conjunto de *stages*, fases do projeto, que executam ações como demonstrado na seguinte **Figura 8**.

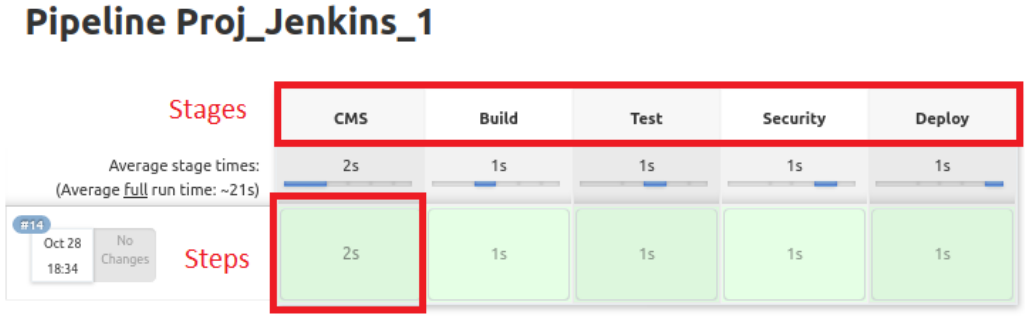


Figura 8: *Jenkins* - Exemplo de um *Pipeline*

Estas ações, executadas no decorrer de um *stage*, denominam-se por *steps* e podem realizar inúmeras operações como o levantamento do código fonte, análise de segurança, testes unitários, etc.

Baseando-se neste funcionamento foi proposto a adição de um componente de segurança, facilmente integrável e flexível que permite identificar as falhas presentes nas *Web Application* em pipeline.

3.1.2 *Scripted e Declarative pipeline*

Tendo em consideração a segurança e as boas práticas, torna-se relevante introduzir as diferentes sintaxes que um pipeline pode usar. Na sub-secção anterior foram introduzidos os diversos conceitos associados à pipeline. A obtenção do pipeline foi conseguido através de um *JenkinsFile*, que define a estrutura e as acções a serem realizadas.

Figura 9: *Jenkins* - *Scripted* e *Declarative* pipeline (McKenzie, 2020)

Na **Figura 9** são apresentado as sintaxes que podem ser usadas para definir um pipeline, sendo estas: *Scripted* ou *Declarative*.

Segundo McKenzie (2020), o *Jenkins*, no seu início, apenas suportava *scripted pipelines* para automatizar o SDLC. Por permitir a injeção de scripts, estes eram

usados invés das funcionalidades do *Jenkins*, aumentando assim a complexidade do *pipeline* e tornando-o mais difícil de ler e manter.

Para dar resposta a esta questão, surgiu o *Declarative pipeline*, que ao não permitir a injeção de código, obriga a exploração e utilização das funcionalidades disponibilizadas pelo *Jenkins*.

3.2 FERRAMENTAS DE WEB SECURITY

Na secção anterior foi estudada a ferramenta *Jenkins* que, devido a sua extensibilidade, permite integrar novas ferramentas no pipeline, inclusive as ferramentas de *Web Security*.

De forma a explorar e selecionar as ferramentas *open-source* e *on-premises applience* de *Web Security* mais apropriadas ao cenário da *VOID SOFTWARE, S.A.* foi necessário realizar um levantamento destas.

Então, as ferramentas foram divididas com base no tipo de análise efetuada: dinâmica (DAST) ou estática (SAST). Ao dividir estas ferramentas, que são fundamentalmente diferentes, foi possível no estudo efectuado, visualizá-las de forma perceptível e organizada; aplicar sobre estas critérios relevantes, permitindo assim a escolha das que seriam adotadas.

3.2.1 *Dynamic Application Security Testing*

Com base nos *benchmarks Vulnerability Scanning Tools* (OWASP, 2019) e *The Input Delivery Method Scanning Support of Web Application Vulnerability Scanners* (Chen, 2016) foi realizado um levantamento das várias ferramentas DAST que cumprissem com os requisitos previamente estabelecidos, *open-source* e *on-premise appliance*, sendo obtidas as seguintes ferramentas:

- **Wapiti**;
- **ZAP**;
- **W3af**;
- **Skipfish**;
- **Vega Vulnerability Scanner**
- **Nikto**;

- **Sqlmap**.

Após a fase de levantamento, foi efetuado um estudo comparativo para avaliar cada ferramenta, sendo o principal critério a capacidade de identificar as vulnerabilidades consideradas essenciais no *NIST 500-269: Web Application Security Scanner* e as mais frequentes no *OWASP - Top Ten*. Na realização deste, para validar esta capacidade foi utilizado o estudo *Evaluation of Web Application Vulnerability Scanners in Modern Pentest* (Chen, 2018) a informação em falta foi completada através da recolha individual das vulnerabilidades suportadas e da documentação disponibilizada por cada ferramenta.

Para além disso, tendo em conta a vertente de automatização e futura integração, foi necessário verificar a capacidade das ferramentas suportarem o seguinte conjunto de propriedades:

- *Automation* - capacidade de suportar automatização nativamente;
- *Depth* - ferramenta específica para um caso de uso em particular;
- *Cli* - mesmo sem suportar nativamente a automação permite facilmente automatizar;
- *OpenApi Support* - capacidade de entender definições de *Open API*;
- *Native Authenticaion* - capacidade de efectuar login no processo de análise;
- *Proxy* - capacidade de remeter pedidos para um servidor externo, melhorando significativamente o controlo e a qualidade da análise.

Na **Tabela 1**, "Estudo Comparativo - Ferramentas DAST", foi realizado um enquadramento das ferramentas recolhidas em relação à sua capacidade de dar resposta às vulnerabilidades e às propriedades previamente enumeradas.

Tabela 1: Estudo Comparativo - Ferramentas *DAST*

	Wapiti	**Zap**	**W3af**	**Skipfish**	**Vega**	**Nikto**	**Sqlmap**
Vulnerabilidades – NIST 500-269							
Cross-Site scripting (XSS)	✗	✗	✗	✗	✗		
SQL injection	✗	✗	✗	✗	✗	✗	✗
OS Command Injection	✗	✗	✗	✗	✗		

Continua na próxima página

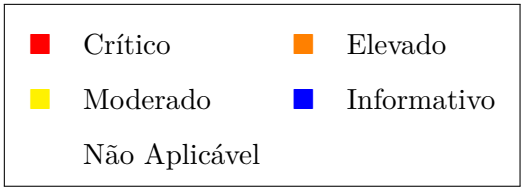
Tabela 1 – *Estudo Comparativo - Ferramentas DAST (Continuação)*

	Wapiti	**Zap**	**W3af**	**Skipfish**	**Vega**	**Nikto**	**Sqlmap**
Malicious File Inclusion		✗	✗		✗		
Insecure Direct Object Reference	✗	✗	✗	✗			
Information Leakage	✗	✗	✗	✗	✗	✗	✗
Improper Error Handling	✗	✗	✗	✗	✗		
Cross Site Request Forgery (CSRF)		✗	✗	✗			
HTTP Response Splitting (CRLF)	✗	✗	✗	✗			
Unrestricted URL Access	✗	✗	✗				
Insecure Communication			✗	✗			
XML Injection	✗	✗	✗	✗			
Weak Authentication and Session Management			✗	✗			
Session Fixation							
Vulnerabilities – OWASP 2017							
Injection	✗	✗	✗	✗	✗	✗	✗
Sensitive Data Exposure	✗	✗	✗	✗	✗		✗
Broken Access Control	✗	✗	✗	✗	✗		✗

Continua na próxima página

Tabela 1 – *Estudo Comparativo - Ferramentas DAST (Continuação)*

	Wapiti	**Zap**	**W3af**	**Skipfish**	**Vega**	**Nikto**	**Sqlmap**
Cross-Site scripting (XSS)		✗	✗		✗		
Insecure Deserialization							
Security Misconfiguration	✗	✗	✗	✗	✗	✗	
Broken Authentication (CSRF)		✗	✗	✗			
Vulnerable Dependencies		✗					
XML External Entities (XXE)	✗	✗	✗	✗			
XML Injection	✗	✗	✗	✗			
Insufficient Logging & Monitoring							
Properties							
Automation		✗			✗		
Depth						✗	✗
CLI	✗	✗	✗	✗		✗	✗
OpenApi Support		✗					
Native Authentication		✗	✗	✗			
Proxy		✗	✗		✗		
Accuracy	Alto	Excelente	Médio	Médio	Baixo	Alto	Alto
Last Release (Jun/2021)	Fev/20	Jan/20	Abr/15	Dez/12	Jun/16	Nov/20	Dez/20



Baseado no estudo efetuado e apresentado na **Tabela 1** podemos afirmar que o *ZAP* é atualmente o software *open-source* de análise dinâmica mais completo. Neste estudo, também se destacou a ferramenta *sqlmap*, que mesmo não tendo a capacidade de detetar as várias vulnerabilidades usuais dos DAST, apresenta um elevado grau de qualidade na deteção de vulnerabilidades do tipo *SQL injection*.

A análise dinâmica, em comparação com os outros tipos de análise, destaca-se por apresentar um baixo número de falsos negativos. Devido a esta particularidade e por não necessitar de conhecer o funcionamento interno da aplicação, metodologia *black-blox*, as boas práticas recomendam que sejam aplicadas múltiplas ferramentas.

Tendo isto em consideração, houve a necessidade de selecionar várias ferramentas. O *ZAP* e *sqlmap* foram as ferramentas que mais se destacaram e por isso adotadas. Como o *sqlmap* não permite realizar uma análise dinâmica completa pelo que foi preciso selecionar mais um ferramenta.

Das ferramentas restantes, o *Wapiti* e *W3af*, mesmo não tendo a qualidade de deteção do *ZAP*, destacaram-se pela sua cobertura na deteção de vulnerabilidades. De entre estas, o *Wapiti* foi adoptado pelas alterações constantes e atualizações que apresenta.

Na *VOID SOFTWARE, S.A.* este estudo serviu para selecionar as ferramentas DAST que pela sua qualidade e cobertura, destacaram-se na deteção de vulnerabilidades, sendo estas: *ZAP*, *sqlmap* e *Wapiti*.

3.2.2 *Static Application Security Testing*

Para a realização do levantamento das ferramentas de análise estática foi usado o *benchmark Source Code Analysis Tools* (OWASP, 2020b), que lista as ferramentas segundo a linguagem de programação a que se destinam. Após o cruzamento dessa listagem com os requisitos propostos, *open-source* e *on-premises appliance*, as ferramentas resultantes apresentaram-se compatíveis com uma das seguintes linguagens:

- *.NET*;

- *Python*;
- *PHP*;
- *Java*;
- *Ruby*;
- *C*;
- *Any* - ferramentas com capacidade de realizar análise estática independentemente da linguagem;
- *Miscellaneous* - ferramentas que não realizam análise estática diretamente.

Para complementar a informação destas ferramentas foi também realizado uma análise individual, através da consulta da documentação, segundo os seguintes critérios:

- *CI/CD* - integração nativa com ferramentas CI/CD;
- *Type* - breve descrição do tipo de ferramenta;
- *GUI* - suporta interface gráfica;
- *CLI* - suporta interface por linha de comandos, fácil de integrar numa ferramenta de CI/CD;
- *Freeware* - ferramenta livre, mas de código fechado;
- *Last Release* - última data de alteração, permite obter uma estimativa da capacidade de evolução e manutenção da ferramenta.

Na **Tabela 2**, "Levantamento - Ferramentas SAST", foi realizado um levantamento das ferramentas obtidas agrupadas segundo a linguagem de programação a que se destinam e enquadrando estas com os critérios previamente enumerados.

Tabela 2: Levantamento - Ferramentas *SAST*

Name	CI/CD	Type	GUI	CLI	Freeware	Last Release (Jun/2021)
.NET						
.NET Security Guard	✗	CLI		✗		Abr/21
Security Code Scan	✗	VSCodeExtension		✗		Mar/21

Continua na próxima página

Tabela 2 – *Levantamento - Ferramentas SAST (Continuação)*

Name	CI/CD	Type	GUI	CLI	Freeware	Last Release (Jun/2021)
Python						
Bandit		CLI		✗	✗	Fev/21
Pyre/Pysa		Linter			✗	Abr/21
PHP						
OWASP WAP		CLI		✗		Out/20
Pixy		IDE Plugin				Jul/21
Progpilot		CLI/IDE Plugin	✗	✗		Mar/21
OWASP ASST		CLI		✗		Fev/21
phpcs-security-audit		CLI		✗		Abr/20
Java						
Deep Dive		Standalone	✗		✗	NA
OWASP FindSecBugs		IDE Plugin				Fev/21
Sink Tank		Standalone	✗		✗	NA
Ruby						
Brakeman		CLI		✗		Mar/21
Dawnscanner		CLI		✗		Mar/21
C						
Flawfinder		CLI		✗		Mar/21
ANY						

Continua na próxima página

Tabela 2 – *Levantamento - Ferramentas SAST (Continuação)*

Name	**CI/CD**	**Type**	**GUI**	**CLI**	**Freeware**	**Last Release (Jun/2021)**
Agnitio		Standalone	✗			Mai/15
Graudit		CLI		✗		Jan/21
Horusec	✗	CLI		✗		Mar/21
HuskyCI	✗	CI/CD pipeline				Mar/21
Insider CLI	✗	CLI		✗		Jan/21
MobSF		Mobile Emulator		✗		Mar/21
PMD		CLI		✗		Abr/21
ShiftLeft Scan	✗	CLI		✗		Mar/21
SonarQube	✗	CI/CD pipeline	✗	✗		Abr/21
VisualCodeGrepper		Standalone	✗	✗		Out/19
Miscellaneous						
APIsecurity.io		Web Page				NA
GolangCI-Lint	✗	CLI		✗		Abr/21
Google CodeSearchDiggity		Standalone	✗		✗	NA
OpenAPI Editor		VSCodeExtension				Mar/21
nodejsscan		CLI/GUI	✗	✗		Mar/21

Na empresa a lógica de negócio das *Web Applications* desenvolvidas são essencialmente realizadas em *Java* pelo que para realizar uma análise mais aprimorada foram seleccionados os grupos de ferramentas classificados como *Java* e *Any*.

Da lista resultante, não só foram excluídas todas as ferramentas *Standalone* e *IDE Plugin* por não permitirem integração de forma direta ou indireta mas também as ferramentas *Freeware* por não cumprirem com o requisito de caráter *open-source*.

Assim sendo, após a aplicação destes critérios foi obtido a seguinte lista de ferramentas:

- *Graudit*;

- *Horusec*;
- *HuskyCI*;
- *Insider CLI*;
- *PMD*;
- *ShiftLeft Scan*;
- *SonarQube*.

Através de uma análise individualizada, a ferramenta *SonarQube* destacou-se pela cobertura de diversas linguagens e flexibilidade de integração e instalação. No entanto, o *SonarQube* já se encontra em uso na empresa e por isso em conjunto com a *VOID SOFTWARE, S.A.* optou-se por explorar e analisar outras ferramentas.

Em relação às ferramentas *Horusec*, *HuskyCI*, *ShiftLeft Scan* estas foram excluídas por dependerem da tecnologia *docker*, que ao ser empregue no *Jenkins* requer um esforço adicional de recursos e um acréscimo do grau de complexidade.

Por exclusão de partes, obteve-se o seguinte conjunto de ferramentas:

- *Graudit*;
- *Insider CLI*;
- *PMD*.

Para a seleção da ferramenta a ser adoptada foi utilizado uma abordagem mais prática, na qual se usou a aplicação vulnerável desenvolvida em *Java*, *Vulnado (Intentionally Vulnerable Java Application)* (ScaleSec, 2020), para avaliar a qualidade e os resultados obtidos da análise de cada ferramenta.

No caso da análise efetuada pela ferramenta *Graudit*, apresentada na **Figura 10**, esta apenas realça o código que pode conter falhas de segurança, mas não tem atualmente capacidade de gerar relatórios nem contextualizar as falhas de segurança encontradas.

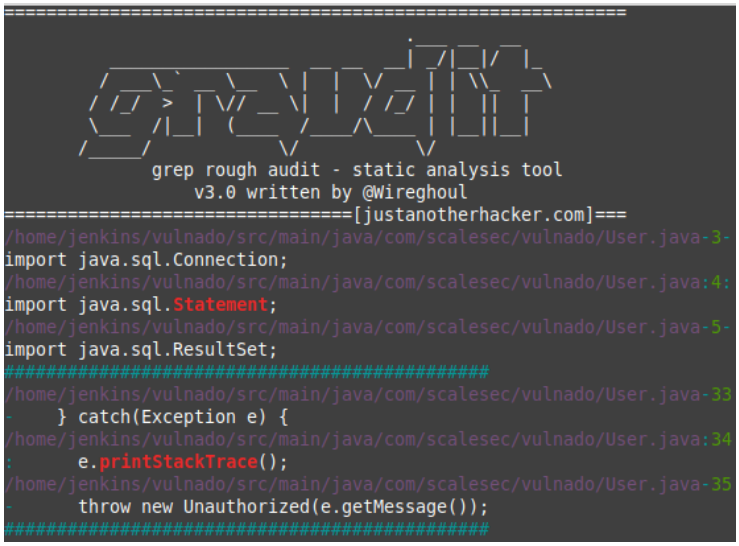


Figura 10: *Graudit* - Resultado da análise estática efetuada sobre Vulnado

A análise da ferramenta *PMD* (relatório disponível no **Apêndice A**) identifica principalmente as falhas de design, incumprimento das boas práticas e a falta de documentação presentes na solução *Vulnado*.

Ao contrário da ferramenta anterior, o *Insider CLI* (relatório disponível no **Apêndice B**) foca-se exclusivamente em encontrar as falhas de segurança presentes na solução, descrevendo-as e classificado-as segundo o CWE correspondente.

Após a análise destas ferramentas, o *Insider CLI* foi selecionado como ferramenta a ser adoptada por ter a capacidade de encontrar falhas de segurança, requerido como essencial para a empresa.

4

DESENVOLVIMENTO

Para responder aos requisitos propostos neste estágio foi necessário desenvolver uma solução de *Web Security*. Neste capítulo, pretende-se explorar todas as fases de desenvolvimento deste software.

Na **Secção 4.1** (Metodologia), será introduzida a metodologia de desenvolvimento empregue e as ferramentas usadas na sua aplicação.

Com a recolha dos *Web Scanners*, previamente selecionados, na **Secção 4.2** (Implementação de ferramentas de Web Security), proceder-se-á à implementação destes, a fim de compreender o seu funcionamento, as suas falhas e resoluções para as mesmas.

Na **Secção 4.3** (Web Security Application Project), será introduzida a ferramenta desenvolvida, a sua arquitetura, estrutura, utilização e integração.

Para validar o normal funcionamento e a utilização do WSAP, na **Secção 4.4** (Testes e Resultados), serão analisados os resultados obtidos com a integração do software nos cenários da empresa.

4.1 METODOLOGIA

Após a definição dos objetivos do trabalho a desenvolver no estágio, procedeu-se à realização do planeamento do trabalho. A *VOID SOFTWARE, S.A.* mostrou-se bastante flexível em aceitar o processo de desenvolvimento, tecnologias e implementação propostos.

Das várias metodologias ágeis, *Kanban* destacou-se pela flexibilidade e facilidade na visualização do trabalho em desenvolvimento. A metodologia *Kanban*, inventada por *Taiichi Ohno*, divide num quadro as tarefas a serem realizadas com base no seu estado de desenvolvimento.

Para aplicar esta metodologia e dar resposta às constantes sugestões foi usado, de forma autónoma, a ferramenta de gestão de projeto *Asana*, demonstrada na **Figura 11**.

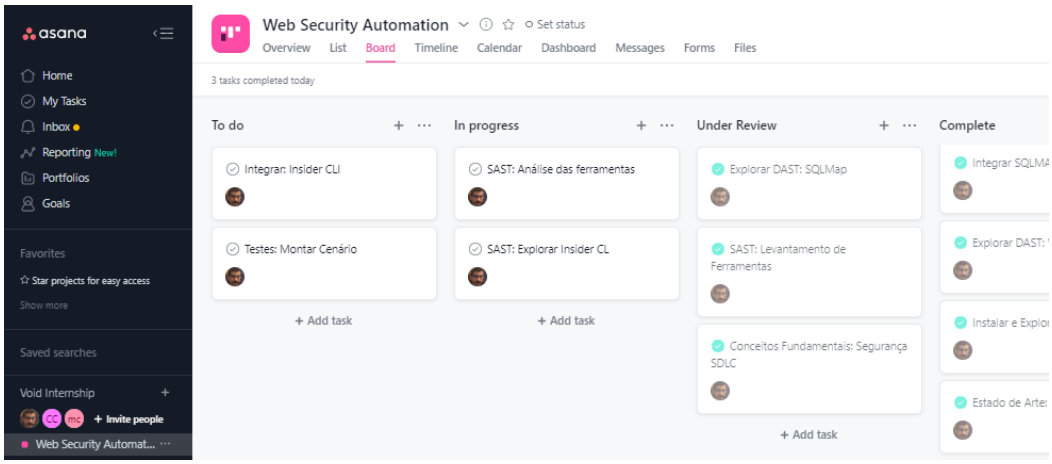


Figura 11: *Asana* - Seguimento da metodologia *Kanban*

4.2 IMPLEMENTAÇÃO DE FERRAMENTAS DE *WEB SECURITY*

Devido à complexidade e às particularidades das ferramentas de análise de vulnerabilidades, houve a necessidade compreender o seu funcionamento e realizar a implementação das mesmas.

Com base no levantamento de ferramentas efetuado nas **Secções** 3.2.1 e 3.2.2, foi possível de um conjunto de ferramentas de análise de vulnerabilidades selecionar as que mais se destacaram segundo os critérios propostos, sendo estas:

- Ferramentas de Análise Dinâmica:
  1. *ZAP*;
  2. *SQLMap*;
  3. *Wapiti*;
- Ferramentas de Análise Estática:
  1. *InsiderCLI*.

Em relação à ferramenta *ZAP*, na **Secção** 4.2.1, "Zed Attack Proxy", serão introduzidas as etapas que efetua durante a sua análise dinâmica e algumas das suas funcionalidades adicionais. Após isto, serão exploradas as opções de implementação, inclusive a adopção da integração com a ferramenta *SQLMap* para complementar a análise.

Na **Secção** 4.2.2 (Wapiti) será explorado o funcionamento não convencional do *Wapiti* e o que o diferencia entre as outras ferramentas DAST. De seguida,

procede-se à implementação desta ferramenta e apresentam-se as soluções que foram tomadas para enderça algumas das suas limitações.

Por fim, na secção **Secção 4.2.3** (InsiderCLI) será realizado uma pequena introdução à ferramenta *InsiderCLI* e demonstrado o seu funcionamento.

4.2.1 *Zed Attack Proxy*

Como previamente referido na **Secção 3.2** (Ferramentas de Web Security), o *ZAP* enquadra-se nas soluções do tipo DAST. De forma a integrar a ferramenta *ZAP* de acordo com a arquitetura que será apresentada na **Secção 4.3.1** (Arquitetura WSAP), houve a necessidade de explorar e compreender como esta realiza a análise dinâmica. Através da exploração, documentação e conteúdos didáticos disponibilizados foi possível definir o seguinte conjunto de etapas:

1. *Configuration* - propriedades e configurações relativas ao proxy usado pelo scanner;
2. *Profiling* - estruturação das informações básicas da *Web Application*.
3. *Crawling* - levantamento da *Attack Surface* exposta pela *Web Application* a ser analisada;
4. *Attacking* - realização de ações maliciosas sobre os *endpoints* previamente identificados;
5. *Authentication* - análise da *Web Application* na perspetiva de um utilizador:
    a) *Crawling* - levantamento da *Attack Surface* exposta pela *Web Application* segundo os privilégios do utilizador;
    b) *Attacking* - realização de acções maliciosas sobre os *endpoints* previamente identificados segundo os privilégios do utilizador.

Como mencionado na **Secção 3.2** (Ferramentas de *Web Security*), o *ZAP* para realizar a sua análise recorre a um *proxy* que, ao dar a possibilidade de intercetar os pedidos e respostas em trânsito, permite encontrar vulnerabilidades que doutra forma não seriam encontradas.

Na etapa *Configuration* torna-se necessário fornecer as propriedades do *proxy*. Enquanto que, na etapa *Profiling* é definido o perfil da aplicação a ser analisada. Ambas as etapas requerem que sejam fornecidas informações necessárias para a realização da análise, sendo estas:

- Scanner:
  - *Proxy IP*, o endereço IP do *proxy* a ser usado pelo scanner;
  - *Proxy Port*, a porta do *proxy* a ser usada pelo scanner.
- Profile:
  - Nome, nome identificativo da análise;
  - *Url base*, o endereço base da *Web Application* a ser analisada;
  - *Include URL*, força a adição de um *endpoint* ao perfil para ser analisado, como robots.txt;
  - *Exclude URL*, remove um *endpoint* da análise, como o logout.

Na seguinte etapa, *Crawling*, o *ZAP* necessita que lhe sejam fornecidas as informações relativas ao método de navegação usado para recolher os *endpoints* que descrevem a *Attack Surface* da *Web Application*. Disponibiliza, assim, nativamente o seguinte conjunto de *crawlers*:

- *Tradicional Crawler*, descreve a *Web Application* através do processamento dos links presentes no código HTML, recomendado para aplicações MPA;
- *Ajax Crawler*, descreve os *endpoints* da *Web Application* através do estudo da interação do utilizador com o sistema, recomendado para aplicações SPA;

Para completar esta recolha, o *ZAP* permite através do *plugin* "OpenAPI Automation Framework Support"que, lhe seja fornecido uma lista com os *endpoints* da aplicação segundo o *standard Open API*, tido como requisito necessário para o cenário em questão.

Logo após a conclusão do levantamento da *Attack Surfice*, prossegue-se a fase de *Attacking*. Nesta fase, a ferramenta *ZAP* realiza sobre a *Web Application* um conjunto de análises de caráter dinâmico com o intuito de encontrar e classificar as vulnerabilidades presentes na solução. Atualmente, o *ZAP* disponibiliza dois modos de análise dinâmica, sendo estes:

- Modo Passivo, efetuado automaticamente e em simultâneo no processo de recolha dos *endpoints* no qual identifica as falhas presentes nos pedidos e respostas recebidas, não sendo desta forma intrusivo;
- Modo Ativo, através do uso de vetores de ataque aplica-os sobre a aplicação de forma a encontrar vulnerabilidades, sendo desta forma intrusivo.

Após a fase de *Crawling*, segue-se a fase de *Authentication* onde é necessário que a ferramenta *ZAP* autentique-se na aplicação de forma automatizada. Atualmente, ZAP providencia o seguinte conjunto de métodos de autenticação:

- Form-based authentication
- Script-based authentication
- JSON-based authentication
- HTTP based authentication

Na *VOID SOFTWARE, S.A.* onde a autenticação é realizada apenas parcialmente em JavaScript Object Notation (JSON), o método *JSON-based authentication* não é suficiente, pelo que, para solucionar este problema foi desenvolvido um *script HTTPSender*, disponível no **Apêndice C**, que permite no processo de autenticação recolher o *token* do *header* e de seguida injeta-lo nos subsequentes pedidos.

Para a integração do *Sqlmap* no *ZAP* foi usado o *plugin "Advanced SQLInjection Scanner"*, que é executado sempre que o *ZAP* realiza um ataque ativo, permitindo desta forma, complementar a análise dinâmica efetuada pelo *ZAP* com uma análise de SQL mais exaustiva. Ao utilizar esta integração, o relatório gerado pela ferramenta *ZAP* já inclui a análise do *Sqlmap*, por isso não há necessidade de a realizar manualmente.

Listagem 1: *ZAP* - Relatório de Vulnerabilidades

```
[
  {
    "sourceid": "3",
    "other": "",
    "method": "GET",
    "evidence": "",
    "pluginId": "10020",
    "cweid": "16",
    "confidence": "Medium",
    "wascid": "15",
    "description": "X-Frame-Options header is not included in the HTTP response
    ↪  to protect against 'ClickJacking' attacks.",
    "messageId": "47706",
    "url": "https://test-vulnado.pt/static/js/zqxjETNZZBWZlc.html",
    "reference": "https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Fr
    ↪  ame-Options",
    "solution": "Most modern Web browsers support the X-Frame-Options HTTP
    ↪  header. Ensure it's set on all web pages returned by your site...",
    "alert": "X-Frame-Options Header Not Set",
    "param": "X-Frame-Options",
    "attack": "",
    "name": "X-Frame-Options Header Not Set",
```

```
    "risk": "Medium",
    "id": "5471",
    "alertRef": "10020"
  },
  ...
]
```

Na **Listagem 1**, "*ZAP* - Relatório de Vulnerabilidades", é demonstrado um pequeno excerto do relatório gerado pelo *ZAP*, através do uso dos diversas *plugins*. O relatório identifica as vulnerabilidades presentes na solução analisada e atribui um nível de criticidade às mesmas. Para além disso, apresenta um caso de uso para replicar o ataque e sugere uma possível resolução para o mesmo.

4.2.2 *Wapiti*

A ferramenta *Wapiti*, como estudado na **Secção 3.2** (Ferramentas de *Web Security*), também realiza uma análise DAST. Ao contrário da maioria das ferramentas DAST não recorre nativamente a um *proxy*. Em vez disso, funciona como um *fuzzer* ao injetar vetores de ataque sobre a aplicação a ser analisada.

Ao não utilizar nativamente um *proxy*, *Wapiti* encontra-se limitado quer na recolha dos *endpoints* que constituem a *Attack Surface* da aplicação, quer na própria deteção de possíveis vulnerabilidades presentes.

Para mitigar este problema, *Wapiti* fornece a possibilidade de redirecionar todas as ações realizadas para um *proxy* externo. Esta funcionalidade, ao ser usada, permite por um lado, possibilitar ao *ZAP* encontrar novas vulnerabilidades através de novos vetores de ataque e por outro, permite aprimorar os resultados obtidos com a análise do *Wapiti*.

De forma a dar início à análise dinâmica, *Wapiti* procede à recolha dos *endpoint* da *Web Application* através do uso de um *Tradicional Crawler*. Ao empregar este tipo de *crawler*, *Wapiti* não tem a capacidade de reconstruir a *Attack Surface* das *Web Applications* HPA e SPA.

Devido a esta limitação e à necessidade de *Wapiti* analisar aplicações SPA foi usado a recolha dos *endpoints* obtidos pela ferramenta anterior. Assim sendo, através do *ZAP* foi gerado um ficheiro com a lista dos *endpoints* identificados e fornecido ao *Wapiti* para que possa realizar a sua análise em aplicações HPA e SPA

Após a recolha dos *endpoits* o *Wapiti* realiza a fase de ataque. A ferramenta aplica um conjunto de módulos (timesql, shellshock...) sobre a aplicação de forma a encontrar potenciais vulnerabilidades. A lista dos módulos está disponível no **Anexo D**.

De seguida, prossegue-se à fase de autenticação. *Wapiti* disponibiliza nativamente suporte para alguns métodos de autenticação mas, devido a especificidade do processo de autenticação das aplicações desenvolvidas pela *VOID SOFTWARE, S.A.*, não se torna exequível aplicá-los. Tirando proveito do redirecionamento de pedidos para o *proxy* é possível trespassar a responsabilidade de autenticar todos as ações realizadas durante a análise do *Wapiti* para o *ZAP* e desta forma garantir a autenticação.

Listagem 2: *Wapiti* - Relatório de Vulnerabilidades


```
{
  "classifications": {
        "Command execution": {
      "desc": "This attack consists in executing system commands on the server.
      ↪  The attacker tries to inject this commands in the request parameters.",
      "sol": "Prefer working without user input when using file system calls.",
      "ref": {
        "OWASP: Command Injection":
        ↪  "https://owasp.org/www-community/attacks/Command_Injection",
        "CWE-78: Improper Neutralization of Special Elements used in an OS
        ↪  Command ('OS Command Injection')":
        ↪  "https://cwe.mitre.org/data/definitions/78.html"
      }
    },
        ...
  },
  "vulnerabilities": {
    "Content Security Policy Configuration": [
      {
        "method": "GET",
        "path": "/",
        "info": "CSP is not set",
        "level": 1,
        "parameter": "",
        "http_request": "GET / HTTP/1.1\nHost: test-vulnado.pt",
        "curl_command": "curl \"https://test-vulnado.pt/\""
      }
    ],
    ...
  },
  "infos": {
    "target": "https://test-vulnado.pt/",
    "date": "Mon, 13 Sep 2021 13:36:42 +0000",
    "version": "Wapiti 3.0.5",
    "scope": "folder"
```

```
32    }
33 }
```

Na **Listagem 2**, "*Wapiti* - Relatório de Vulnerabilidades", é apresentado o relatório gerado pelo *Wapiti* após a conclusão da sua análise. Na primeira parte deste, o *Wapiti* enumera e descreve todos módulos usados. De seguida, através da aplicação de cada módulo, as vulnerabilidades são identificadas e classificadas, entre zero e quatro, segundo o seu grau de criticidade. Por fim, são apresentadas algumas informações relativas à execução da análise.

4.2.3 *InsiderCLI*

Baseado no estudo e selecção de ferramentas na **Secção 3.2** (Ferramentas de Web Security), a ferramenta *InsiderCLI* enquadra-se no conjunto de soluções de análise de vulnerabilidades SAST.

Para realizar a análise estática, esta ferramenta apenas necessita de, à priori, conhecer a linguagem de desenvolvimento da *Web Application* que, no cenário em questão, é o *Java* e a localização do código fonte da aplicação, como demonstrado na seguinte **Figura 12**.

```
marquez@marquez-VirtualBox:/home/jenkins$ insider -tech java -target vulnado/
[insider] 2021/07/14 16:32:06 Starting analysis
[insider] 2021/07/14 16:32:06 Analysing Java dependencies
[insider] 2021/07/14 16:32:06 Starting source code analysis
[insider] 2021/07/14 16:32:06 Analysing files on directory vulnado/
[insider] 2021/07/14 16:32:06 Load 37 rules to file .mvn/wrapper/maven-wrapper.jar
[insider] 2021/07/14 16:32:06 Load 37 rules to file .mvn/wrapper/maven-wrapper.properties
[insider] 2021/07/14 16:32:06 Load 37 rules to file Dockerfile
[insider] 2021/07/14 16:32:06 Load 37 rules to file LICENSE
[insider] 2021/07/14 16:32:06 Load 37 rules to file README.md
[insider] 2021/07/14 16:32:06 Load 37 rules to file client/Dockerfile
[insider] 2021/07/14 16:32:06 Load 37 rules to file client/css/main.css
[insider] 2021/07/14 16:32:06 Load 37 rules to file client/images/doggo.jpg
[insider] 2021/07/14 16:32:06 Load 37 rules to file client/images/signout-hover.png
[insider] 2021/07/14 16:32:06 Load 37 rules to file client/images/signout.png
```

Figura 12: *InsiderCLI* - Exemplo de análise sobre Vulnado

Após concluir a análise, a ferramenta *InsiderCLI* gera um relatório em JSON e *HMTL* com as vulnerabilidades encontradas na solução.

Listagem 3: *InsiderCLI* - Relatório de Vulnerabilidades

```
1 {
2  "vulnerabilities": [
3   {
4    "cvss": 3.2,
```

```
    "cwe": "CWE-532",
    "line": 10,
    "class": "Cowsay.java (10:5)",
    "vul_id": "752ec8bb9b3b233b30dfcd426b152d55",
    "method": "System.out.println(cmd);",
    "column": 5,
    "description": "The App logs information. Sensitive information should not be
    ↪  logged.",
    "classMessage": "main/java/com/scalesec/vulnado/Cowsay.java (10:5)"
   },
   ...
  ],
  "none": 0,
  "low": 18,
  "medium": 0,
  "high": 2,
  "critical": 0,
  "total": 20,
  "sast": {
   "averageCvss": 7.4,
   "securityScore": 26,
   "size": "15680 Bytes",
   "numberOfLines": 499
  }
}
```


Na **Listagem 3**, "*InsiderCLI* - Relatório de Vulnerabilidades", é apresentado no formato JSON um pequeno excerto do relatório gerado pela ferramenta após concluir a sua análise. Neste relatório são identificadas as vulnerabilidades encontradas e caracterizadas segundo o CWE correspondente. Após a listagem de todas as vulnerabilidades, a ferramenta *InsiderCLI* classifica a solução analisada segundo o seu grau de segurança.

4.3 WEB SECURITY APPLICATION PROJECT

Com base nos requisitos estipulados no **Capítulo 1** (Introdução), através deste estágio, é pretendido que seja explorado e adicionado uma componente de *Web Security*, que possibilite à *VOID SOFTWARE, S.A.* identificar o mais rapidamente e antecipadamente possível as falhas de segurança presentes nas suas soluções.

Foi proposto o desenvolvimento de uma solução que aborda ambas as vertentes DAST e SAST e permite ser integrada no processo CI/CD; ser fácil de usar e capaz de ser automatizada. Para responder a estas necessidades foi desenvolvida a solução

WSAP que, agrega as diversas ferramentas previamente exploradas e aplica-as sobre as soluções de forma a encontrar eventuais vulnerabilidades.

Na **Secção 4.3.1**, "Arquitetura", será abordado a forma como a ferramenta WSAP é integrada na ferramenta Jenkins e exemplificado como a sua integração pode ser alcançada via linha de comandos.

De seguida, na **Secção 4.3.2**, "Estrutura", será realizado uma introdução à estrutura da ferramenta WSAP, onde é apresentado a solução base, a estrutura dos relatórios gerados e evidenciado os ficheiros de relevância para o entendimento da solução.

A **Secção 4.3.3** (Utilização) pretende introduzir a interface Command Line Interface (CLI) do WSAP ao explorar em detalhe as opções que a ferramenta suporta.

O WSAP através do *plugin* publicado na plataforma *Jenkins* disponibiliza uma interface gráfica que pretende facilitar o uso da ferramenta. Todo o processo de publicação e utilização do *plugin* WSAP no Jenkins encontra-se documentado nas **Secções 4.3.4** (Publicação) e **4.3.5** (Integração).

4.3.1 *Arquitetura*

Tendo como base o trabalho previamente realizado na **Secção 4.2** (Implementação de ferramentas de Web Security) houve a necessidade de explorar uma arquitetura que permitisse integrar as várias ferramenta de análise de vulnerabilidades na plataforma Jenkins.

Dessa pesquisa um dos trabalhos que se destacou foi o *Automating Security Tests using OWASP ZAP and Jenkins* de Kelebek (2015) que ao separar a instância do Jenkins da ferramenta de análise dinâmica, permite facilmente integrar a ferramenta e ao mesmo tempo delegar o processo exaustivo de análise de vulnerabilidades para uma máquina apropriada para o efeito.

Baseado no trabalho de Kelebek (2015) na **Figura 13** é apresentado o esquema que retrata a arquitectura e o funcionamento da solução WSAP na plataforma do *Jenkins*.

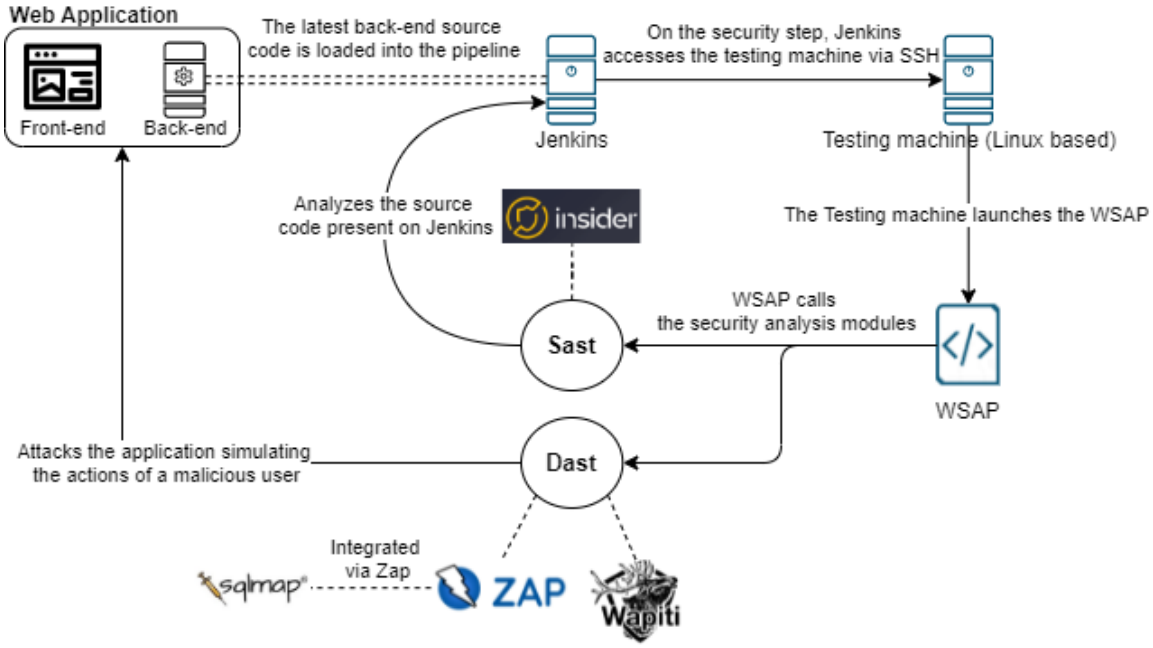


Figura 13: *WSAP* - Arquitetura - Adaptado de Kelebek (2015)

Com esta arquitectura, a implementação da solução WSAP é composta pelos seguintes componentes:

- *Web Application* - usualmente constituída por uma componente de interação com o utilizador (front-end) e outro componente de lógica de negócio (back-end);
- Jenkins - plataforma de automatização CI/CD que tem capacidade de interagir com todo o processo de desenvolvimento da *Web Application*, nomeadamente, acesso ao código fonte, aplicar testes de segurança, entre outras operações;
- Máquina de testes - responsável por receber os parâmetros da *Web Application* e sobre esta aplicar a solução WSAP.

Através da plataforma Jenkins, o WSAP pode ser diretamente integrado num pipeline ou periodicamente invocado como um *job* isolado.

Independentemente da opção escolhida, o *Jenkins* para poder dar inicio à análise de vulnerabilidades, invoca a solução WSAP na máquina de testes e prossegue a realização da análise. Na **Figura 14** é possível consultar a invocação da ferramenta WSAP via linha de comandos numa *Declarative Pipeline*.

Script

```
pipeline {
    agent any

    stages {
        stage('Security') {
            steps {
                sshagent (credentials: ['ab4d70b8-f7ca-42e8-bd10-56af090d3e8c']) {
                  sh '''ssh marquez@127.0.0.1 /usr/bin/python3 ~/Desktop/wsap/main.py --target.url http://test.com
                    --scanner.port 8010 --target ~/Desktop/dast/test.com --scan.mode FULL
                  '''
                }
            }
        }
    }
}
```

Figura 14: *Jenkins* - Invocação do WASP via *Declative Pipeline*

No que respeita à análise realizada, o WSAP através das ferramentas de análise de vulnerabilidades que implementa consegue abranger ambas as vertentes SAST e DAST.

Para iniciar a vertente SAST, o WSAP necessita que lhe seja fornecido acesso ao código fonte da aplicação para poder aplicar a ferramenta *InsiderCLI*.

Enquanto que, para o módulo DAST o WSAP automaticamente emprega as ferramentas *Wapiti*, *Zap* e *Sqlmap* integrado no *ZAP* para realizar a análise e obter as vulnerabilidades.

4.3.2 *Estrutura*

Para o desenvolvimento da solução WSAP foi optada a linguagem *Python* pela sua facilidade de desenvolvimento e manutenção e por ser usualmente usada em problemas de integração e automatização.

Em relação à estrutura da solução WSAP, apresentada na **Figura 15**, as várias ferramentas de análise de vulnerabilidades encontram-se divididas pelos seus correspondentes módulos, o que permite facilmente adicionar e integrar novas ferramentas.

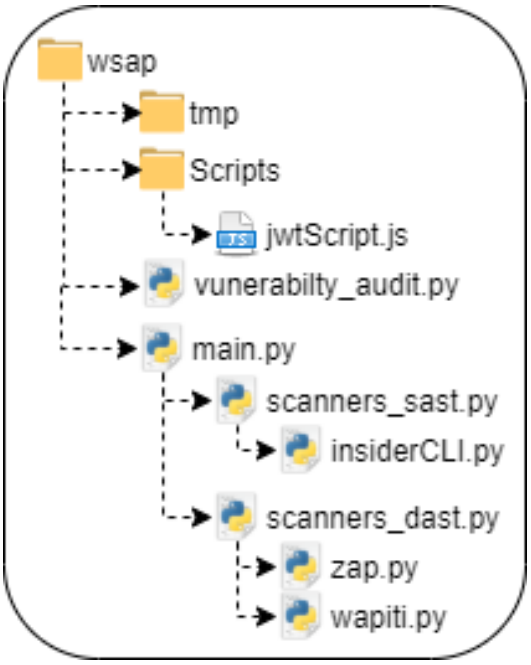


Figura 15: *WSAP* - Estrutura base

O WSAP para realizar a análise SAST faz uso do módulo *ScannersSast* que, por sua vez, executa a ferramenta atualmente implementada *InsiderCLI*. No caso da análise dinâmica, o WSAP faz uso do módulo *ScannersDast*, no qual implementa as ferramentas *ZAP* e *Wapiti*.

No fim, o WSAP executa o módulo *VulnerabilityAudit* para gerar uma compilação com todos os relatórios obtidos pelas várias ferramentas.

Como previamente apresentado na **Figura 15**, o WSAP também é constituído por pastas, sendo estas:

- *scripts* - armazena os *scripts* auxiliares à aplicação;
- *tmp* - armazena os ficheiros temporários, relatórios de análise, etc.

Atualmente, na pasta *Scripts* apenas reside o ficheiro *jwtScript.js*, que ao ser injetado no *ZAP*, permite dar ao *scanner* a capacidade de efetuar o *login* nas *Web Applications* desenvolvidas.

Na pasta *tmp*, sempre que uma análise é iniciada uma pasta constituída pelo o nome e *timestamp* é gerada.

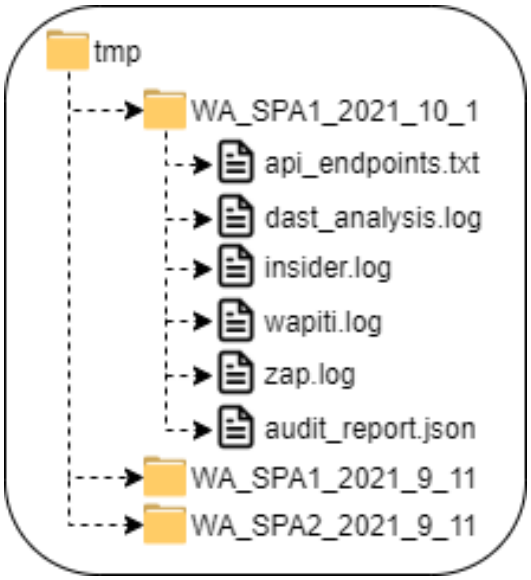


Figura 16: *WSAP* - Estrutura da análise efetuada

Na **Figura 16** é possível observar um conjunto de análises que foram realizadas. Na primeira análise são apresentados os componentes principais que a constituem, sendo estes:

- *api_endpoints.txt* - lista de todos os *endpoints* identificados através da leitura da *OpenAPI*;
- *dast_analysis.log* - log de todo o processo de análise dinâmica;
- *insider.log* - log do processo que invoca a ferramenta *InsiderCLI*;
- *wapiti.log* - log do processo e os comandos invocados com a ferramenta *Wapiti*:
- *zap.log* - log do processo da ferramenta *ZAP*;
- *audit_report.json* - relatório final em formato JSON, que contém todas as vulnerabilidades encontradas pelas várias ferramentas.

4.3.3 *Utilização*

A interação com a ferramenta WSAP é realizada através de uma interface de linha de comando, que facilmente pode ser usada no *Jenkins*, como demonstrado na **Secção 4.3.1** (Arquitetura).

```
marquez@marquez-VirtualBox:~/Desktop/wsap$ python3 main.py -h
usage: Web Security Application Project(WSAP) [-h] [-dT TARGET.URL] [-sT TARGET]
                                              [-dIp SCANNER.IP] [-dPort SCANNER.PORT]
                                              [-dM {full,apionly,traditional,ajax}]
                                              [-dI INCLUDE.URL] [-dE EXCLUDE.URL]
                                              [--scan.apiUrl SCAN.APIURL]
                                              [--scan.apiDefinition SCAN.APIDEFINITION]
                                              [--login.url LOGIN.URL]
                                              [--login.request LOGIN.REQUEST]
                                              [--login.userField LOGIN.USERFIELD]
                                              [--login.passField LOGIN.PASSFIELD]
                                              [-u username password]
optional arguments:
  -h, --help            show this help message and exit

Required arguments:
  -dT TARGET.URL, --target.url TARGET.URL
                        The url base of the target web application
```

Figura 17: *WSAP* - Propriedades base

Para utilizar o WSAP é necessário fornecer o parâmetro *target.url* com o *URL* da aplicação a ser analisada (**Figura 17**). Através deste parâmetro, o WSAP irá gerar uma pasta para conter os ficheiros relativos à análise, como previamente demonstrado na **Secção 4.3.2** (Estrutura).

```
★SAST Scanner Properties★:
  -sT TARGET, --target TARGET
                        The target web application directory path
```

Figura 18: *WSAP* - Análise SAST

A análise estática do WSAP é conseguida através da invocação da opção *target* com a localização do código fonte, mais especificamente do código do *back-end* da *Web Application*, onde as operações da camada de negócio são realizadas (**Figura 18**).

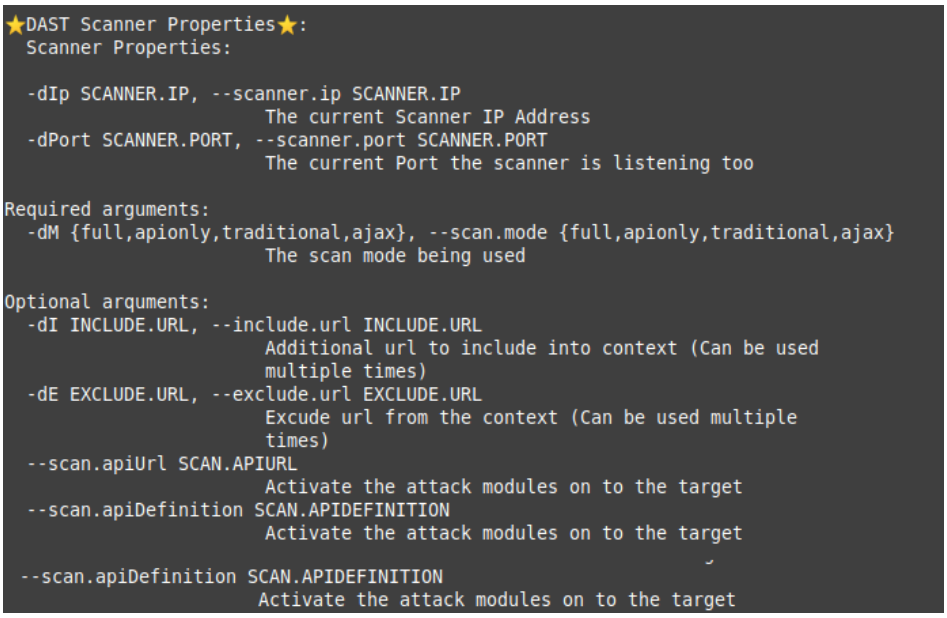


Figura 19: *WSAP* - Análise *DAST*

Em relação à análise dinâmica, apresentada na **Figura 19**, o WSAP necessita inicialmente conhecer os parâmetros relativos aos *scanners*, ou seja:

- *scanner IP* - endereço do *proxy* usado pela ferramenta *ZAP*;
- *port* - porto em que o *proxy* usado pela ferramenta *ZAP* se encontra;

De seguida, o WSAP necessita de compreender o cenário em que a *Web Application* se encontra inserida e as propriedades da mesma. O WSAP para realizar o levantamento de *endpoints* disponibiliza a opção *scan.mode* com as seguintes opções:

- *TRADITIONAL* - levantamento dos *endpoints* através do processamento dos links presentes no código HTML;
- *AJAX* - recolha de endpoints segundo a interação do utilizador com a *Web Application*;
- *API_ONLY* - levantamento dos *endpoints* segundo o formato *OpenAPI*;
- *FULL* - engloba todos as opções anteriores;

Para afinar esta recolha, o WSAP disponibiliza os parâmetros opcionais *include.url* e *exclude.url*, que permitem adicionar ou remover *endpoints*.

Caso o método de *scanning* seleccionado anteriormente for *FULL* ou *API_ONLY* torna-se necessário descrever a *API* da *Web Application* em análise. O *scan* deve ser complementado com as seguintes opções:

- *scan.apiUrl* - atual endereço da *API*;

- *scan.apiDefinition* - localização da definição da API da aplicação no formato standard *OpenAPI*, suporta os formatos *URI* e *URL*.

```
Login properties:
  --login.url LOGIN.URL
                        The login url necessary to successfully log into the application
  --login.request LOGIN.REQUEST
                        A skeleton of the JSON Request sent to perform a successfull login
  --login.userField LOGIN.USERFIELD
                        The Username key used on the JSON Request Login
  --login.passField LOGIN.PASSFIELD
                        The Password key used on the JSON Request Login
  -u username password, --login.user username password
                        The username:password of the wanted user (Can be used multiple times)
```

Figura 20: *WSAP* - Análise *DAST* - Autenticação

A análise dinâmica tem a particularidade de possibilitar a deteção de falhas de segurança segundo a perspetiva de um utilizador malicioso. O WSAP para testar a segurança em diferentes utilizadores e com diferentes níveis de acesso necessita de conhecer o processo de autenticação empregue.

Na **Figura 20** é possível consultar as propriedades que podem ser usadas para configurar o processo de autenticação. Através do parâmetro *login.url* torna-se possível fornecer ao WSAP o *endpoint* encarregue de realizar o *login* na aplicação a ser analisada. Para além disso, o WSAP necessita também de conhecer como a aplicação realiza o pedido de autenticação.

```
1 {
2   "username": "dummyuser",
3   "password": "1234567",
4   "country": "portugal",
5   "date_time": "17-12-2021"
6 }
```

Figura 21: *Login* - Exemplo de Pedido de Autenticação

Devido às diferentes implementações da autenticação nas *Web Applications*, um pedido de autenticação pode necessitar de parâmetros adicionais, como demonstrado na **Figura 21**. Para contemplar estes cenários, o WSAP necessita que lhe seja fornecido através do parâmetro *login.request*, o pedido de *login* usado pela aplicação, no formato JSON e *quoted*.

Após conhecer o pedido de *login*, o WSAP através dos parâmetros *login.userField* e *login.passField* permite mapear os campos que estão associados às credenciais. Dando uso à opção *login.user* torna-se possível fornecer as credenciais de cada utilizador e testar a segurança segunda a sua perspectiva.

4.3.4 *Publicação*

Esta secção pretende descrever todo o processo desde o desenvolvimento até a publicação do *plugin* no *Jenkins*. De acordo com a documentação, o desenvolvimento de *plugins* é realizado através da tecnologia *Maven*, sendo esta a tecnologia adotada pelo WSAP *plugin*.

Listagem 4: *WASP Plugin* - Criar plugin

```
$ mvn -U archetype:generate -Dfilter="io.jenkins.archetypes:"
...
Choose archetype:
1: remote -> io.jenkins.archetypes:empty-plugin (Skeleton of a Jenkins plugin
↪  with a POM and an empty source tree.)
2: remote -> io.jenkins.archetypes:global-configuration-plugin (Skeleton of a
↪  Jenkins plugin with a POM and an example piece of global configuration.)
3: remote -> io.jenkins.archetypes:global-shared-library (Uses the Jenkins
↪  Pipeline Unit mock library to test the usage of a Global Shared Library)
4: remote -> io.jenkins.archetypes:hello-world-plugin (Skeleton of a Jenkins
↪  plugin with a POM and an example build step.)
5: remote -> io.jenkins.archetypes:scripted-pipeline (Uses the Jenkins Pipeline
↪  Unit mock library to test the logic inside a Pipeline script.)
```

Na **Listagem 4** é possível consultar o comando usado para gerar a estrutura inicial do *plugin*. De seguida, foi necessário estender o comportamento do *plugin*, para que este possa ser executado quer num pipeline quer num *job*. Das várias possibilidades, a vertente *StepBuilder* evidenciou-se como a escolha mais adequada por possibilitar a criação de um *step* customizado com comportamento embutido.

Segundo o cenário da **Secção 4.3.1** (Arquitetura) para o uso da solução WSAP a máquina de testes tem de ter uma instância ativa do WSAP. Para solucionar este problema, o *plugin* através do protocolo SSH acede à máquina de testes e lança o serviço WSAP. Com serviço activo, o *plugin* consegue comunicar com a instância remota do WSAP através do protocolo Transmission Control Protocol (TCP).

Após a conclusão do *plugin* procedeu-se ao processo de publicação. Para publicar o *plugin* foi necessário criar um pedido de *Hosting* na plataforma *Jira*, como apresentado na **Figura 22**.

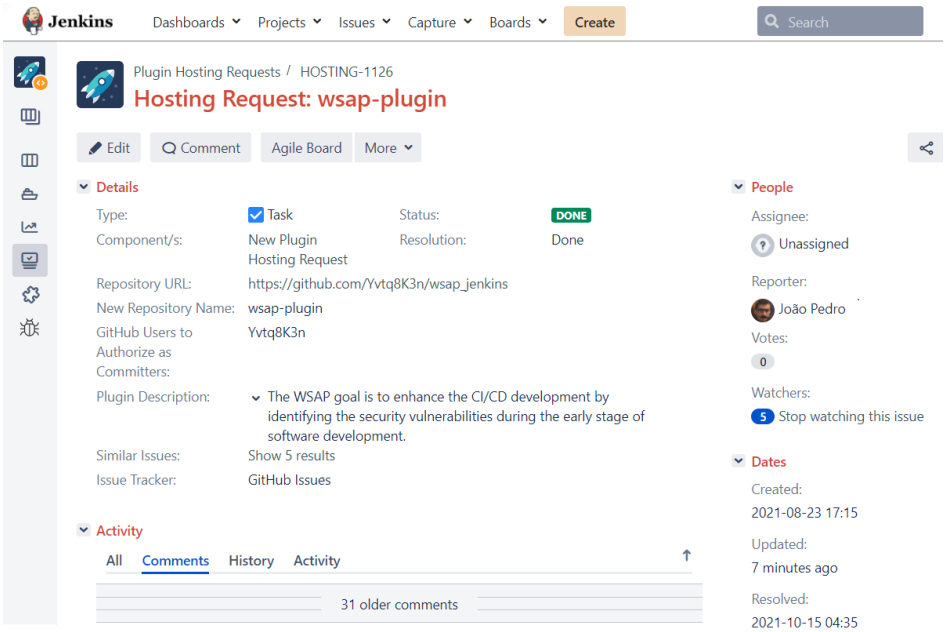


Figura 22: *WASP Plugin* - Hosting

Através do *Jira* a solução WSAP foi rigorosamente validada por diversas entidades e após o cumprimento de todas as recomendações aceite na plataforma *Jenkins*.

Com a aceitação do *plugin*, o código fonte foi automaticamente adicionado no repositório Source Code Management (SCM) do Jenkins. Procedeu-se a fase de publicação, onde através do *Maven* foi criado uma *release* e publicado-a no repositório *Maven* do *Jenkins* para que possa ser usada por outros programadores ou equipas de desenvolvimento.

Figura 23: *WASP Plugin* - Disponível no *Jenkins*

4.3.5 *Integração*

Como previamente apresentado na **Secção 4.3.1** (Arquitetura), a solução WSAP pode ser facilmente integrada no *Jenkins* através da especificação das propriedades da aplicação numa interface de linha de comandos. Esta abordagem consegue dar resposta aos requisitos inicialmente propostos. Contudo, a longo prazo, com as constantes alterações e o crescimento da aplicação torna-se difícil e penoso garantir que estas sejam refletidas no scanner.

Para endereçar estes problemas e tirando proveito da extensibilidade da plataforma *Jenkins*, foi desenvolvido um *BuildStep plugin* que, através da própria interface gráfica da plataforma, permite definir os parâmetros da aplicação em análise.

É de denotar que este *plugin* desenvolvido em *Java* e *Jelly* apenas permite complementar a solução WSAP, mas não realiza a análise em si. Somente redireciona os parâmetros recolhidos para uma máquina destinada para o efeito.

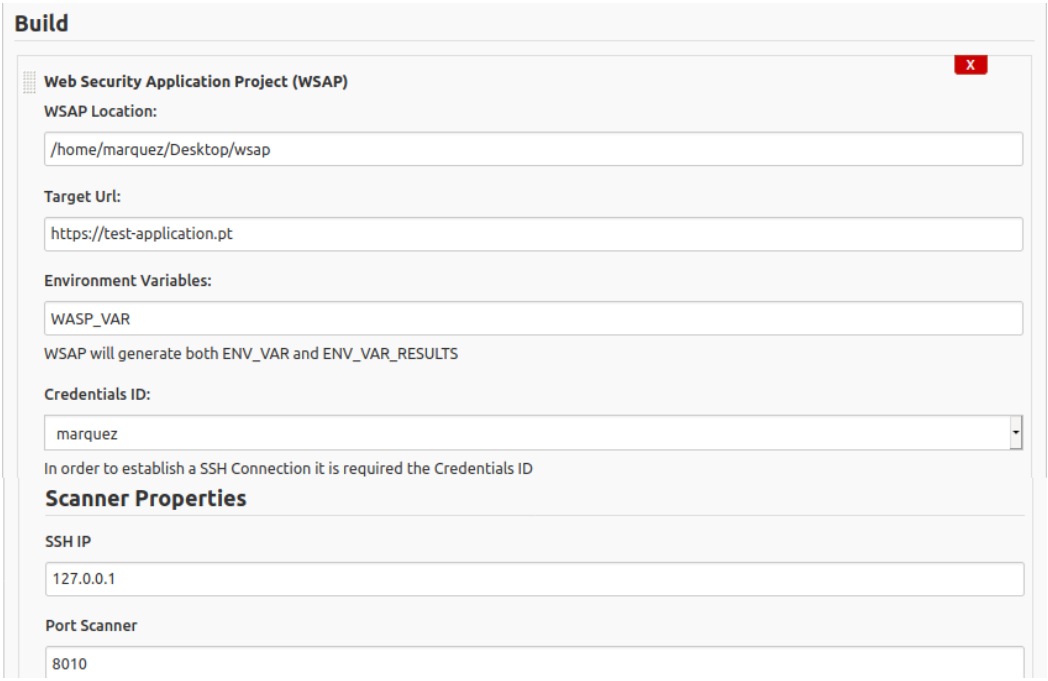


Figura 24: *WSAP Plugin* - Configurações base

Na **Figura 24** são apresentados os parâmetros base recolhidos pelo *plugin*, sendo estes:

- *WSAP Location* - localização da solução WSAP na máquina de teste;
- *Target Url* - endereço da aplicação a ser testada, que será usada como raiz no decorrer de todas as fases da análise do WSAP,

- *Enviroment Variables* - variáveis ambiente geradas no Jenkins, que permitem a este obter informações inerentes a análise;
- *Credentials ID* - credenciais *SSH* previamente definidas no Jenkins, que serão usadas para aceder à máquina de testes;
- *SSH IP* - endereço *IP* da máquina de testes que será usado para aceder via SSH.
- *Port Scanner* - porto que o WSAP irá usar no decorrer da sua análise.

Figura 25: *WSAP Plugin* - Análise *SAST*

O *plugin*, como apresentado na **Figura 25**, recolhe os parâmetros relativos à análise SAST, que como o WSAP apenas necessita da localização do código fonte do *back-end* da aplicação a ser testada.

A análise DAST também não apresenta grandes alterações. Através da interface gráfica do *Jenkins*, o *plugin* suporta a seleção do método de *scan* e atributos a este associados, repetições de parâmetros como o *include.url* e *exclude.url* e até mesmo providencia um *linter* para o *login.request*. Todas estas funcionalidades podem ser consultadas nos **Apêndices E e F**.

Após ser finalizada a configuração do *plugin*, através da plataforma *Jenkins* é possível proceder à sua execução como um *schedule job* ou num *pipeline*.

```
Processing reports...
Loading file: /home/marquez/Desktop/wsap/tmp/test[redacted]2021_08_30_18:14:34/insider_report.json
Loading file: /home/marquez/Desktop/wsap/tmp/test[redacted]2021_08_30_18:14:34/zap_report.json
Loading file: /home/marquez/Desktop/wsap/tmp/test[redacted]2021_08_30_18:14:34/wapiti_report.json
Loading file: /home/marquez/Desktop/wsap/tmp/test[redacted]2021_08_30_18:14:34/wapiti_report_admin@[redacted].pt.json

Analysis Summary:

SAST_InsiderCLI
- High [5]
- Medium [18]
- Low [0]
- Info [0]

DAST_ZAP
- High [0]
- Medium [94]
- Low [283]
- Info [0]


DAST_WAPITI
- High [0]
- Medium [0]
- Low [112]
- Info [4]
```

Figura 26: *WSAP Plugin* - Resultados da análise

No fim desta execução, o *plugin* gera um resumo das vulnerabilidades encontradas, como mostrado na **Figura 26**, e expõe os resultados da análise para as variáveis ambiente definidas na configuração. Através da plataforma Jenkins, estas variáveis podem ser usadas para notificar o estado da análise e das vulnerabilidades encontradas. Podemos consultar um exemplo de uma notificação deste tipo na **Figura 27**.

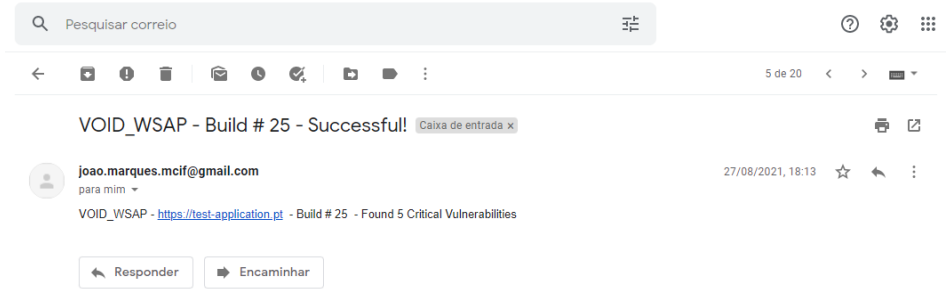


Figura 27: *WSAP Plugin* - Exemplo de notificação

4.4 TESTES E RESULTADOS

Para assegurar a qualidade e robustez do software WSAP, na etapa de testes e resultados optou-se por uma vertente mais prática, na qual, através desta pretendeu-se compreender as limitações da solução, os resultados obtidos e potenciais falhas.

Cumprindo com as políticas de confidencialidade empregues pela empresa foi providenciado acesso limitado a uma *Web Application*, denominado ao longo deste documento por *WA_SPA1*. Com o estudo desta foi possível compreender a sua estrutura e generalizar a solução WSAP de forma a poder ser usado noutras *Web Applications* desenvolvidas pela *VOID SOFTWARE, S.A.*

4.4.1 *Realização de Testes*

Na fase de realização de testes, um colaborador da empresa ficou responsável por aplicar a solução WSAP sobre as *Web Applications* desenvolvidas e disponibilizar os resultados de cada iteração para que pudessem ser analisados.

Na **Tabela 3** são apresentadas as propriedades, os resultados obtidos com a recolha dos dados gerados durante as análises do WSAP sobre as *Web Applications* testadas.

Tabela 3: Testes - Resultados de Dados

Name	Scan	Endpoints	Authentication	Logins	Duration
WA_SPA1	FULL	253	Successful	1	3h:43min
WA_SPA2	FULL	169	Successful	2	2h:24min
WA_SPA3	FULL	221	Successful	6	7h:58min
WA_SPA4	FULL	282	Failed	4	6h:52min
WA_SPA5	FULL	223	Failed	2	6h:03min
WA_SPA6	FULL	90	Successful	1	2h:55min
WA_SPA7	FULL	147	Successful	1	2h:47min
WA_SPA8	FULL	263	Successful	1	8h:11min

Com esta recolha pretende-se contextualizar as aplicações que foram analisadas segundo a perspetiva do WSAP, através das seguintes propriedades:

- *Name* - nome anonimizado da *Web Application*;
- *Scan* - o tipo de método de *scan* usado no decorrer da análise.
- *Endpoints* - quantidade de *endpoints* válidos que foram obtidos;
- *Authentication* - identifica que pelo menos uma autenticação foi realizada com sucesso;

- *Logins* - quantidade de utilizadores usados no decorrer do login;
- *Duration* - contabilização do tempo desde a criação da pasta à respetiva obtenção do relatório final.

As análises apresentadas na **Tabela 3** foram aplicadas sobre aplicações SPA, sendo usado a opção *FULL* e, com esta, obtidos os *endpoints* apresentados. O WSAP também conseguiu dar resposta ao processo de autenticação na maioria das *Web Applications* testadas, com a exceção da *WA_SPA4* que será descontinuada e da *WA_SPA5* que usa um processo de autenticação diferente do estudado.

4.4.2 *Análise de Resultados*

Na secção anterior foram apresentadas as propriedades das *Web Applications* analisadas pelo WSAP. Nesta secção, pretende-se interpretar os resultados obtidos com essas análises, nomeadamente a capacidade de detetar vulnerabilidades.

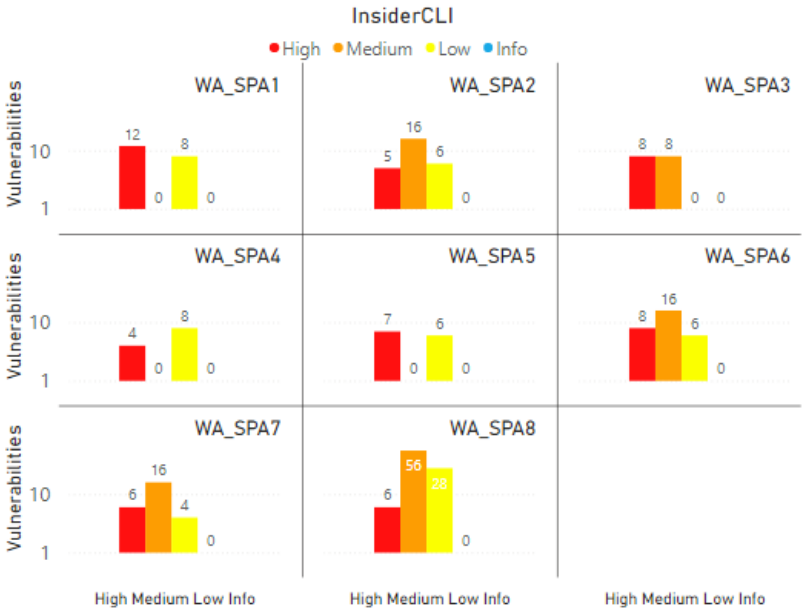


Figura 28: *InsiderCLI* - Vulnerabilidades

Na **Figura 28** é apresentado um gráfico com as vulnerabilidades encontradas nas várias *Web Applications* no decorrer da análise estática. Este gráfico, apresenta entropia nos resultados e até eventuais vulnerabilidades críticas a serem endereçadas. Devido ao género da solução *InsiderCLI* é normal existirem falsos positivos.

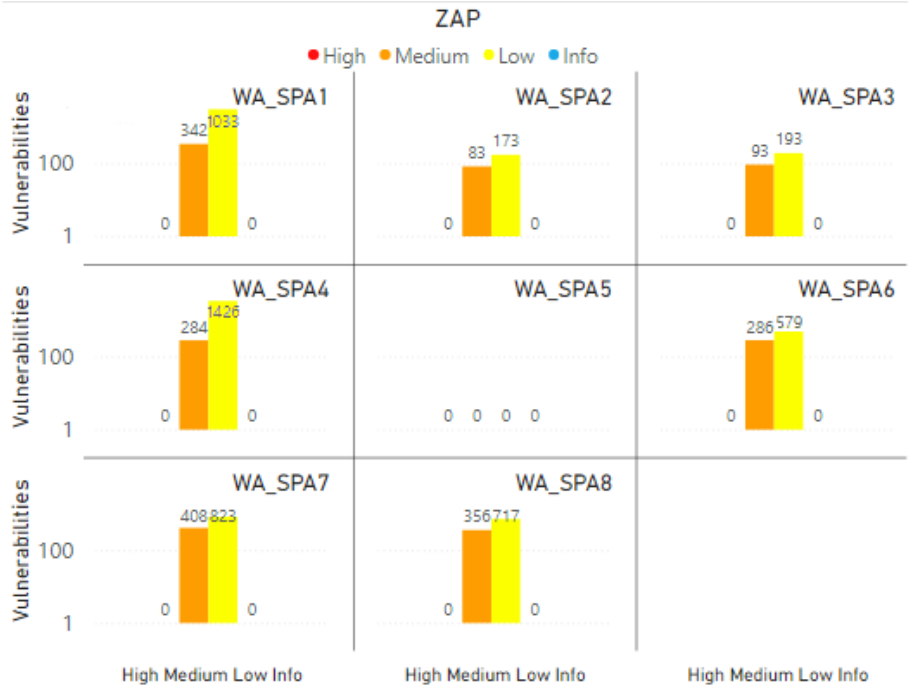


Figura 29: *ZAP* - Vulnerabilidades

Na análise dinâmica, a ferramenta *Zap* não conseguiu encontrar falhas críticas, o que realça a qualidade das aplicações desenvolvidas pela *VOID SOFTWARE, S.A.* Em relação à categoria *Info*, devido ao seu baixo nível de importância e pelo facto de apresentar um grande volume de dados não foram recolhidos e por isso não são apresentados.

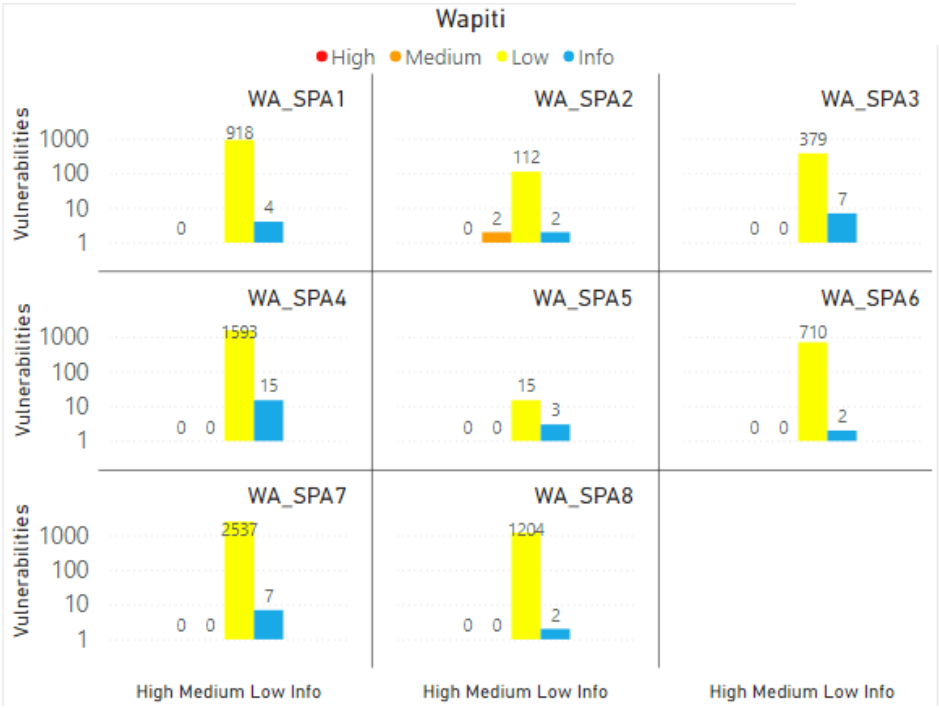


Figura 30: *Wapiti* - Vulnerabilidades

A análise efetuada pelo ferramenta *Wapiti* focou-se mais no modelo MPA pelo que teve algumas dificuldades em encontrar vulnerabilidades com um maior grau de criticidade. É de evidenciar, mesmo assim, na *WP_SPA2* conseguiu encontrar algumas vulnerabilidades de categoria média.

4.4.3 *Retrospectiva e Adoção*

Com os testes efetuados a ferramenta WSAP conseguiu cumprir com as expectativas ao identificar as vulnerabilidades presentes nas diferentes *Web Applications* desenvolvidas. É de realçar que para a deteção destas vulnerabilidades, o WSAP aplicou as metodologias *black-box* e *white-box*.

Nesta fase de testes foi possível avaliar o funcionamento da aplicação WSAP e introduzir a ferramenta no processo de desenvolvimento contínuo da empresa.

Segundo o testemunho da *VOID SOFTWARE, S.A.*, a solução WSAP veio reforçar na busca pelo desenvolvimento de aplicações robustas e seguras. Passo a citar, o *feedback* obtido:

> *"- Na VOID, a implementação do WSAP veio complementar o conjunto de ferramentas e procedimentos de segurança existentes. Considera-*

*mos que a segurança dos dados é essencial e deve estar presente ao longo de todo o ciclo de vida do software. A utilização do WSAP no processo de desenvolvimento da VOID permite-nos desde muito cedo detectar eventuais vulnerabilidades e más práticas de segurança. A integração de diversas ferramentas distintas para análises SAST e DAST e a possibilidade de expansão futura através de outras ferramentas e plugins fazem do WSAP uma solução versátil e capaz de ser aplicada em diferentes cenários. O Jenkins é utilizado na VOID como ferramenta CI/CD e com o plugin criado para o efeito, é possível integrar facilmente o WSAP nas pipelines CI/CD existentes. Adicionalmente, a ferramenta CLI permite-nos executar análises específicas não integradas nas pipelines. No final de cada análise os developers podem ser notifcados directamente com um resumo das incidências e é enviado um relatório das vulnerabilidades encontradas, sua gravidade e ferramenta utilizada. Durante o desenvolvimento do WSAP e respectivo plugin para o Jenkins, algumas das suas funcionalidades foram sendo complementadas conforme o feedback dado pela VOID permitindo adaptar facilmente a sua utilização à nossa infraestrutura e processos de desenvolvimento."(*VOID SOFTWARE, S.A.*, 2021)*

5

CONCLUSÕES

Neste estágio tive a oportunidade de fazer parte da *VOID SOFTWARE, S.A.* onde fui inserido no contexto empresarial e pus em prática os conhecimentos adquiridos ao longo do mestrado de Cibersegurança e Informática Forense.

Para solucionar o desafio proposto, de adicionar uma camada de segurança ao processo de desenvolvimento, foi realizado todo um estudo das várias metodologias, tecnologias e ferramentas, afim de garantir uma correta implementação do mecanismo de segurança, *Security Automation*.

A implementação deste mecanismo culminou no desenvolvimento da solução WSAP, um agregador de ferramentas de *Web Security*, que possibilita a deteção de vulnerabilidades segundo as metodologias DAST e SAST. A adoção destas metodologias permite que a análise efetuada consiga analisar a aplicação de diferentes perspetivas e desta forma realizar um levantamento das vulnerabilidades o mais completo possível.

No que toca a integração no pipeline CI/CD, a solução WSAP evidenciou-se como uma solução versátil ao providenciar múltiplas formas de integração, quer via linha de comandos, quer através da interface gráfica disponibilizada pelo *plugin*.

Com isto pode-se concluir que, a solução WSAP consegui cumprir com os objetivos definidos e está atualmente a ser aplicada no processo de desenvolvimento da *VOID SOFTWARE, S.A.*

5.1 TRABALHO FUTURO

Como trabalho futuro seria interessante continuar o desenvolvimento da solução WSAP, que tirando proveito da sua estrutura, permite incorporar novas ferramentas de segurança.

Através da interface gráfica providenciada pelo *plugin*, poderia ser explorado a possibilidade de apresentar em gráficos as informações relativas à análise efetuada e atribuir-lhe uma classificação.

Como foi observado na fase de testes, a análise realizada pelo WSAP facilmente se pode tornar morosa, pelo que seria uma mais valia estudar a possibilidade de realizar optimizações no próprio WSAP ou nas ferramentas que implementa.

BIBLIOGRAFIA

Black, Paul E. et al. (2007). *NIST 500-269: Web Application Security Scanner. Software Assurance Tools: Web Application Security Scanner Functional Specification Version 1.0*. https://samate.nist.gov/docs/webapp_scanner_spec_08_21_07.pdf. (Acedido em 11/01/2020).

Chen, Shay (2016). *The Input Delivery Method Scanning Support of Web Application Vulnerability Scanners. WAVSEP Benchmark 2014/2016*. http://www.sectoolmarket.com/input-vector-support-of-open-source-web-application-scanners.html. (Acedido em 15/04/2021).

— (2018). *Evaluation of Web Application Vulnerability Scanners in Modern Pentest. WAVSEP Benchmark 2017/2018*. https://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html. (Acedido em 15/04/2021).

Cross, Michael et al. (2007). *Web Application Vulnerabilities. Detect, Exploit, Prevent*. http://index-of.co.uk/Hacking-Coleccion/WebApplicationVulnerabilities-Detect,Exploit,Prevent.pdf. Syngress Publishing, Inc; Elsevier, Inc. ISBN: "978-1-59749-209-6".

Google (2012). *Skipfish*. https://code.google.com/archive/p/skipfish/. (Acedido em 15/04/2021).

Heller, Martin (2020). *What is Jenkins? The CI server explained*. https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html. (Acedido em 13/04/2021).

Hoffman, Andrew (2020). *Web Application Security. Exploitation and Countermeasures for Modern Web Applications*. O'Reilly Media, Inc. ISBN: "978-1-49205-311-8".

Jajodia, Sushil et al. (2012). *Moving Target Defense II. Application of Game Theory and Adversarial Modeling*. https://mlsec.info/pdf/mtd13.pdf. Springer Publishing Company, Incorporated. ISBN: "978-1-4614-5415-1".

Janca, Tanya (2020). *Alice and Bob Learn Application Security*. Willey. ISBN: "978-1-119-68735-1".

JTFTI, Joint Task Force Transformation Initiative (2011). *NIST 800-39: Managing Information Security Risk. Organization, Mission, and Information System View*. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-39.pdf. (Acedido em 18/01/2021).

Kaspersky (2017). *What is Cyber Security?* https://www.kaspersky.com/resource-center/definitions/what-is-cyber-security. (Acedido em 29/01/2021).

Kelebek, Burak (2015). *Automating Security Tests using OWASP ZAP and Jenkins*. https://www.securify.nl/blog/automating-security-tests-using-owasp-zap-and-jenkins. (Acedido em 21/06/2021).

Kulshrestha, Saurabh (2020). *What is Jenkins? Jenkins For Continuous Integration*. https://www.edureka.co/blog/what-is-jenkins/. (Acedido em 13/04/2021).

Laurent, Simon St. (2014). *Web Application development i different and better - O'Reilly*. https://www.oreilly.com/radar/web-application-development-is-different-and-better/. (Acedido em 06/04/2021).

McKenzie, Cameron (2020). *Declarative vs. scripted pipelines: What's the difference?* https://www.theserverside.com/answer/Declarative-vs-scripted-pipelines-Whats-the-difference. (Acedido em 27/07/2021).

Merkow, Mark S. (2019). *SECURE, RESILIENT, AND AGILE SOFTWARE DEVELOPMENT*. Prentice Hall. ISBN: "978-0-367-33259-4".

MITRE (2020). *Common Weakness Enumeration*. https://cwe.mitre.org/about/index.html. (Acedido em 07/01/2021).

OWASP (2019). *Vulnerability Scanning Tools*. https://owasp.org/www-community/Vulnerability_Scanning_Tools. (Acedido em 15/04/2021).

— (2020a). *OWASP - Top Ten. Introduction*. https://owasp.org/www-chapter-new-zealand/assets/slides/2020-02-09-IntroductiontotheOWASPTopTen.pdf. (Acedido em 10/01/2021).

— (2020b). *Source Code Analysis Tools*. https://owasp.org/www-community/Source_Code_Analysis_Toolsl. (Acedido em 19/05/2021).

Programatic (2015). *Ameaça, Vulnerabilidade e Risco*. http://www.programatic.com.br/canal/blog/programatic/2015/06/30/ameaca-vulnerabilidade-e-risco/. (Acedido em 02/02/2021).

Rieger, C. G., D. I. Gertman e M. A. McQueen (2009). «Resilient control systems: Next generation design research». Em: *2009 2nd Conference on Human System Interactions*, pp. 632–636. DOI: 10.1109/HSI.2009.5091051.

ScaleSec (2020). *Vulnado (Intentionally Vulnerable Java Application)*. https://github.com/ScaleSec/vulnado. (Acedido em 15/06/2021).

Simplilearn (2018). *What Is Jenkins?* Video. https://www.youtube.com/watch?v=LFDrDnKPOTg.

Skólski, Paweł (2016). *Single-page application vs. multiple-page application*. https://neoteric.eu/blog/single-page-application-vs-multiple-page-

application/?utm_source=medium.com&utm_medium=social&utm_content=neo&utm_campaign=blog. (Acedido em 06/04/2021).

Software, Void (2006). *VOID SOFTWARE, S.A.* Website. https://www.linkedin.com/company/void/about/. (Acedido em 30/11/2020).

Stallings, William (2005). *Cryptography and Network Security Principles and Practices, Fourth Edition. Principles and Practices*. Prentice Hall. ISBN: "0-13-187316-4".

SubGraph (2014). *Vega Vulnerability Scanner*. https://subgraph.com/vega/. (Acedido em 15/04/2021).

Sullo, Chris (2021a). *Nikto*. https://github.com/sullo/nikto. (Acedido em 15/04/2021).

— (2021b). *Sqlmap. Automatic SQL injection and database takeover tool*. http://sqlmap.org/. (Acedido em 15/04/2021).

Surribas, Nicolas (2021). *Wapiti. a Free and Open-Source web-application vulnerability scanner in Python for Windows, Linux, BSD, OSX*. https://wapiti.sourceforge.io/. (Acedido em 15/04/2021).

Team, Apps (2013). *An Intro Into Single Page Applications (SPA)*. Website. https://blog.4psa.com/an-intro-into-single-page-applications-spa/.

Team, Lvivity (2020). *Single-page App vs. Multi-page App: Pros, Cons, and Which is Better?* Website. https://lvivity.com/single-page-app-vs-multi-page-app.

W3AF (2013). *W3af. Open Source Web Application Security Scanner*. https://w3af.org/. (Acedido em 15/04/2021).

Watts, Stephen (2020). *IT Security Vulnerability vs Threat vs Risk: What are the Differences?* https://www.bmc.com/blogs/security-vulnerability-vs-threat-vs-risk-whats-difference/. (Acedido em 18/01/2021).

ZAP, OWASP (2021). *ZAP*. https://www.zaproxy.org/. (Acedido em 15/04/2021).

APÊNDICES

APÊNDICE A

Listagem 5: *PMD* - Excerto do relatório gerado com a análise da aplicação Vulnado


```
{
  "formatVersion": 0,
  "pmdVersion": "6.35.0",
  "timestamp": "2021-06-15T18:43:40.035+01:00",
  "files": [
    {
      "filename": "/home/jenkins/vulnado/src/main/java/com/scalesec/vulnado/Comment.java",
      "violations": [
        {
          "beginline": 3,
          "begincolumn": 1,
          "endline": 3,
          "endcolumn": 34,
          "description": "Unused import \u0027org.apache.catalina.Server\u0027",
          "rule": "UnnecessaryImport",
          "ruleset": "Code Style",
          "priority": 4,
          "externalInfoUrl": "https://pmd.github.io/pmd-6.35.0/pmd_rules_java_codestyle.html#unnecessaryimport"
        },
              {
          "beginline": 30,
          "begincolumn": 10,
          "endline": 30,
          "endcolumn": 25,
          "description": "Field comments are required",
          "rule": "CommentRequired",
          "ruleset": "Documentation",
          "priority": 3,
          "externalInfoUrl": "https://pmd.github.io/pmd-6.35.0/pmd_rules_java_documentation.html#commentrequired"
        },
        {
          "beginline": 30,
          "begincolumn": 17,
          "endline": 30,
          "endcolumn": 24,
          "description": "Found non-transient, non-static member. Please mark as transient or provide accessors.",
          "rule": "BeanMembersShouldSerialize",
          "ruleset": "Error Prone",
```

```
            "priority": 3,
            "externalInfoUrl": "https://pmd.github.io/pmd-6.35.0/pmd_rules_java_errorprone.html#beanmembersshouldserialize"
          },
          ...
      {
        "filename": "/home/jenkins/vulnado/src/main/java/com/scalesec/vulnado/Postgres.java",
        "violations": [
          {
            "beginline": 12,
            "begincolumn": 23,
            "endline": 117,
            "endcolumn": 1,
            "description": "All methods are static.  Consider using a utility class instead. Alternatively, you could add a private constructor or make the class abstract to silence this warning.",
            "rule": "UseUtilityClass",
            "ruleset": "Design",
            "priority": 3,
            "externalInfoUrl": "https://pmd.github.io/pmd-6.35.0/pmd_rules_java_design.html#useutilityclass"
          },
          ...
          {
            "beginline": 56,
            "begincolumn": 13,
            "endline": 56,
            "endcolumn": 30,
            "description": "System.out.println is used",
            "rule": "SystemPrintln",
            "ruleset": "Best Practices",
            "priority": 2,
            "externalInfoUrl": "https://pmd.github.io/pmd-6.35.0/pmd_rules_java_bestpractices.html#systemprintln"
          }
        ]
      },
          ...
    ],
    "suppressedViolations": [],
    "processingErrors": [],
    "configurationErrors": [
      {
        "rule": "LoosePackageCoupling",
        "ruleset": "Design",
        "message": "No packages or classes specified"
      }
    ]
}
```

B

APÊNDICE B

Listagem 6: *InsiderCLI* - Excerto do relatório gerado com a análise da aplicação Vulnado


```
{
 "vulnerabilities": [
  {
   "cvss": 3.2,
   "cwe": "CWE-532",
   "line": 10,
   "class": "Cowsay.java (10:5)",
   "vul_id": "752ec8bb9b3b233b30dfcd426b152d55",
   "method": "System.out.println(cmd);",
   "column": 5,
   "description": "The App logs information. Sensitive information should not be
    ↪  logged.",
   "classMessage": "main/java/com/scalesec/vulnado/Cowsay.java (10:5)"
  },
  ...
  {
   "cvss": 3.2,
   "cwe": "CWE-532",
   "line": 28,
   "class": "LinkLister.java (28:7)",
   "vul_id": "8c34acc24095fa80207426ba9a3fa42c",
   "method": "System.out.println(host);",
   "column": 7,
   "description": "The App logs information. Sensitive information should not be
    ↪  logged.",
   "classMessage": "main/java/com/scalesec/vulnado/LinkLister.java (28:7)"
  },
  ...
  {
   "cvss": 3.2,
   "cwe": "CWE-532",
   "line": 56,
   "class": "Comment.java (56:7)",
   "vul_id": "04b62a039eb059a41d55bb7eca8d6957",
   "method": "System.err.println(e.getClass().getName()+\": \"+e.getMessage());",
   "column": 7,
   "description": "The App logs information. Sensitive information should not be
    ↪  logged.",
   "classMessage": "main/java/com/scalesec/vulnado/Comment.java (56:7)"
  },
  {
   "cvss": 3.2,
```

```
    "cwe": "CWE-532",
    "line": 56,
    "class": "Comment.java (56:7)",
    "vul_id": "04b62a039eb059a41d55bb7eca8d6957",
    "method": "System.err.println(e.getClass().getName()+\": \"+e.getMessage());",
    "column": 7,
    "description": "The App logs information. Sensitive information should not be
    ↪  logged.",
    "classMessage": "main/java/com/scalesec/vulnado/Comment.java (56:7)"
   },
   ...
   {
    "cvss": 7.4,
    "cwe": "CWE-327",
    "line": 67,
    "class": "Postgres.java (67:32)",
    "vul_id": "b7650b9eb039f4bbab68a587c7a92283",
    "method": "MessageDigest md = MessageDigest.getInstance(\"MD5\");",
    "column": 32,
    "description": "MD5 is a hash algorithm considered weak and can return the
    ↪  same result for two different contents, which can cause collisions and in
    ↪  extreme cases it can cause a security breach.
    ↪  https://en.wikipedia.org/wiki/Collision_resistance",
    "classMessage": "main/java/com/scalesec/vulnado/Postgres.java (67:32)",
    "recomendation": "It is recommended to use some CHF (Cryptographic Hash
    ↪  Function), which is mathematically strong and not reversible. SHA512
    ↪  would be the most recommended hash for storing the password and it is
    ↪  also important to adopt some type of Salt, so that the Hash is more
    ↪  secure."
   },
   ...
  ],
  "none": 0,
  "low": 18,
  "medium": 0,
  "high": 2,
  "critical": 0,
  "total": 20,
  "sast": {
   "averageCvss": 7.4,
   "securityScore": 26,
   "size": "15680 Bytes",
   "numberOfLines": 499
  }
}
```

APÊNDICE C

Listagem 7: *ZAP* - *Script* de Autenticação

```
function logger() {
  print('[' + this['zap.script.name'] + '] ' + arguments[0]);
}

var HttpSender    = Java.type('org.parosproxy.paros.network.HttpSender');
var ScriptVars    = Java.type('org.zaproxy.zap.extension.script.ScriptVars');
var HtmlParameter = Java.type('org.parosproxy.paros.network.HtmlParameter')
var COOKIE_TYPE   = org.parosproxy.paros.network.HtmlParameter.Type.cookie;
var ForcedUser    = org.parosproxy.paros.control.Control.getSingleton()
                        .getExtensionLoader().getExtension(
                            org.zaproxy.zap.extension.forceduser.ExtensionForced
                            ↪ User.class
                        );

function sendingRequest(msg, initiator, helper) {
  if (initiator === HttpSender.AUTHENTICATION_INITIATOR) {
     logger("Sending authentication Request")
     return msg;
  }

  logger('Is forced user: '+ ForcedUser.isForcedUserModeEnabled())
  if (!ForcedUser.isForcedUserModeEnabled()) {return;}
  var token = ScriptVars.getGlobalVar("header_token")
  if (!token) {
     logger('Undefined token value')
     return;
  }

  logger("Adding authorization header token" + (' ' + token).slice(0, 20))
  var headers = msg.getRequestHeader();
  msg.getRequestHeader().setHeader('Authorization', token);
  return msg;
}

function responseReceived(msg, initiator, helper) {
  var resbody     = msg.getResponseBody().toString()
  var resheaders  = msg.getResponseHeader()

  if (initiator != HttpSender.AUTHENTICATION_INITIATOR) {
     return;
  }

```

```
42    var contextId = msg.getRequestingUser().getContextId()
43    var user = ForcedUser.getForcedUser(contextId)
44
45    logger('Trying to authenticate with user: '+ user.getName())
46    if (!ForcedUser.isForcedUserModeEnabled()) {return;}
47
48    logger("Handling auth response")
49    logger("Server Response: "+resheaders.getStatusCode())
50    if (resheaders.getStatusCode() > 299) {
51
52      logger("Auth failed")
53      return;
54    }
55
56    logger("Authentication request was successful")
57
58    //Retrieve Authorization Token from header
59    token = msg.getResponseHeader().getHeader("Authorization")
60
61    // If auth request was not succesful move on
62    if (token.length==0) {return;}
63
64    // @todo abstract away to be configureable
65    logger("Capturing token for Authorization\n" + token)
66    ScriptVars.setGlobalVar("header_token", token)
67  }
```

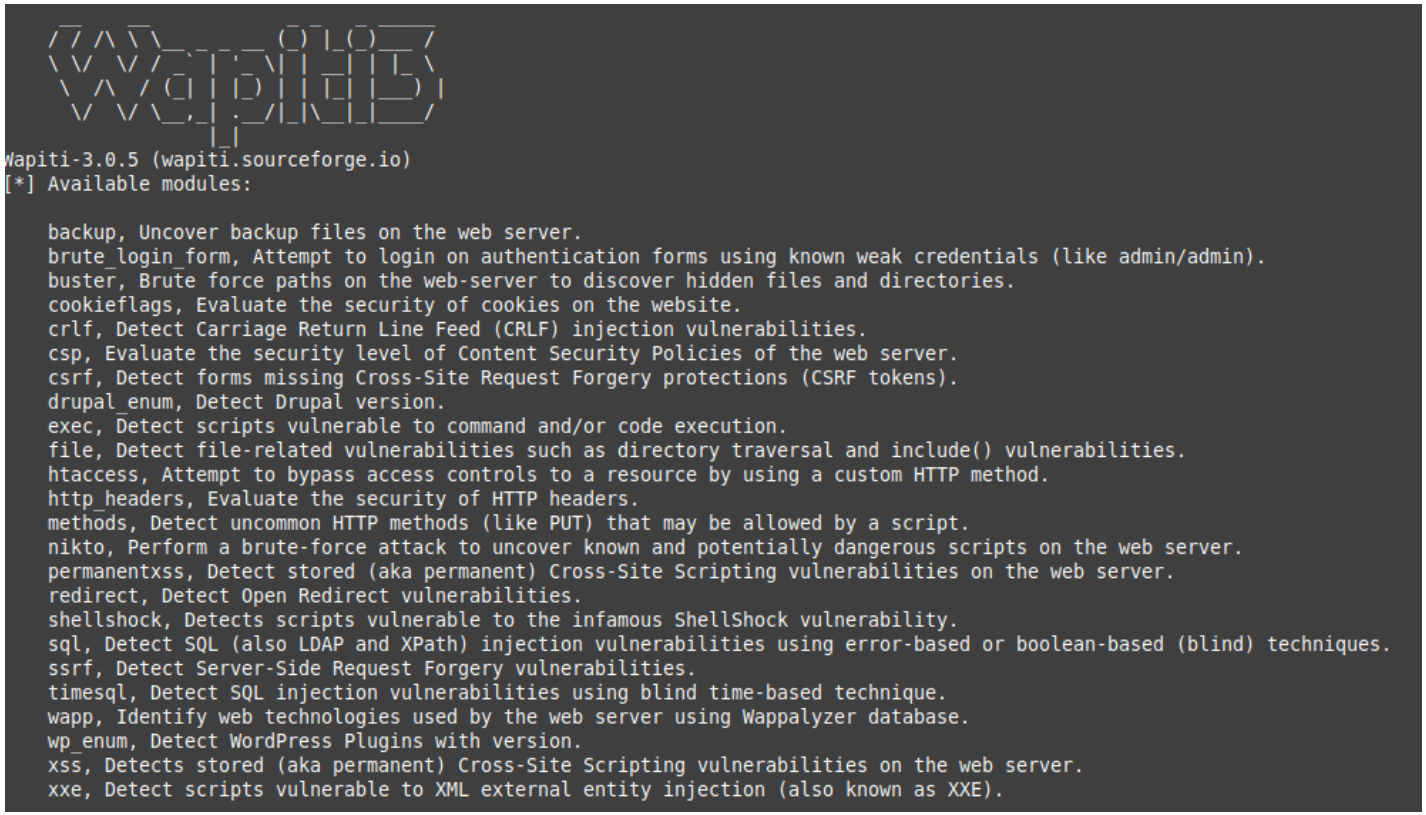


Figura 31: *Wapiti* - Recolha dos módulos de ataque

E

APÊNDICE E

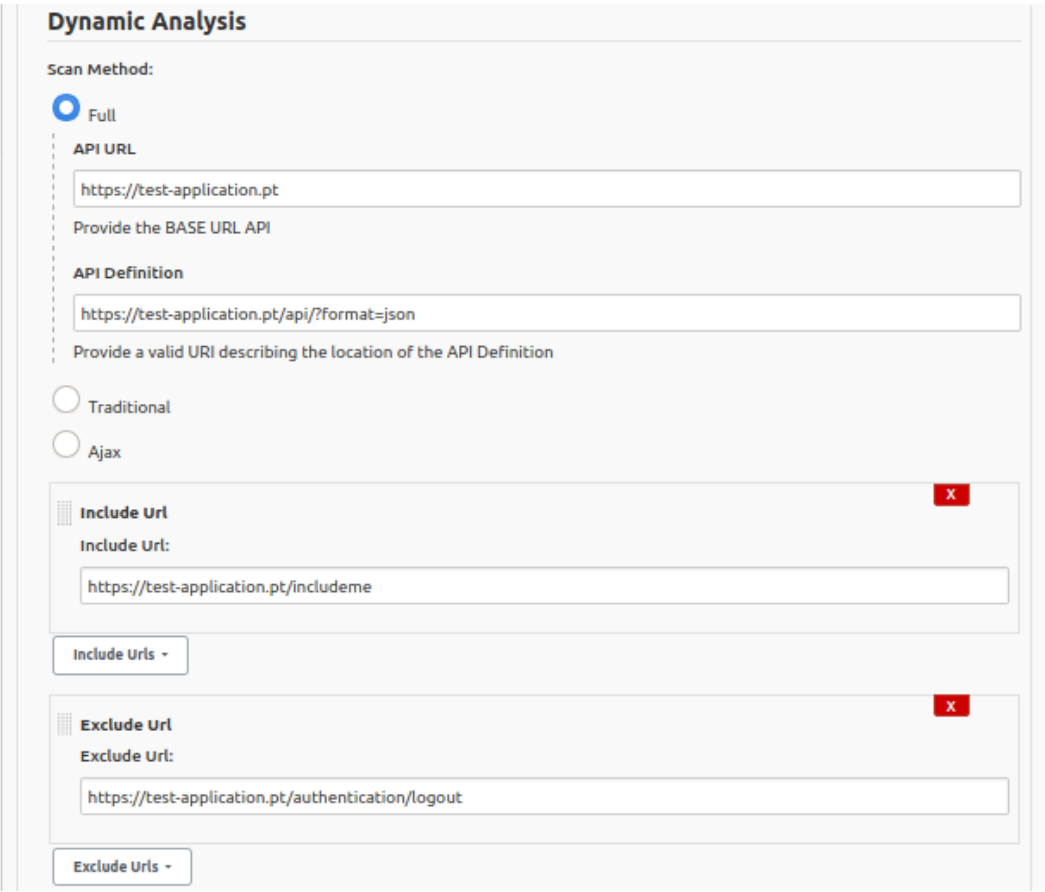


Figura 32: *WSAP Plugin* - Análise *DAST*

APÊNDICE F

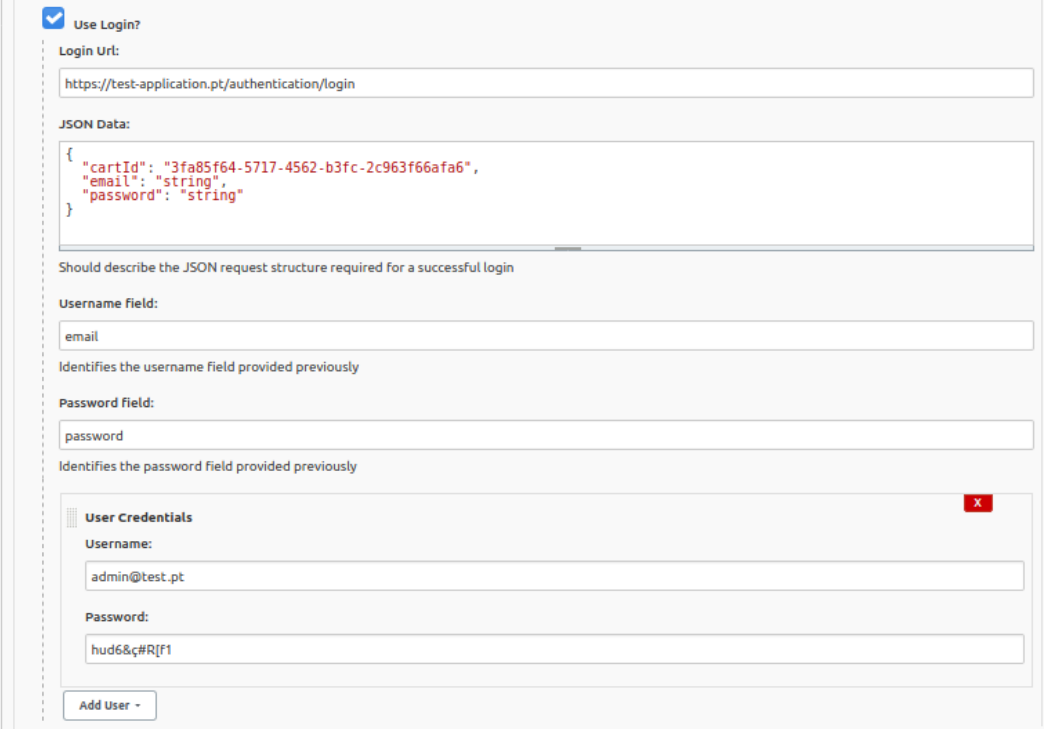


Figura 33: *WSAP Plugin* - Análise *DAST* - Login

ANEXOS

I

ANEXO I

Name	Description	Related terms	OWASP Top Ten 2007	CWE ID	PCI DSS 1.1
Cross Site Scripting (XSS)	A web application accepts user input (such as client-side scripts and hyperlinks to an attacker's site) and displays it within its generated web pages without proper validation.	Reflected XSS, persistent (stored) XSS, DOM-based XSS	A1	79	X
SQL Injection	Unvalidated input is used in construction of an SQL statement.	Blind SQL injection	A2	89	X
OS Command Injection	Unvalidated input is used in an argument to a system operation execution function.		A2	78	X
XML Injection	Unvalidated input is inserted into an XML document.	XPath injection, XQuery injection	A2	91	X
HTTP Response Splitting	Unvalidated input is used in construction of HTTP response headers.	CRLF injection	A2	113, 93	X
Malicious File Inclusion	Unvalidated input is used in an argument to file or stream functions.	File inclusion, Remote code execution, Directory traversal	A3	98	
Insecure Direct Object Reference	Unvalidated input is used as a reference to an internal implementation object, such as a file, directory, or database key.	Parameter tampering, Cookie poisoning, Path manipulation	A4	233, 73, 472	X
Cross Site Request Forgery (CSRF)	An application authorizes requests based only on credentials that are automatically submitted by the browser. A CSRF attack forces a logged-in victim's browser to send a request to a vulnerable application, which then performs the chosen action on behalf of the victim, to the benefit of the attacker.	Session riding, One-click attacks, Hostile Linking	A5	352	
Information Leakage	Disclosure of sensitive information or the internal details of the application.	File and directory information leaks, System information leak.	A6	538, 200, 497	X
Improper Error Handling	Error message may display too much information that is useful in exploring a vulnerability.	Error message information leaks, Detailed error handling	A6	388, 209, 390	X
Weak Authentication and Session Management	Lack of proper protection of account credentials and session tokens through their lifecycle.		A7	287	X
Session Fixation	Authenticating a user without invalidating any existing session identifier. This gives an attacker the opportunity to steal authenticated sessions		A7	384	X
Insecure Communication	Transmitting sensitive information (session tokens, credit card numbers or health records) without proper encryption (e.g., SSL).		A9		X
Unrestricted URL Access	Missing or insufficient access control for sensitive URLs and functions.	Predictable resource location, security by obscurity	A10	425	X

Figura 34: *NIST 500-269: Web Application Security Scanner* - Vulnerabilidades

II

ANEXO II

Defense Mechanism	Description	Example	Affected vulnerabilities
Typecasting	Convert the input string to specific type, such as integer, Boolean, double.	(int)($_GET['var']) transforms "8<script>" into the integer 8	XSS, Injections, Cookie poisoning
Meta-character replacement	Encode characters from a blacklist	"<" is replaced with "<" for HTML documents, quotes replacement… For XSS, replace these characters: ', ", <, >, &, %, #	All technical vulnerabilities
Restrict input range	Restrict the range of integers, the type of an entry (only alphanumeric), length of a string etc.	For HTML injection, use a regular expression such as: [a-zA-Z0-9_\s]+ to restrict the input to alphanumeric characters plus underscore and space. Using data binding for SQL queries (prepared statements), etc.	All technical vulnerabilities
Restricted user management	Use a restricted account for performing data manipulation, SQL queries, etc.	If user is not logged in, use a read-only SQL account that only allows commands like SELECT and EXECUTE; no UPDATE or INSERT allowed.	SQL Injection, OS Command Injection
Use of stronger function	Use a stronger function for performing a secure action	Use SHA-256 instead of SHA-1 or MD5, Salt the passwords, Secure cookies. Using HttpOnly cookies to prevent client-side script to read them.	Weak cryptographic functions, Insecure cookies
Token Validation	Use a unique token to verify the origin of the HTTP request	By adding a unique identifier (token) in the forms or in the HTTP header, the application must be able to verify the origin of the request	Cross-Site Request Forgery
Character encoding handling	Canonicalize resource names and other input that may contain encoded characters.	Sample attacks using encoding: XSS: The UTF-7 encoded string: "+ADw-script+AD4-	XSS, Injections, Cookie poisoning

Figura 35: *NIST 500-269: Web Application Security Scanner* - Defense

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado neste relatório de estágio, com o título *"Web Security Application Project"*, é original e foi realizado por João Pedro Borges Ferreira Castanheira Marques (2192637) sob orientação de Professor Doutor Marco António de Oliveira Monteiro (marco.monteiro@ipleiria.pt).

Leiria, Dezembro de 2021

João Pedro Borges Ferreira Castanheira Marques