Carolina de Castilho Godinho

Análise da Composição de Software (Sca) como Medida para Reduzir o Risco Cibernético da Cadeia de Fornecimento

1 2 9 0

UNIVERSIDADE D
COIMBRA

Carolina de Castilho Godinho

Análise da Composição de Software (Sca) como Medida para Reduzir o Risco Cibernético da Cadeia de Fornecimento

VOLUME 1

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Segurança Informática orientada pelo Doutor Afonso Neto e pelo Professor Doutor Paulo Simões e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Setembro de 2022

Faculdade de Ciências e Tecnologia
Departamento de Engenharia Informática

ANÁLISE DE COMPOSIÇÃO DE SOFTWARE

Carolina de Castilho Godinho

Relatório de Estágio no âmbito do Mestrado em Engenharia Informática, especialização em Segurança Informática orientada pelo Doutor Afonso Neto e pelo Professor Doutor Paulo Simões e apresentada à Faculdade de Ciências e Tecnologia / Departamento de Engenharia Informática.

Setembro de 2022

1 2 9 0

UNIVERSIDADE D
COIMBRA

Resumo

Com o grande crescimento do setor tecnológico, ocorreu também o aumento da procura de software. Para responder rapidamente a esta necessidade, uma alternativa é a utilização de software já criado por terceiros, principalmente software *open-source*, o que resulta num processo de desenvolvimento mais eficiente. A utilização de software de terceiros, no entanto, acaba por criar uma dependência entre o software desenvolvido e os de terceiros, o que necessariamente levanta questões de segurança. Em especial, chama a atenção a possibilidade de vulnerabilidades, a necessidade de atender às condições impostas pelo licenciamento, e o facto de que os processos de desenvolvimento de terceiros podem utilizar premissas de segurança diferentes das desejadas. Desta forma, é necessário gerir e reduzir os riscos de segurança da utilização de software de terceiros, sendo este o papel da Análise de Composição de Software (SCA – *Software Composition Analysis*).

O objetivo deste trabalho é o desenvolvimento de um procedimento que permite avaliar e selecionar ferramentas de SCA. Este procedimento pode ser utilizado, por exemplo, para selecionar ferramentas de SCA no contexto de um processo de aquisição de software num ambiente corporativo.

Para este fim, o trabalho apresenta a SCA, a sua contextualização no âmbito dos ciclos de desenvolvimento de software, e os trabalhos relacionados. Para fins de construção do procedimento proposto, são apresentados e discutidos um conjunto de métricas e parâmetros que tipicamente podem ser utilizados na avaliação destas ferramentas. Por fim, o procedimento é aplicado e testado na prática, sendo a sua eficiência validada.

Este trabalho é realizado no âmbito do estágio na empresa (do Mestrado de Segurança Informática da Universidade de Coimbra) Critical Software.

Palavras-Chave

SCA, software, segurança, vulnerabilidades, dependências, cadeia de fornecimento de software, FOSS, OSS

Abstract

With the great growth of the technology sector, the demand for software has also increased. To quickly respond to this need an alternative is to use software already created by third parties, mainly open-source software, which results in a more efficient development process. The use of third-party software, however, ends up creating a dependency between the developed software and third-party software, which necessarily raises security issues. In particular, the possibility of vulnerabilities, the need to comply with licensing conditions, and the fact that third-party development processes may use different security assumptions than desired. Thus, it is necessary to manage and reduce the security risks of using third-party software, and this is the role of Software Composition Analysis (SCA).

The objective of this work is to develop a procedure to evaluate and select SCA tools. This procedure can be used, for example, to select SCA tools in the context of a software procurement process in a corporate environment.

To this end, the work presents SCA, its contextualization in the context of software development cycles, and related works. For the purposes of developing this procedure, a set of metrics and parameters that typically are used when evaluating these tools are presented and discussed. Finally, the procedure is applied and tested in practice, and its effectiveness is validated.

This work is carried out in the scope of the internship at Critical Software of the MSc in Computer Security of the University of Coimbra.

Keywords

SCA, software, security, vulnerabilities, dependencies, software supply chain, FOSS, OSS

Agradecimentos

Ao longo de um ano de realização desta dissertação ultrapassei diversos desafios, obstáculos, incertezas, percalços, tristezas e alegrias, sendo este um processo solitário a que qualquer estudante de mestrado se submete dispendendo de inúmeras horas de trabalho, que só foram possíveis ultrapassar graças ao incentivo e apoio que recebi, os quais foram indispensáveis para esta etapa da minha vida, que estou eternamente grata e a que exprimo os meus sinceros agradecimentos.

Ao meu orientador de estágio, Professor Doutor Afonso Neto, pela sua orientação, total apoio, disponibilidade, pela sabedoria transmitida, pelo solucionamento de dúvidas, pelas suas opiniões e críticas construtivas, que foi um pilar essencial para que este trabalho fosse possível.

Ao meu orientador da universidade, Professor Doutor Paulo Simões, pela sua exigência, ensinar-me a ultrapassar os meus limites, pela sua visão crítica, pelo tempo dispendido em revisões formais do trabalho e opiniões que contribuíram para enriquecer o trabalho realizado.

À minha família, que sempre apoiou as minhas decisões, que sempre teve orgulho do meu trabalho e esforço, e que sempre se preocupou comigo. Até mesmo aqueles que deixaram de estar presentes durante esta fase da minha vida e que terei sempre no meu coração, o meu avô Joaquim Fernando Castilho, a minha bisavó Arminda dos Santos Trindade e a minha cadelinha Miley.

À minha família de praxe, principalmente, Ana Luísa Coelho, Margarida Felício, Maria Paula Viegas, Sofia Meireles, Bárbara Gonçalves que se tornaram uma segunda família para mim dentro da faculdade e por quem estou grata de todo o apoio, felicidade, alegria e ensinamentos que me deram e sobretudo o seu companheirismo incondicional.

Aos meus amigos e colegas de residência com quem partilhei a maioria do meu tempo ao realizar este trabalho, que foram meus companheiros de estudo e me proporcionaram momentos de alegria. Principal agradecimento à minha colega de quarto, Inês Peres, que acompanhou toda a minha jornada, que foi uma verdadeira amiga e sempre me apoiou, escutou, ajudou e que nunca esquecerei.

Ao meu namorado, Diogo Cascalheira, pelo seu incondicional apoio, amor, paciência, ajuda, companheirismo, lealdade e sinceridade. Agradeço a sua enorme alegria e felicidade com que me brindou constantemente, contribuindo chegar ao final deste percurso.

Por fim, concedo um especial agradecimento aos meus pais, Liliana Castilho e Eduardo Godinho, a quem dedico este trabalho, por serem modelos de coragem e bravura, que fizeram enormes sacrifícios para eu ser quem sou hoje e me fazerem chegar até aqui, pelo seu apoio incondicional, incentivo, paciência e ajuda a superar todos os obstáculos que foram aparecendo ao longo do caminho. Não há palavras que descrevam o quão estou agradecida e orgulhosa de todo o esforço investido em mim. Obrigada pais!

Índice

Abreviaturas, Acrónimos e Siglas

- AHP - *Analytic Hierarchy Process*
- API - *Application Programming Interface*
- BIOS - *Basic Input/Output System*
- BOM - *Bill Of Materials*
- CD/CI - *Continuous Delivery/Continuous Integration*
- CLI - *Command-Line Interface*
- CVE - *Common Vulnerabilities and Exposures*
- DAST - *Dynamic Application Security Testing*
- DevOps - *Development Operations*
- DIY - *Do It Yourself*
- FOSS - *Free and Open-Source Software*
- GPL - *General Public License*
- IAST - *Interactive Application Security Testing*
- IDE - *Integrated Development Environment*
- JNDI - *Java Naming Directory Interface*
- M&A - *Mergers and Acquisitions*
- MAC - *Media Access Control*
- NPM - *Node Package Manager*
- NVD - *National Vulnerability Database*
- OSS - *Open-Source Software*
- OWASP - *Open Web Application Security Project*
- SAST - *Static Application Security Testing*
- SBOM - *Software Bill of Materials*
- SCA - *Software Composition Analysis*
- SDK - *Software Development Kit*
- SDLC - *Software Development Life Cycle*
- SLA - *Service-Level Agreement*
- SPDX - *Software Package Data Exchange*
- TI - Tecnologia da Informação
- UEFI - *Unified Extensible Firmware Interface*
- UI - *User Interface*
- URL - *Uniform Resource Locators*
- XML - *Extensible Markup Language*

Lista de Figuras

Lista de Tabelas

Capítulo 1
Introdução

1.1 Contexto e motivação

A defesa em profundidade é atualmente cada vez mais usada por muitas organizações, assim como o número e a complexidade dos ataques informáticos também aumenta. A SCA (Software Composition Analysis) vem responder como defesa em profundidade para ataques em cadeias de fornecimento de software [1].

A dependência entre software cresce cada vez mais e isso levanta riscos de segurança para quem os usa. Diferentes projetos podem ter diferentes premissas de segurança, licenças para fins diferentes, podem trazer vulnerabilidades para o novo software que usa a dependência. Estas dependências podem ser código *open-source*, bibliotecas e software de empresas ou de outras entidades.

A SCA possui ferramentas de teste de segurança para gerir a utilização dos componentes de software do qual dependem. Estas ferramentas executam scans automáticos no código, analisando se os dados estão em conformidade com as licenças e se existem vulnerabilidades que já são conhecidas no mercado. Faz-se uma verificação se existem pacotes maliciosos e vulneráveis através de uma lista de bloqueio, ou seja, se uma versão de um pacote estiver na lista de bloqueio é porque já foi reportada por ter algum problema e, portanto, as ferramentas de SCA ajudam a impedir a instalação de software problemático que se encontre na lista de bloqueio [2].

A maioria das ferramentas e técnicas de SCA são aplicadas em ambientes de desenvolvimento de software onde é usado software *open-source* e grátis (*Free and Open-Source Software* - FOSS). A maioria dos componentes FOSS usados são para uso interno, tais como compiladores, ambientes de desenvolvimento integrados, ferramentas de construção, ferramentas de teste e bibliotecas. Como tal, é necessário controlar os componentes FOSS para entender a atribuição e redistribuição das obrigações [3]. A utilização de SCA no FOSS inclui a verificação da conformidade de licenças, identificação de vulnerabilidades e qualidade de atributos.

1.2 Objetivo do trabalho

O objetivo deste trabalho é criar um procedimento de comparação de ferramentas de SCA no âmbito do estágio efetuado para a empresa Critical Software, uma empresa cuja atividade principal é o desenvolvimento de software, com o objetivo de avaliar as ferramentas de SCA atualmente existentes. Este estudo tem o propósito de dar a conhecer a SCA e apresentar um procedimento que ajude na escolha de uma ferramenta dentro do ciclo de aquisição de software com base nas especificações e características das ferramentas a serem consideradas na aquisição.

Neste trabalho desenvolver-se-á um procedimento e métricas/parâmetros que permitam avaliar ferramentas de SCA. Como consequência disso, este trabalho vai propor um

procedimento e aplicá-lo para analisar diversas ferramentas já existentes. Como validação do resultado do procedimento irão ser testadas as ferramentas com os melhores resultados, dentro das métricas definidas, em software de teste, onde a empresa Critical Software irá fornecer casos reais de software.

Definir um procedimento deste género traz benefícios para uma empresa em termos de eficácia na identificação dos candidatos, poder avaliar objetivamente os prós e contras, assim como atributos de custo-benefício.

1.3 Planeamento do trabalho

Nesta secção é apresentado o planeamento de trabalho feito para o primeiro e segundo semestre.

1.3.1. Planeamento do primeiro semestre

- Desenvolvimento do resumo e introdução (2 semanas)
- Contexto e trabalhos relacionados (11 semanas)
    - Revisão da literatura (2 semanas)
    - Contextualização da SCA (2 semanas)
    - Aplicações da SCA (2 semanas)
    - Ataques a cadeias de fornecimento de Software (3 semanas)
    - FOSS (2 semanas)
- Ferramentas de SCA (11 semanas)
    - Requisitos básicos de ferramentas de SCA (2 semanas)
    - Características das ferramentas de SCA (3 semanas)
    - Tratamento de componentes FOSS (3 semanas)
    - Métricas de avaliação, procedimento (3 semanas)

Ao longo do primeiro semestre o planeamento definido inicialmente sofreu alterações. Assim, o desenvolvimento dos tópicos não se encontra pela sua ordem exata como se pode observar no diagrama de Gantt apresentado.

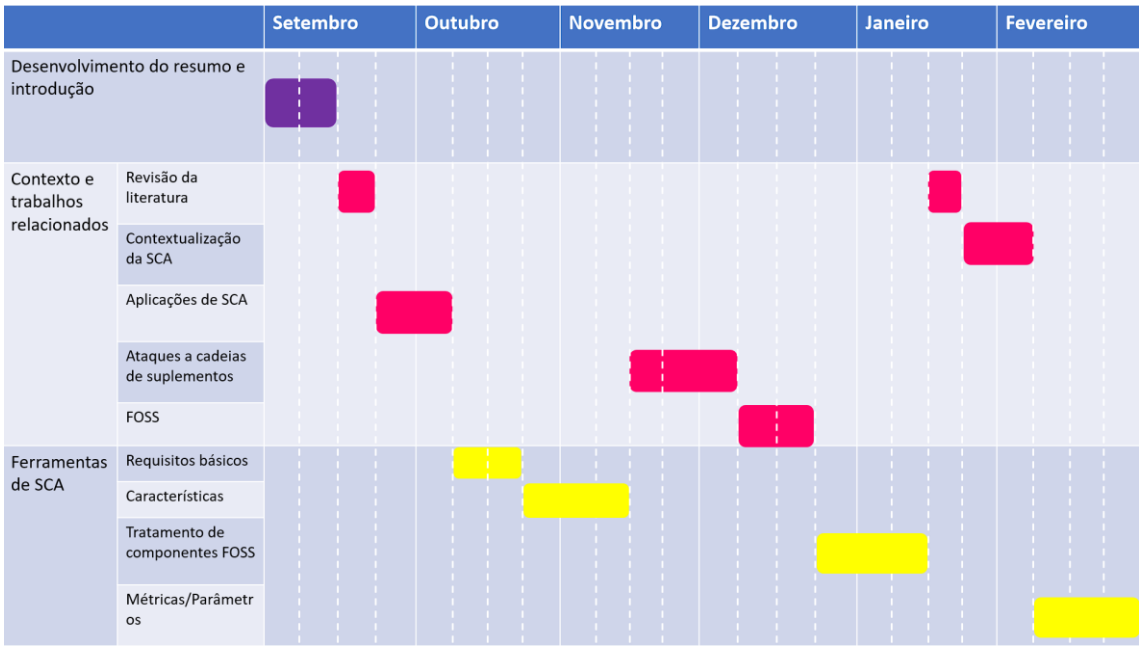

Figura 1 - Diagrama de Gantt que reflete a execução no 1º semestre

1.3.2. Planeamento do segundo semestre

- Desenvolvimento da Metodologia (16 semanas);
  - Recolha de informação das ferramentas e definição dos critérios de avaliação (15 semanas);
  - Proposta AHP (1 semana);
- Aplicação do procedimento (5 semanas);
  - Atribuições de pesos aos tópicos, critérios e respostas (3 semanas);
  - Avaliação individual das ferramentas e discussão de resultados (2 semanas);
- Conclusão e detalhes finais (3 semanas).

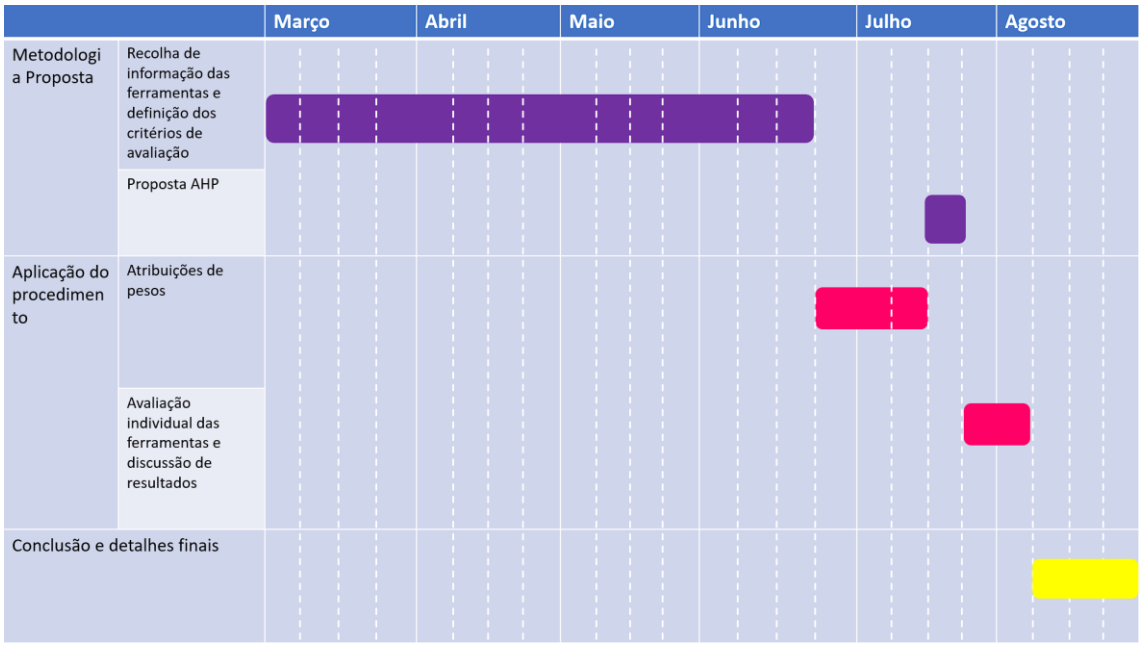

Figura 2 – Diagrama de Gantt que reflete a execução no 2º semestre

Primeiramente, o início do trabalho no segundo semestre iniciou-se a metade de fevereiro, no entanto, por motivos de saúde, a entrega foi adiada para setembro. A Figura 2 apresenta o diagrama de Gantt a partir de março, sendo que apenas a final de maio a minha saúde melhorou.

A parte mais demorada do trabalho no 2º semestre foi a recolha de informação das ferramentas e definição dos critérios de avaliação. Para além de já se esperar que a mesma fosse prolongada dada a sua natureza de extensa pesquisa, por motivos de saúde demorou ainda mais a ser realizada. Esta fase incluiu também a presença de membros da Critical Software no qual foram analisadas conjuntamente os critérios deste procedimento.

De seguida, realizou-se a atribuição de pesos a tópicos, critérios e respostas do procedimento. Ao concluir esta etapa foi apercebida a limitação da atribuição de pesos e por conseguinte inclui-se a proposta AHP. Após isto são efetuadas todas as avaliações das ferramentas utilizando o procedimento criado.

Por fim, é feita a discussão de resultados e a conclusão deste trabalho, aproveitando o tempo final para se fazer pequenos ajustes ao trabalho.

1.4 Estrutura do documento

O restante documento encontra-se estruturado da seguinte maneira: no Capítulo 2, são apresentados tópicos relacionados com o contexto e trabalhos relacionados. Este capítulo tem o intuito de apresentar a informação já existente em artigos científicos e académicos acerca deste tema, assim como contextualizar a SCA. É apresentada a contextualização da SCA no desenvolvimento de software, de modo a clarificar o seu enquadramento no ciclo de desenvolvimento. Também é realizada uma comparação da SCA com outras atividades de segurança, para compreender a sua especificidade. De seguida, são apresentadas as principais aplicações da SCA existentes atualmente, entendendo os principais pontos de atuação das ferramentas. Também são apresentados ataques recentes a cadeias de fornecimento de software de modo a realçar a importância da SCA e as consequências que estes ataques acarretam. Por fim neste capítulo, é exibido um tópico dedicado a componentes

de FOSS (*Free and Open Source Software*) por ser um tema com bastante presença e relevância dentro da SCA.

No Capítulo 3 aborda-se as ferramentas de SCA, onde se começa por especificar os requisitos básicos e essenciais das ferramentas, de seguida, são exibidas as características das ferramentas para melhor entendimento das suas capacidades e funcionamento. Volta-se depois ao tópico de FOSS, mas agora explicando o seu funcionamento nas ferramentas e as atividades necessárias ao tratamento destes componentes. Para finalizar este capítulo são apresentadas as métricas definidas para posteriormente serem utilizadas na componente prática definida no segundo semestre.

O Capítulo 4 apresenta a metodologia proposta, isto é, os critérios de avaliação considerados e também a proposta AHP que serve de auxílio à metodologia criada.

No Capítulo 5 tem-se uma aplicação do procedimento demonstrativa, onde são atribuídos os pesos dos tópicos, critérios e das respostas aos mesmos, por conseguinte é feita a avaliação de cada uma das ferramentas selecionadas. Por fim, são discutidos os resultados deste procedimento.

Por último, o Capítulo 6 apresenta a conclusão sobre o trabalho realizado.

Capítulo 2
Contexto e trabalhos relacionados

Neste capítulo são apresentados tópicos de contexto desta dissertação de modo a entender a importância deste tema, assim como trabalhos relacionados. Como tal, é iniciado com a revisão de literatura, na Secção 2.1, com o objetivo de tomar conhecimento de artigos e trabalhos relacionados e sobre o que estes abordam especificamente, podendo assim comparar, relacionar e contrastar diferentes teorias e descobertas.

De seguida, na Secção 2.2, discutem-se componentes *open source* e a relevância da sua presença em praticamente todos os projetos de software, sendo um ponto fulcral o modo como as ferramentas de SCA operam sobre esses componentes.

A Secção 2.3, aborda a contextualização da SCA no desenvolvimento de software, isto serve para entender o papel da SCA e onde esta se encaixa. Ainda dentro deste tópico, são feitas comparações com outras atividades de segurança para realçar a singularidade da SCA.

Na Secção 2.4, é explicado o processo de aquisição de software e onde o procedimento criado se insere no mesmo.

Na Secção 2.5, são mostradas as principais aplicações da SCA, de modo a entender os objetivos primários desta atividade.

Por fim, a Secção 2.6 apresenta recentes ataques a cadeias de fornecimento de software, para salientar a importância e relevância da SCA no combate a esses tipos de ataques e denotar as repercussões que os mesmos apresentam.

2.1 Revisão da literatura

Comparando o objetivo deste trabalho com outros artigos científicos e trabalhos académicos, o mais similar em termos de objetivo (comparação de ferramentas segundo um procedimento) é um trabalho de Imtiaz et al. [4], que no entanto possui limitações na seleção das ferramentas, pois destina-se a aplicações web de relatórios eletrónicos hospitalares. Outro trabalho de Philippe Ombredanne [3], aprofunda o conhecimento de ferramentas de SCA para software *open source*, focado na explicação e funcionamento do tratamento de componentes *Free and Open Source Software*. Este artigo é de bastante relevância teórica, mas limita-se ao tratamento de componentes *open source*. Ainda relacionado com a comparação de ferramentas, Haddad [5], [6] apresenta parâmetros de avaliação de ferramentas de SCA. Alguns destes parâmetros são subjetivos. No entanto, como este trabalho tem por objetivo apresentar algo diferente do que os artigos revistos apresentam, vai ter por base algumas das informações dos mesmos, tendo o intuito de determinar um procedimento e métricas mais objetivas para a avaliação das ferramentas.

A restante literatura identificada serve de apoio à compreensão e ao conhecimento aprofundado de tópicos relacionados à SCA. Tem-se um trabalho académico de Dimov e Dimitrov [7] que é útil para contextualização e compreensão de SCA comparativamente a outras abordagens de segurança. Também foi desenvolvido um artigo científico por uma equipa da empresa Veracode [8] sobre um meio inovador e modular de combinar *call graphs* (um gráfico de controlo de fluxo, que representa chamadas de relações entre sub-rotinas num programa) derivados de análises estáticas e dinâmicas para melhorar a eliminação de falsos positivos. Para além dos trabalhos mencionados, Chen et al. [9] apresentam um sistema de

machine learning desenvolvido para identificar as bibliotecas correspondentes a cada vulnerabilidade da NVD (*National Vulnerability Database*), que resolve pela primeira vez a identificação de nomes de bibliotecas a partir de dados da NVD como XML (*Extensible Markup Language*) e implementa a solução em sistemas de produção completos. Por fim, temos o trabalho de Steve Mansfield-Devine [10] que aborda o tema de "achar espaço para a segurança em DevOps", afirmando estar a ser esquecido. Nesse artigo, é mencionada a utilização de atividades de segurança, incluindo SCA, a frequência com que este é utilizada em DevOps e as dificuldades que apresenta nos sistemas de DevOps atuais, onde muitas vezes o fator de segurança é ignorado.

2.2 FOSS – *Free and Open Source Software*

Dada a crescente adoção de uso de código *open source*, juntamente com a publicidade de recentes infrações e ataques informáticos a cadeias de fornecimento de software, é de esperar o aumento do crescimento desta fonte de risco. O código *open source* está a desempenhar um papel fulcral na transformação digital que se está a tornar cada vez mais evidente, e não há nenhum motivo para se supor que estas tendências mudarão.

As organizações estão a usar *open source* para as ajudar a competir no mercado, e consequentemente, começa a haver um entendimento crescente de que devem controlar esta utilização, gerindo e mitigando os riscos que a acompanham. As ferramentas de SCA são fundamentais para ajudar as organizações a alcançar estes objetivos [11].

Atualmente, os programadores utilizam grandes quantidades de código de terceiros para construir as suas aplicações, uma vez que a reutilização do código aumenta significativamente a produtividade e reduz os custos de desenvolvimento. Dado o veredicto de que até 80% das aplicações típicas são de código de terceiros, há necessidade de ferramentas para gerir o risco do código *open source* [8].

Segundo a empresa Veracode [12], 44% das aplicações contém vulnerabilidades críticas em componentes *open source*. Ainda existem muitas empresas que não possuem uma maneira viável de serem notificadas quando são encontradas vulnerabilidades do tipo *zero-day* ou quando são disponibilizados *patches*. Consequentemente, isto faz com que os vetores de ataque em componentes FOSS perdurem, mais do que deveriam [13].

Como tal, o tratamento específico de componentes FOSS (*Free and Open Source Software*) torna-se uma parta fundamental nas ferramentas de SCA, sendo que neste trabalho é abordada uma secção específica de como as ferramentas operam sobre estes componentes e quais as técnicas utilizadas.

2.3 Contextualização da SCA no desenvolvimento de software

Nesta secção contextualiza-se a Análise de Composição de Software em desenvolvimento de software. Para tal, é necessário saber que o *Software Development Life Cycle* (SDLC) possui cinco fases num dos seus modelos ágeis, sendo estas, como se pode ver na Figura 1:

1. Fase de análise e definição requisitos
2. Fase de design do sistema e do software
3. Fase de desenvolvimento
4. Fase de testes
5. Fase de integração

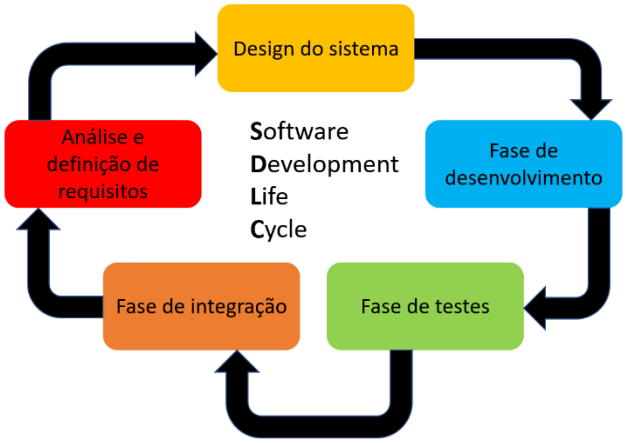

Figura 3 – Ciclo de vida de desenvolvimento de software (SDLC)

Para além deste modelo, também é utilizado o modelo em Cascata tendo as seguintes fases iterativas, como se pode observar na Figura 2:

1. Análise de Requisitos
2. Design de Software
3. Implementação
4. Integração (Verificação)
5. Teste e depuração (Verificação)
6. Manutenção de software

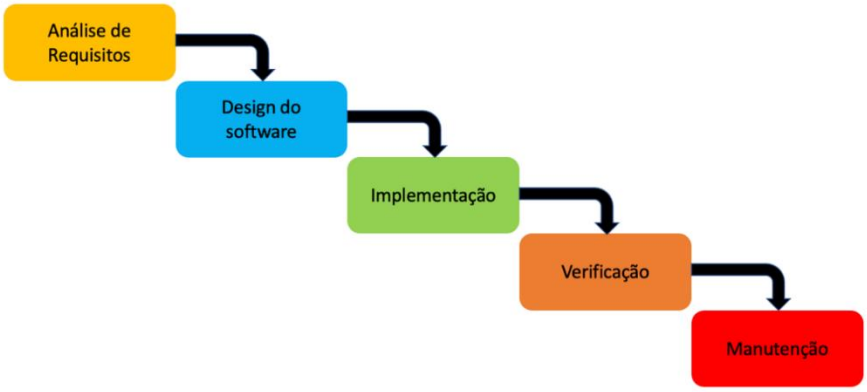

Figura 4 - Modelo em Cascata

Apresenta-se de seguida um excerto do trabalho desenvolvido em [7], onde se consideram seis categorias de ferramentas de segurança de acordo com o modelo em cascata do SDLC.

"- Ferramentas aplicadas durante a definição de requisitos;
- Ferramentas aplicadas durante o design do software;
- Ferramentas aplicadas durante a implementação do software
- Ferramentas aplicadas durante a fase de testes;

- *SAST (Static Application Security Testing);*
- *DAST (Dynamic Application Security Testing);*
- *IAST (Interactive Application Security Testing);*

- *SCA (Software Composition Analysis);*

- Ferramentas aplicadas durante manutenção e usabilidade do sistema;

- *Scanners de malware;*
- *Scanners de segurança de websites;*
- *Ferramentas de proteção em tempo real (como por exemplo, firewalls e application lockers);*

- Ferramentas utlizadas para aprender sobre segurança

- *Esta categoria é considerada horizontal para as outras, pois supõe-se que uma boa educação, níveis superiores de segurança sejam alcançados em todas as fases do SDLC."* [7]

Esta lista revela-se pouco eficaz na prática. Atualmente os testes de segurança já não são realizados perto do fim do SDLC. Quanto mais tarde forem encontrados problemas de conformidade ou vulnerabilidades, mais dispendioso e demorado é resolvê-los. Como tal, há mais tendência em adotar as soluções de análise e composição de software com a conduta de CI/CD (*Continuous Integration*/*Continuous Delivery*), como parte do processo de construção. Ou seja, sempre que uma nova mudança é introduzida e uma compilação é efetuada, um novo *scan* também é efetuado. Isto proporciona *feedback* imediato aos programadores, desde cedo no ciclo de desenvolvimento, reduzindo os custos de problemas que possam surgir [14].

2.3.1. Comparação de SCA com outras atividades de segurança

A SCA tradicionalmente enquadra-se nas atividades aplicadas durante a fase de testes do SDLC, juntamente com DAST (*Dynamic Application Security Testing*), SAST (*Static Application Security Testing*) e IAST (*Interactive Application Security Testing*). Como tal, é fundamental entender as suas diferenças. As ferramentas de DAST, SAST e IAST analisam código proprietário e também possuem tendência a serem utilizadas mais tarde no SDLC. No entanto, é comum para as organizações implementarem uma solução SCA e uma ou mais soluções de teste de código proprietário [14].

SAST – É um tipo de *whitebox testing* que analisa o código fonte, o código byte e os binários, de dentro para fora, enquanto os componentes estão estáticos, para falhas de codificação e de design que sugerem possíveis vulnerabilidades de segurança. É normalmente implementada durante o desenvolvimento e na garantia de qualidade. Também costuma ser integrada em servidores de integração contínua e em ambientes de desenvolvimento integrados (IDEs). Os *scans* da SAST têm fundamento num conjunto de regras pré-definidas que determinam os erros de código fonte que precisam de ser avaliados. Estes *scans* podem identificar vulnerabilidades de segurança comuns, como *SQL injection*, validação de dados de entrada e *stack* e *buffer overflows* [15].

DAST – Este tipo de teste é do tipo *blackbox testing* que procura vulnerabilidades de segurança e falhas arquiteturais simulando ataques externos a uma aplicação enquanto a mesma está em execução. Tenta penetrar a aplicação a partir de vetores exteriores, como interfaces expostas com falhas e vulnerabilidades, sendo que não possui acesso ao código fonte e apenas descobre vulnerabilidades através de ataques externos. A parte dinâmica desta abordagem é o facto da aplicação estar em execução aquando da realização dos testes, enquanto que a DAST pode ser utilizada em produção, a testagem geralmente é feita em ambientes de garantia de qualidade [15].

IAST – Esta abordagem de teste analisa o código fonte de uma aplicação após a *build* em ambiente dinâmico. Ou seja, os testes ocorrem em tempo real, enquanto a aplicação está em execução, frequentemente em ambiente de garantia de qualidade ou de teste. Estes testes são capazes de identificar a linha problemática do código e notificar o programador da sua remediação imediata. Embora a SAST analise também diretamente o código, a IAST fá-lo após a *build* num ambiente dinâmico através de instrumentação do código. Isto é, agentes e sensores são implementados na aplicação, analisando o código para identificar vulnerabilidades. Esta abordagem é facilmente integrada na conduta CI/CD, é escalável e pode ser automatizada ou executada manualmente [15].

SCA – As ferramentas de análise de composição de software realizam *scans* automáticos à base do código de uma aplicação de modo a encontrar as dependências que o mesmo possui com outro software/bibliotecas, principalmente *open source.* Isto inclui a identificação de todos os componentes *open source*, a informação de conformidade de licenças e vulnerabilidades de segurança. As ferramentas de SCA dão prioridade às vulnerabilidades de componentes *open source* e, idealmente, aconselham remediações para as ameaças de segurança existentes [15].

A Tabela I sintetiza as diversas características dos diferentes tipos de testes de segurança.

Tabela I – Testes de segurança de software [15]

		Cobertura	Baixos falsos-positivos	Explorabilidade	Visibilidade do código	Recomendações de remediação	Integração no SDLC	Suporte de plataforma ampla
Ferramentas de *scanning* de segurança	SAST	X			X	X	X	X
	DAST		X	X				X
	IAST		X	X	X	X	X	
	SCA	X		X	X	X	X	X

De acordo com Simon Roe [16], as abordagens de teste ao longo do SDLC, são aplicadas conforme apresentado na Figura 3.

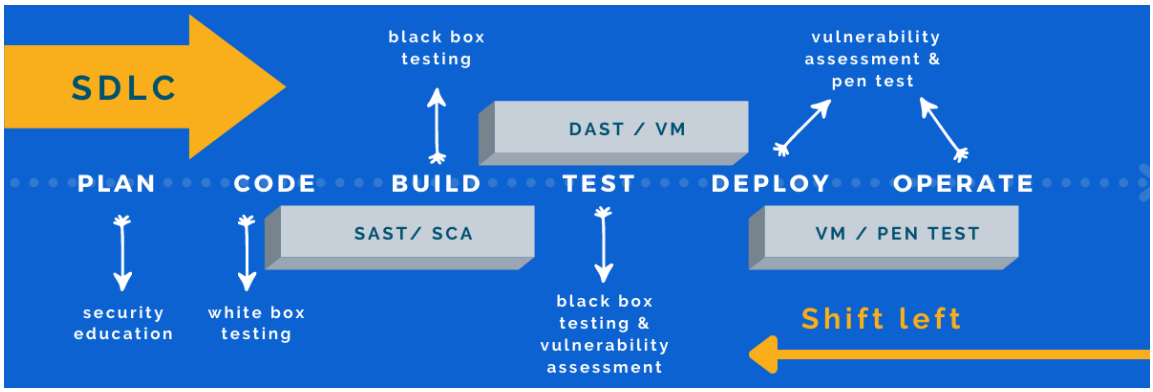

Figura 5 - Abordagens de segurança no SDLC

Como é possível observar, as diferentes abordagens complementam-se, ou seja, ter-se-ão melhores soluções de segurança combinando diferentes abordagens de testes de segurança ao longo do SDLC.

O que distingue a SCA de outras ferramentas de segurança é a sua crescente presença e importância no SDLC, onde a utilização de software *open source* é cada vez maior. Uma solução de SCA permite a gestão segura do risco da utilização de software *open source* ao longo de toda a cadeia de fornecimento de software [17].

2.4 Processo de aquisição de software

A SCA é uma atividade que utiliza ferramentas e como todo o software utilizado numa empresa devem ser adquiridas através do processo já utilizado para esse tipo de aquisição. Cada empresa possui o seu próprio processo de aquisição de software específico. O procedimento criado neste trabalho entra como um passo no processo de aquisição com o propósito de ajudar a selecionar as ferramentas de SCA mais adequadas. Para tal, são definidos pesos técnica e financeiramente para avaliar as ferramentas.

De acordo com ISO/IEC/IEEE as práticas recomendadas para aquisição de software possuem oito passos [18], sendo estes listados abaixo e esquematizados na Figura 6.

1. Planeamento da estratégia de aquisição de software
2. Definir os requisitos de aquisição e software
3. Identificar potenciais fornecedores
4. Preparar requisitos contratuais
5. Avaliação de propostas e escolha de um fornecedor
6. Gestão do desempenho do fornecedor
7. Aceitação do software
8. Avaliação do processo e identificação de oportunidades de melhoria

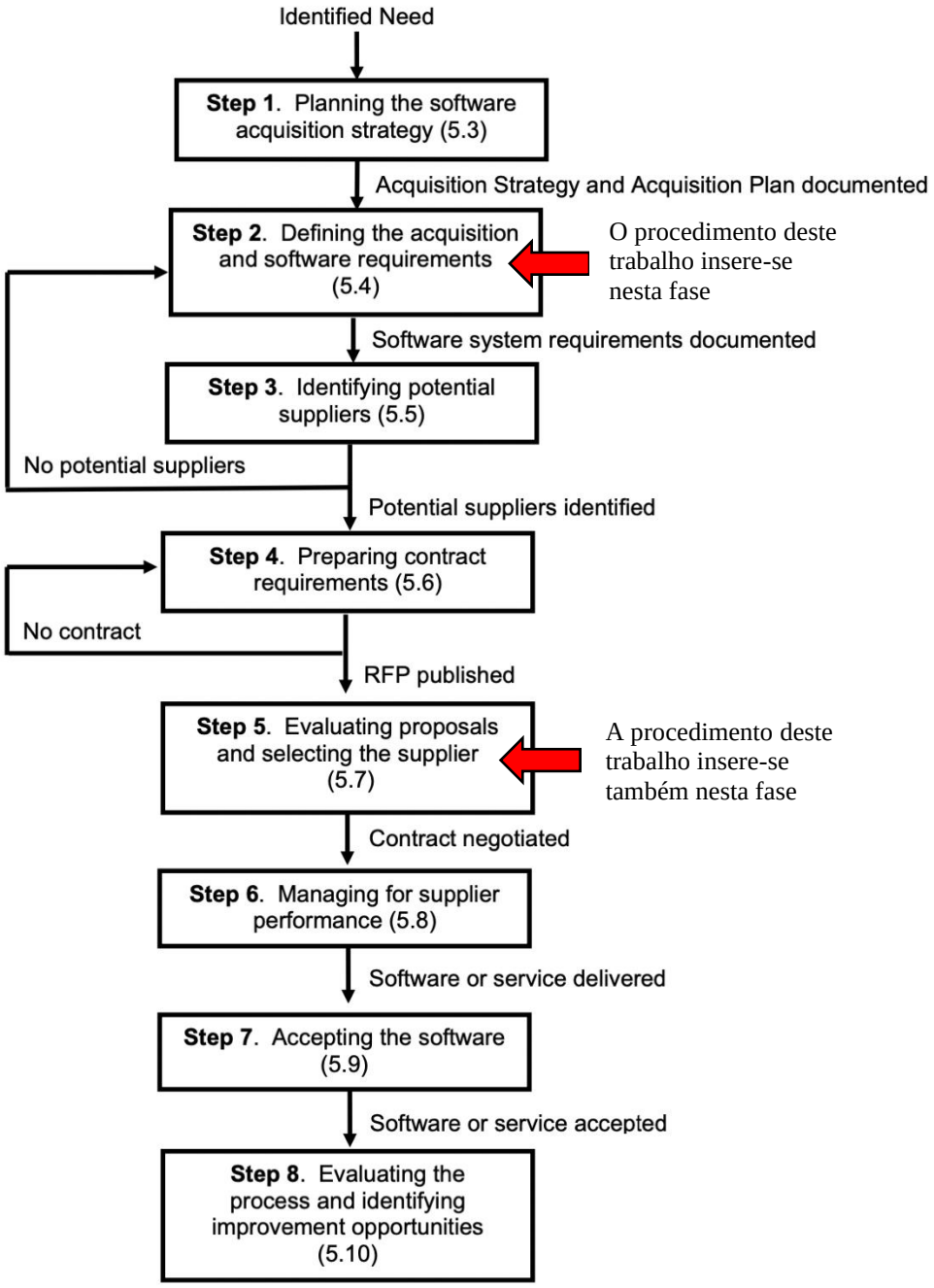

Figura 6 - Oito passos envolvidos na aquisição de software [15]

Seguindo o modelo apresentado, o procedimento deste trabalho é aplicado na Fase 5, de modo a ajudar a desempatar as propostas e a escolher o fornecedor final.

2.5 Aplicações da SCA

As ferramentas de SCA têm três grandes objetivos ao proteger cadeias de fornecimento, tal como ilustrado na Figura 7: proteger o software proprietário de uma dependência com vulnerabilidades, impedir a criação de dependências com malware e por fim, fazer com que as licenças das dependências de software estejam em conformidade com a natureza do software proprietário. De seguida discute-se cada um destes objetivos com mais detalhe.

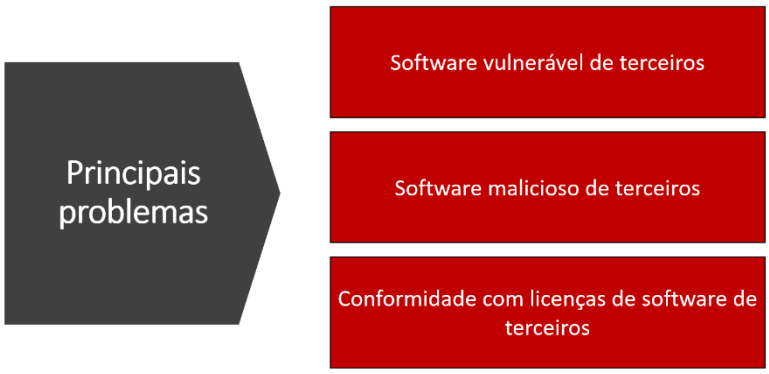

Figura 7 – Principais problemas solucionados por SCA

2.5.1. Software vulnerável de terceiros

O software do qual se possui uma dependência pode possuir vulnerabilidades, principalmente quando for software mais complexo. Outro fator é a quantidade de contribuidores para o seu desenvolvimento, que pode afetar também a quantidade de vulnerabilidades no software. Ao usar um software vulnerável de terceiros, transportamos também essas vulnerabilidades para o software que depende dele [2].

2.5.2. Software malicioso de terceiros

Por vezes podemos colocar o nosso software em risco ao criar uma dependência de software malicioso. Estes softwares maliciosos estão feitos de maneira a ludibriar os desenvolvedores. Exemplos de táticas nesse sentido são *Typosquatting* e Cavalos de Troia. *Typosquatting* é uma técnica onde são registados domínios com nomes estrategicamente mal soletrados de sites conhecidos, como por exemplo "google.com" que é um domínio extremamente conhecido, e o domínio malicioso seria "googl.com". Ao acedermos a estes sites podemos fazer o download de software malicioso ou dar informações pessoais a agentes malfeitores que podem comprometer o nosso software. Um Cavalo de Troia é software que aparenta ser legítimo e de grande utilidade, mas que na verdade é malware [2], [19].

2.5.3. Conformidade com licenças de software de terceiros

Uma organização possui as suas próprias premissas consoante o software que está a desenvolver e as suas próprias normas e regras de empresa. Como tal, tem de decidir que licenças são aceitáveis para o seu software, colocando-as numa lista. Ao fazer composição de software, tem de se verificar se estas dependências estão de acordo com a lista feita pela organização. As ferramentas de SCA fazem esta verificação de forma automática. Como exemplo de licenças mais usadas temos: MIT, BSD, Apache 2.0, LGPL, GPL e MLP [2].

2.6 Ataques a cadeias de fornecimento de software

Esta secção tem o intuito de salientar a importância de SCA e dar a compreender como o adversário pode explorar vulnerabilidades de modo a comprometer cadeias de fornecimento de software.

2.6.1. Tipos de ataques a cadeias de fornecimento

Existem 3 tipos de ataques às cadeias de fornecimento [20]:

- Injeção de código malicioso em bibliotecas de terceiros que as usam no seu software.
- Comprometer uma aplicação legítima e confiável que as organizações usem como parte da sua tecnologia.
- Atingir atualizações de software ou de servidores de atualizações que comprometem o utilizador final.

2.6.2. Exposição de ataques recentes

Nesta subsecção são identificados e discutidos alguns ataques recentes feitos a cadeias de fornecimento de software ordenados segundo a sua linha temporal na (Figura 8).

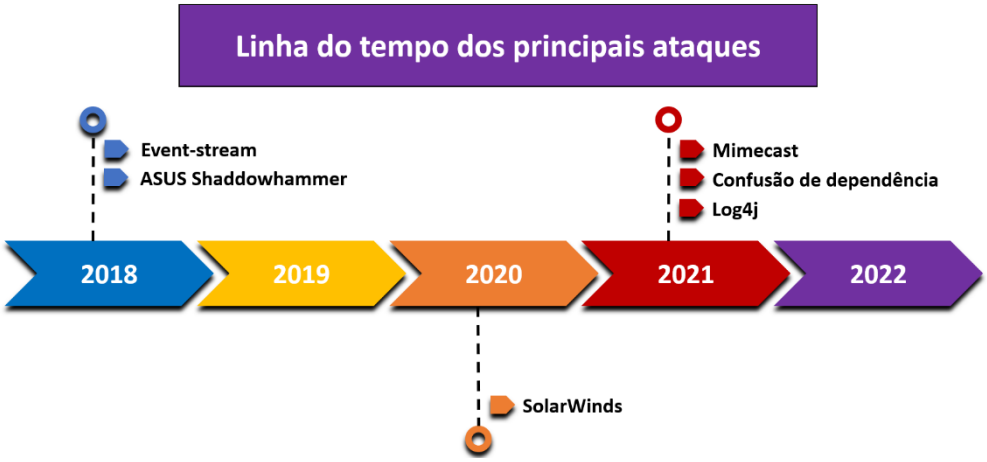

Figura 8 - Linha do tempo dos principais ataques a cadeias de fornecimento de software

Event-stream, 2018: Este ataque foi feito num repositório Github, onde foi injetado malware na dependência *flatmap-stream* que caiu em desuso e que fazia parte de um pacote npm de *event-stream*. Este ataque está relacionado com aplicações que usam Javascript. O *event-stream* é um pacote npm que fornece vários auxílios para trabalhar com *streams* de uma aplicação de Node, no entanto, não está em desenvolvimento há alguns anos. O autor, *dominctarr*, tinha um grande número de projetos e não tinha mais uso para o *event-stream*, então o mesmo caiu no esquecimento. Um utilizador com o nome de *right9ctrl* ofereceu-se para assumir as tarefas de manutenção, o *dominictarr* aceitou e concedeu acesso ao GitHub e ao npm. Após concedido o acesso, *right9ctrl* adicionou uma dependência chamada *flatmap-stream*, com o fim de suportar um *flatmap* numa função de *event-stream*. Mais tarde, o mesmo sujeito removeu a dependência *flatmap-stream* e implementou diretamente a função *flatmap* dentro da *event-stream*. A este ponto nada parecia preocupante, pois estes tipos de mudanças não são considerados incomuns. A parte maliciosa foi colocada deliberadamente ofuscada no final da sua saída minificada (processo de remover elementos desnecessários e reescrever o código para reduzir o tamanho do arquivo) construída para não ser detetada. Esta tenta descodificar e executar *strings* do pacote de ficheiro 'test/data.js' que usa a descrição de um pacote *top-level* como a chave AES256. A *copay-dash* é uma famosa plataforma bitcoin que inclui uma dependência *event-stream*. O objetivo deste script é roubar carteiras bitcoin e enviá-las ao servidor do atacante. Para outros pacotes, isto produz um erro silencioso (pois a chave de descrição/decriptação está errada), mas para a *copay-dash* é um JavaScript válido que executa outra ronda de desencriptação e executa um script malicioso que rouba

informações da carteira. A quantidade de aplicações que usaram a dependência comprometida ainda é desconhecida [21], [22].

ASUS Shadowhammer, 2018: Este ataque foi chamado de ShadowHammer e teve como alvo os proprietários de computadores ASUS. Este ataque à cadeia de fornecimento foi feito através da ASUS Live Update Utility, que foi *hijacked* por *hackers* para instalar *backdoors* em milhares de computadores, graças a se ter conseguido comprometer o servidor de atualização de *live software* da empresa. O ASUS *Live Update Utility* vem pré-instalado na maior parte dos computadores ASUS e a sua função é atualizar automaticamente componentes como a BIOS (*Basic Input/Output System*), o UEFI (*Unified Extensible Firmware Interface*), *drivers* e aplicações. Dados relativos a 2017 apontam que a empresa ASUS era a 5ª maior vendedora de computadores no mundo, o que a torna um alvo bastante atrativo para ataques. O objetivo do ataque era atacar estrategicamente um conjunto de utilizadores que eram identificados pelos endereços MAC (*Media Access Control*) dos seus adaptadores de rede. Para atingir este objetivo, os atacantes codificaram uma lista de endereços MAC em amostras trojanizadas e essa lista foi utilizada para identificar alvos reais desta operação de larga escala. A partir de 200 amostras utilizadas foi possível identificar mais de 600 endereços alvo. Embora pareça que os atacantes só tinham como alvo esses 600 sistemas, o malware procurou os sistemas alvo através dos seus endereços MAC únicos e, uma vez num sistema, quando encontrado um desses endereços alvos, o malware chegava ao servidor de comando e de controlo que os atacantes operavam e instalava malware adicional nessas máquinas. Este ataque não foi detetado durante um longo período de tempo devido ao facto dos atualizadores trojanizados estarem assinados com certificados legítimos ("ASUSTeK Computer Inc."). Os atualizadores comprometidos foram alojados nos servidores oficiais. A Kaspersky Lab disse ter descoberto este ataque depois de ter adicionado uma nova funcionalidade de deteção a cadeias de fornecimento à sua ferramenta de *scan*, para apanhar fragmentos anómalos de código escondidos em código legítimo ou capturar o código que faz o *hijack* das operações normais da máquina. Segundo as estatísticas, mais de 57 mil utilizadores do antivírus Kaspersky descarregaram e instalaram a versão *backdoored* do ASUS *Live Update*. Investigações indicam que a utilização de um binário antigo com um certificado atual sugere que os atacantes tiveram acesso ao servidor onde a ASUS assina os seus ficheiros, mas não ao *build server* onde se compilam os novos certificados. Ou seja, como os atacantes usaram sempre o mesmo binário ASUS conclui-se que não tiveram acesso à total infraestrutura de assinatura da ASUS. Não é possível calcular o número real de utilizadores afetados. Contudo, estima-se que a escala real do problema afetou mais de um milhão de utilizadores por todo o mundo. Este ataque mostra que o modelo de confiança baseado em nomes de fornecedores conhecidos e validação de assinaturas digitais não pode garantir proteção contra malware [21], [23], [24].

SolarWinds, 2020: Este é considerado o maior ataque a cadeias de fornecimento de software até hoje. Este ataque teve origem a partir de um *backdoor*, SUNBURST, que foi injetado na ferramenta de atualização da aplicação de gestão de TI da rede Orion, da SolarWinds. Especialistas dizem também que este ataque pode ter tido origem noutro vetor inicial, um desvio de autenticação multifator, feito acedendo à chave secreta do servidor Outlook Web App. Os atacantes conseguiram ganhar espaço na rede da empresa dos fornecedores e foram capazes de alterar o código que compõe uma das DLLs (*Dynamic-link library*) empacotadas com o software. Esta passou pelo *build* e foi assinada com um certificado legítimo. A plataforma Orion é o núcleo do portfólio de gestão de TI da SolarWinds, fornecendo uma

arquitetura estável e escalável que realiza a recolha, processamento, armazenamento e apresentação de dados. Esta plataforma também fornece recursos comuns, como contas de utilizadores e grupos, visualizações, painéis, alertas, que podem ser usados em todos os produtos da *Orion Platform* e ser acedidos a partir do *Orion Web Console*. Os atacantes que invadiram a SolarWinds usaram as atualizações de software da empresa para instalar o *backdoor* em sistemas pertencentes a governos, entidades militares e de defesa e várias empresas do setor privado. Julga-se que o ataque tenha sido realizado da seguinte maneira: a SolarWinds fabrica um *software* de gestão de rede, chamado Orion, que é usado amplamente por agências governamentais e empresas Fortune 500. Como qualquer fornecedor de *software*, este envia atualizações regulares do *software* para os clientes. Os atacantes inseriram *software* malicioso em atualizações feitas pela empresa entre março e junho de 2020. Cerca de 18 mil clientes descarregaram essas atualizações que funcionavam como cavalos de Troia, que aguardavam instruções dos atacantes. Alguns destes clientes receberam instruções provenientes dos atacantes dando-lhes uma maneira de roubar dados, e com isso conseguiram aceder a emails, descarregar software e realizar reconhecimento de redes. Este ataque iludiu tanto as medidas de segurança dos Estados Unidos da América que nem foi descoberto por oficiais de inteligência, mas por um alerta de segurança automático enviado a um funcionário da FireEye que tinha sido comprometido. Este aviso, que também tinha sido enviado à equipa de segurança da empresa, informava que alguém tinha usado as credenciais do funcionário para efetuar login na rede privada da empresa a partir de um dispositivo não reconhecido. Este é o tipo de alertas que na maioria das vezes é ignorado, mas por sorte a FireEye decidiu prestar atenção ao mesmo. A 12 de dezembro a FireEye informou a SolarWinds que estava a fornecer *software* que inconscientemente enviava software contaminado desde março, e também se informou o governo dos Estados Unidos da Améria. A SolarWinds diz ter lançado uma rápida correção para o problema de segurança existente, mas apenas cortar o ponto de acesso dos atacantes não é o suficiente, pois eles já se tinham infiltrado em muitos sistemas graças àquele primeiro ponto de entrada. A prioridade deste ataque foi a furtividade e não causar destruição. Isto permitiu que o ataque não fosse detetado por um longo período de tempo. Não se sabe ao certo como é que os atacantes obtiveram acesso aos sistemas da SolarWinds para introduzir o código malicioso. A empresa disse que as suas contas de email da Microsoft tinham sido comprometidas e que isso pode ter sido usado para recolher mais informações das ferramentas do Office da empresa. Para obter uma noção da escala deste ataque a empresa FireEye diz ter recebido ligações de clientes que acreditam ter sido afetados por este ataque, embora nunca tenham instalado o software da SolarWinds nas suas redes. Dentro das organizações afetadas por este ataque temos partes altamente seguras do governo dos Estados Unidos da Améria, como o Departamento de Energia, Segurança Interna e o Departamento de Estado [21], [25], [26].

Mimecast, 2021: Mimecast é uma empresa focada em segurança em nuvem e relatou que atacantes comprometeram um certificado usado pelo fornecedor para autenticar os seus serviços (*Mimecast Sync and Recover*, *Continuity Monitor* e *Internal Email Protect* - IEP) no *Microsoft 365 Exchange Web Services*. Segundo as estatísticas, apenas 10% dos clientes do Mimecast usam aplicações que dependem do certificado comprometido e apenas alguns desses clientes foram afetados (cerca de 2%), pois há indicações que apenas um número baixo de clientes M365 foram alvo, tendo sido todos contactados para remediar o problema. Como precaução, a empresa pediu aos clientes que usavam o locatário M365 para eliminarem a ligação existente e restabelecer uma nova com uma conexão de base de certificado usando o novo certificado disponibilizado. Ao tomar esta ação não há impactos no fluxo de mensagens de entrada ou saída ou na verificação de segurança associada. Os atores maliciosos parecem

ter comprometido o certificado, no entanto não existem detalhes públicos publicados. Os certificados comprometidos foram usados para conceder aos serviços do Mimecast acesso a servidores de email e contas hospedadas pela Microsoft. O email é uma fonte rica em informações para os invasores. Contém grandes quantidades de informações confidenciais de empresas e pode ser usado para realizar outro tipo de atividades maliciosas como *phishing* [21], [27], [28].

Confusão de dependência, 2021: Um investigador em segurança informática foi capaz de *hackear* sistemas pertencentes à Apple, Microsoft, PayPal, Uber, Tesla, Shopify, Netflix e Yelp utilizando uma nova técnica de ataque à cadeia de fornecimento de software. O *ethical hacker* Alex Birsan detalhou a 9 de fevereiro de 2021 como obteve acesso aos sistemas internos dos alvos que definiu ao explorar uma vulnerabilidade chamada de "confusão de dependência". Esta técnica permite que um invasor execute malware nas redes de uma empresa substituindo pacotes de dependências usados de forma privada por pacotes públicos malignos com o mesmo nome. As empresas tanto usam dependências privadas como públicas e as dependências públicas podem ser enviadas para repositórios GitHub *open-source*, o que significa que podem conter código malicioso. O investigador diz que esta ideia de pesquisa surgiu enquanto estudava um ficheiro Node.js destinado a uso interno do PayPal, que foi partilhado com o seu amigo investigador Justin Gardner. O código continha dependências privadas e públicas e apercebeu-se que alguns dos nomes dos pacotes privados não se encontravam no NPM (*Node Package Manager*) Registry público. Foi nesta altura que Alex Birsan levantou a questão do que aconteceria se código malicioso fosse enviado para o NMP com estes nomes. Birsan testou esta teoria enviado código malicioso para registos públicos de dependência do NMP com o nome dos pacotes privados que encontrou no GitHub, sites de *hosting* de *packages* e fóruns online. Nenhum destes pacotes se encontrava listado como público. Com isto descobriu que se um pacote dependente estivesse listado no mesmo nome em ambos os repositórios (público e privado) a versão pública ganharia prioridade e seria usada. Birsan disse que o NPM permite que um código arbitrário seja executado automaticamente na instalação do pacote, permitindo criar um pacote Node que recolha informações básicas sobre cada máquina em que esteja instalado por meio do *script* de pré-instalação. De seguida, enviou de volta as informações para si mesmo, usando exfiltração de DNS para evitar a deteção dos desenvolvedores. O investigador testou a mesma técnica criando pacotes públicos RubyGems e Python ao portar o seu código malicioso para as dependências, recriando assim o ataque. Esta vulnerabilidade foi detetada em mais de 35 organizações nas três linguagens de programação testadas. Posteriormente, o problema foi corrigido pela Apple, Yelp e Microsoft, que lançou um comunicado detalhado sobre como mitigar o risco ao usar pacotes privados [21], [29], [30].

Log4j, 2021: Cerca de 35 mil pacotes Java, que corresponde a 8% do repositório Central Maven [31] (um dos pacotes Java de maior importância) foram afetados pela vulnerabilidade "*Remote Code Injection in Log4j*" que teve grandes repercussões na indústria de software [32], [33]. As vulnerabilidades apresentadas permitem que um invasor possa executar código remoto ao explorar o recurso JNDI (*Java Naming Directory Interface*) exposto pela biblioteca Log4j. Este recurso encontra-se habilitado em muitas versões desta biblioteca e permite que mensagens registadas contenham *strings* formatadas que referenciem informação externa. A contagem de dependências diretas ronda os 7 mil artefactos afetados, mas a maioria afetada corresponde a dependências indiretas. Como aplicações afetadas temos, por exemplo, Twitter, Amazon, Microsoft, Apple, IBM, Oracle, Cisco, Google e Minecraft, e, por

conseguinte, centenas de milhares de utilizadores destas aplicações. Já foram registadas pelo menos 3,700,000 tentativas de *hacking* ao explorar esta vulnerabilidade, sendo 46% desses ataques conduzidos por grupos maliciosos conhecidos. O grupo de voluntários Apache Software Foundation foi alertado a 24 novembro de 2021, após um membro de segurança em nuvem da Alibaba a ter descoberto. Para remediar esta vulnerabilidade, a Apache já lançou atualizações que contém as correções e a Microsoft lançou um aviso a encorajar clientes a contactar fornecedores de aplicações de software para confirmar se estão a usar a linguagem de programação Java. Para aqueles que não conseguem realizar já a atualização para corrigir a vulnerabilidade, a Cybereason lançou uma "vacina" para afastar temporariamente os agentes malfeitores [31], [34], [35].

2.6.3. Conclusões destes tipos de ataques

Os ataques à cadeia de fornecimento de software podem ser bastante devastadores, pois podem permanecer desconhecidos por muito tempo. Isto leva tipicamente a ataques do tipo *zero-day*. Por outro lado, quando são conhecidas, as vulnerabilidades passam a ser incluídas na base de dados de vulnerabilidades das ferramentas de SCA. Em muitos casos é usada a furtividade no ataque e a recolha de informações torna-se uma poderosa arma a longo prazo para, posteriormente, se fazer um ataque de destruição muito maior, que pode afetar outras cadeias de fornecimento. Comparando as duas mais recentes e maiores vulnerabilidades, (SolarWinds e Log4j), é de notar que o ataque à SolarWinds foi totalmente direcionado à cadeia de fornecimento por um adversário específico e altamente sofisticado, com intenção de comprometer organizações específicas para atingir objetivos políticos ou sociais. Já o Log4j é uma vulnerabilidade extremamente difundida, fácil de explorar e com um potencial prejudicial altíssimo, que pode ser usada para causar danos reais. O Log4j é uma vulnerabilidade descomunal comparada à da SolarWinds, pois não é apenas um pacote de software que as empresas usam, é um código de software que nós, consumidores, usamos regularmente, código aberto ao qual todos têm acesso.

Capítulo 3 Ferramentas de SCA

O objetivo deste capítulo é entender os requisitos básicos das ferramentas de SCA, de modo a compreender quais são as partes fundamentais neste tipo de ferramentas.

De seguida, são abordadas as caraterísticas das ferramentas de SCA, aprofundando o conhecimento sobre as mesmas e tomado em consideração as suas capacidades e restrições.

Por fim, é explanado mais especificamente, o tratamento geral de componentes FOSS (*Free and Open Source Software*) pelas ferramentas de SCA, sendo este tipo de componentes fulcral dada a sua grande utilização na construção de software e o facto de este estar disponível gratuitamente para qualquer indivíduo. Sendo também um alvo fácil na descoberta de vulnerabilidades, podem ser exploradas por cibercriminosos.

3.1 Requisitos básicos das ferramentas de SCA

O foco das organizações ao procurar ferramentas de automatização de SCA é conseguir orientar os seus desenvolvedores a resolver ou dificultar a exploração de vulnerabilidades no seu software, ajudar na gestão de políticas sólidas dentro da organização e, um dos mais importantes do ponto de vista de segurança, facilitar a criação de relatórios de segurança. Estes são importantes para auditorias, testes de penetração e avaliações de segurança [36]. São ilustrados na Figura 9 os requisitos básicos das ferramentas de SCA.

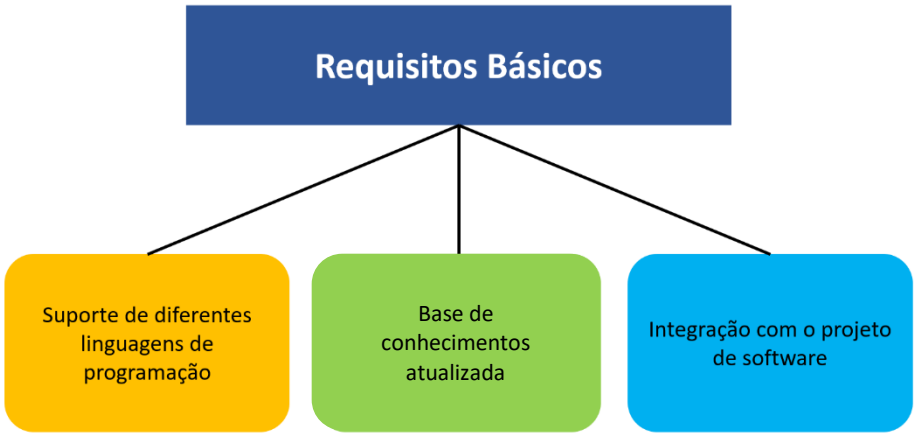

Figura 9 – Esquematização dos requisitos básicos das ferramentas de SCA

3.1.1. Suporte de diferentes linguagens de programação

Uma ferramenta de SCA deve estar preparada para suportar as linguagens de programação identificadas nos ambientes do ciclo de desenvolvimento onde a ferramenta vai ser utilizada. Também pode ser relevante, para o âmbito do software onde será aplicada a ferramenta, suportar linguagens de programação para além das linguagens identificadas antecipadamente. Isto é, ter em conta as linguagens de programação com mais importância no mercado que se possam tornar úteis futuramente em caso de necessidade de mudança.

3.1.2. Bases de conhecimentos atualizadas

As soluções de SCA possuem uma base de dados, onde agregam a informação de variadas fontes. Quanto mais informação estas bases de dados possuírem, melhor será a identificação de problemas. As decisões de ferramentas de SCA baseiam-se maioritariamente na base de dados que cada uma possui, pois é onde se verifica se é a ferramenta mais adequada para determinado tipo de software. Para preservar a segurança do nosso software, dependemos da base de dados da nossa solução de SCA, sendo este um ponto crucial de decisão, visto que não existe uma fonte centralizada de informações sobre atualizações e *patches* [36].

3.1.3. Integração com o projeto de software

Outro fator importante a considerar numa ferramenta de SCA é a integração com repositórios, ferramentas de desenvolvimento de software, e gestores de pacotes de servidores. Ou seja, as melhores soluções de SCA são as que garantem o seu serviço durante todo o ciclo de vida de produção de software [36].

3.2 Características das ferramentas de SCA

Vistos os problemas que as soluções de SCA pretendem resolver irá agora explicar-se as características das mesmas e como vêm resolver os problemas acima referidos.

As ferramentas de SCA fornecem aos desenvolvedores detalhes sobre os elementos que estão a usar para criar o seu software, sobre a conformidade com as licenças e a qualidade dos elementos usados. Estas ferramentas ajudam a monitorizar e a automatizar a descoberta de vulnerabilidades em softwares, código *open source* e bibliotecas que sejam dependentes, como ilustrado na Figura 10 [37].

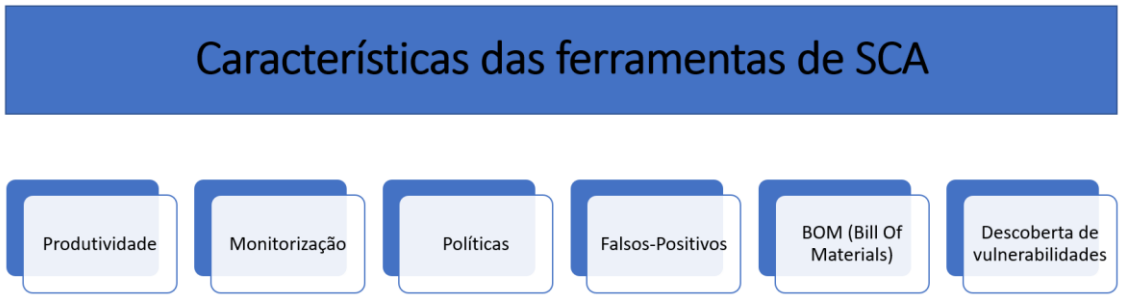

Figura 10 – Características das ferramentas de SCA

3.2.1. Produtividade

As ferramentas de SCA aumentam a produtividade dos desenvolvedores, graças às indicações fornecidas pelas ferramentas dos potenciais problemas e de quão graves estes poderão ser. Consequentemente os desenvolvedores podem tomar medidas adequadas. Esta funcionalidade economiza tempo de trabalho, pois não é preciso ir à procura do problema, sendo que ao se resolver o problema detetado pela ferramenta pode-se voltar imediatamente ao desenvolvimento do código [37].

3.2.2. Monitorização

Para fazer o melhor uso possível das ferramentas de SCA, estas devem ter ativa a monitorização contínua do código. Desta maneira os desenvolvedores podem receber atualizações sobre a segurança do código em todas as fases de desenvolvimento do mesmo. A monitorização também garante confiança de que uma etapa foi bem concluída, e que pode passar para a próxima fase de desenvolvimento.

Esta monitorização deve ser aplicada até em software que não esteja ativo, pois quando se voltar a trabalhar nesse software, os desenvolvedores receberão um relatório das vulnerabilidades atuais que necessitam de ser resolvidas juntamente com um conjunto de sugestões para a resolução das mesmas.

Isto fornece uma visão mais ampla sobre a segurança do código e sobre como os atacantes podem tentar explorar vulnerabilidades [37].

3.2.3. Políticas

As organizações têm a obrigação de que o seu software cumpra o conjunto de políticas definidas. Se os desenvolvedores não estiverem cientes das políticas envolvidas com as dependências que usam, podem possuir graves problemas. As ferramentas de SCA dão informações sobre os padrões de segurança envolvidos com as dependências que existentes.

Outro fator de grande importância é garantir que as medidas de segurança implementadas tenham algum nível de flexibilidade. As ferramentas ajudam a manter as políticas flexíveis juntamente ao manter os componentes seguros. Um desenvolvedor que esteja habituado a trabalhar com SCA consegue criar código mais seguro do princípio ao fim do projeto, pois tem uma maior noção sobre as políticas de segurança em vigor [37].

3.2.1. Falsos-Positivos

Um grande problema das ferramentas de automatização são os falsos positivos que podem ser um problema difícil de lidar. Cerca de 75 a 85% das vulnerabilidades descobertas não são críticas pois as partes do software vulnerável não são necessariamente chamadas pelo software proprietário.

Para evitar os falsos positivos, é necessário configurar a ferramenta de SCA de modo que esta seja usada com precisão. Os falsos positivos podem atrasar o processo, quando esse não é o objetivo das ferramentas de SCA. Como referido, um dos objetivos é economizar o tempo do desenvolvedor nas questões de política e segurança. Para tal, dispor de uma entidade ou serviço que permita fazer a configuração correta destas ferramentas é importante [37].

3.2.2. BOM (*Bill Of Materials* – Lista de Materiais)

As ferramentas de SCA funcionam com uma lista de materiais (Bill Of Materials - BOM). A BOM fornece informações sobre as dependências do nosso software, como licenças e versões. A BOM ajuda as equipas de segurança a obter uma compreensão mais aprofundada de todos os elementos que estão a ser usados pelo software. Assim, pode-se descobrir falhas de licenciamento e segurança das aplicações e corrigi-las antecipadamente [37].

Algumas ferramentas utilizam linguagens de descrição para a BOM, isto é, permitem criar a BOM independentemente da linguagem de programação utilizada. Como exemplos tem-se a linguagem CycloneDX usada na OWASP Dependency Track, SPDX (*Software Package Data Exchange*) e SWID (*Software Identification Tags*) [38].

Para a criação das BOM foram criados individualmente mecanismos específicos de cada linguagem para identificação das dependências e das suas versões. No entanto, não são boas soluções genéricas pois variam em capacidade e não possuem informações, tais como licenças.

A Tabela II contém as linguagens de programação e os respetivos mecanismos para auxiliar a criação das BOM. Adicionalmente, também existem ferramentas de SCA que apenas são compatíveis com determinadas linguagens de programação.

Tabela II – Soluções das linguagens de programação para a identificação de dependências

Linguagem de programação	**Solução**
Java	pom.xml
Javascript	yarn.lock
PHP	composer.lock
Python	requirements.txt
Ruby	Gemfile.lock
Rust	Cargo.lock
.NET	<PackageReference>

3.2.3. Descoberta de Vulnerabilidades

As ferramentas de SCA conseguem ajudar ambas as abordagens de descoberta de riscos: *top-down* e *bottom-up*.

A abordagem de descoberta de riscos do tipo *top-down* garante que as dependências de fontes *open-source* e de terceiros sejam seguras para que haja a garantia de que o software em geral não seja vulnerável. Os desenvolvedores também podem receber informações sobre licenças desatualizadas, realizando assim atualizações e *patches* de acordo com essas informações.

A abordagem do tipo *bottom-up* envolve o uso de SCA para ter controlo sobre os elementos de segurança quando é usado código de terceiros. O uso de software de terceiros torna o desenvolvimento de software bastante mais rápido, pois pode não precisar de ser novamente digitado bastando uma ligeira adaptação ao caso.

Quando são exploradas vulnerabilidades nas dependências usadas, os desenvolvedores têm de estar informados o mais rapidamente possível sobre o que foi afetado para que possam corrigir o sucedido ou mitigar o problema o quanto antes. As ferramentas de SCA permitem identificar de que área do software foi afetada o que facilita a resolução de problemas.

Para além disto, o SCA também pode ser usado para escolher bibliotecas. Estas bibliotecas são sujeitas a uma verificação das suas atualizações para garantir que a atualização mais recente seja usada, o que diminui os riscos de segurança [37].

3.3 Tratamento de Componentes FOSS

Dado que a maior aplicação das ferramentas de SCA é em componentes FOSS, existem abordagens específicas para os mesmos, tendo considerações mais específicas para o tipo de componentes.

Para tal, primeiramente, é necessário identificar os componentes FOSS usados no software, esta é das partes mais demoradas, onde é mais benéfico o uso de ferramentas de automação.

3.3.1. *Scanning* de components FOSS

Scanning é o conjunto de operações que identificam os componentes FOSS utilizados, a sua proveniência e as respetivas licenças. Ou seja, quando se extrai informação diretamente da fonte e de ficheiros binários. Para além do mencionado, o *scanning* não necessita de uma base de dados externa [3].

Através do *scanning* é possível identificar [3]:

- Informação estruturada de manifestos de pacotes e scripts de *built*;
- Avisos explícitos, etiquetas, menções e textos de licenças;
- Pistas de proveniência:
    - Email;
    - URL (*Uniform Resource Locators*);
    - Código específico de construções, como a importação de linguagens de programação (incluindo declarações, *namespaces* e estruturas de árvore de código).

A técnica mais simplista de *scan* é recolher a informação de um componente FOSS que possui a proveniência estruturada e a informação de licença. Para tal é preciso ter em consideração [3]:

- A instalação a partir de um repositório de pacotes, pois o pacote possui um manifesto que contém informação estruturada da proveniência, incluindo referências ao código fonte e repositórios de controlo de versão.
- Ficheiros de código e documentação podem conter outra informação estruturada, como o documento SPDX (*Software Package Data Exchange*) ou um ficheiro de avisos

As ferramentas de gestão de pacotes FOSS e algumas ferramentas externas oferecem formas de recolher dados dos pacotes FOSS. No entanto, fornecer uma visão unificada dos metadados do software é ainda um desafio, pois cada gestor de pacotes possui a sua própria maneira de apresentar os metadados estruturados. Para contextualização deste desafio, existem aproximadamente 800 maneiras diferentes de declarar que um ficheiro foi licenciado sob a GNU *General Public License* (GPL) nas fontes do kernel do Linux. Para além deste fator, alguns gestores de pacotes, no seu manifesto, não possuem informações de licença (por exemplo os gestores para a linguagem de programação Golang [3]).

Devido à incompletude, ambiguidade ou inexistência da declaração de licença no manifesto de um pacote, é necessário detetar e normalizar outras referências de licenças em texto e ficheiros de avisos numa base de código. Existem três abordagens principais para detetar licenças, apresentadas na Tabela III [3].

Tabela III – Abordagens de deteção de licenças

Abordagens	Descrição
1. Correspondência de padrões	São selecionados manualmente pequenos padrões de texto e usados como *proxies* para pesquisa das licenças
2. Correspondência probabilística de texto	É utilizada uma métrica de similaridade (geralmente a *edit distance*, que quantifica o quão diferentes duas *strings* são contando o número de operações necessárias para transformar uma *string* na outra) para encontrar a correspondência textual mais próxima da licença ou do aviso associado à licença.
3. Comparações exaustivas de pares	Utiliza alinhamentos de sequência de texto (conhecido como diff) para encontrar licenças semelhantes.

A maioria das ferramentas existentes utiliza as abordagens 1 e 2. Dentro das ferramentas FOSS mais usadas para a deteção de licenças tem-se [3]:

- Fossology (usa a abordagem 1).
- Github licensee (usa a abordagem 2).
- ScanCode Toolkit (pode usar qualquer uma das três abordagens).

3.3.1. *Matching* de componentes FOSS

Matching é a pesquisa da proveniência de ficheiros baseado na correspondência do conteúdo do ficheiro e nos seus atributos para um índice externo dos componentes FOSS. Ou seja, em contraste com o *scanning*, o objetivo do *matching* é encontrar "aproveitamento" de projetos FOSS com base na deteção das similaridades de código, como duplicados, quase duplicados e clones. Para o *matching* é necessária uma base de dados pré-indexada de componentes FOSS conhecidos, incluindo metadados e código [3].

O *scanning* não é possível no caso de não existir informação da proveniência e das licenças no código que se analisa. Como tal, o caso de uso principal do *matching* é analisar ficheiros de código-base que não têm nenhuma informação clara da proveniência e de licenças. O segundo caso de uso do *matching* é a verificação de que os componentes FOSS identificados no *scanning* correspondem aos ficheiros originais do projeto FOSS correto [3].

A abordagem básica no *matching* consiste em encontrar semelhanças textuais entre o código em análise e outros ficheiros de código fonte e binário. O tamanho destas *queries* pode atingir vários Terabytes, e as pesquisas mais pequenas conseguem estar na ordem de Gigabytes, uma vez que toda a base de códigos está a ser analisada. Para dar uma noção de proporção, as pesquisas na internet, como o "Google Search", encontram-se na ordem de Petabytes [3].

Para se reduzir este problema de escala de pesquisa, a solução é reduzir a dimensão do mesmo. Para tal, são usados ficheiros de tamanho fixo, fragmentos de código de verificação e impressões digitais e esboços "fuzzy" como *proxies*, para pesquisar por similaridades de ficheiros de uma maneira mais eficaz em termos de tempo e custo [3].

O principal desafio do *matching* é, que ao se usar um índice maior, as ferramentas tendem a fornecer muitos falso-positivos que requerem revisão de um especialista. Isto acontece,

porque os componentes FOSS tendem a ser bastante reutilizados por outros projetos FOSS. Isto gera presenças de duplicados ou quase duplicados criando ambiguidade nas correspondências [3].

Como principais ferramentas de *matching* tem-se [3]:

- BlackDuck
- Palamida
- FOSSID

3.3.2. Conclusões gerais sobre *scanning* e *matching*

O *scanning* e o *matching* complementam-se. No entanto, é mais eficiente começar por se realizar o *scanning*, pois é mais rápido e preciso. Independentemente por onde se escolha começar, e quais as ferramentas e técnicas a utilizar, é necessário realizar o planeamento das atividades de SCA de acordo com os componentes FOSS usados. Ou seja, que componentes são distribuídos versus aqueles que apenas se utilizam internamente para desenvolvimento, testes e integrações/entregas contínuas [3].

Capítulo 4 Metodologia Proposta

Neste capítulo são apresentados os critérios de avaliação. Também é apresentada a metodologia AHP que visa resolver dificuldades em conciliar pesos e critérios para este procedimento.

4.1 Critérios de avaliação

Para esta secção foram selecionadas algumas ferramentas com base na sua presença atual no mercado. Para além disso, também há duas ferramentas que são abordadas por pedido da empresa na qual se realiza a dissertação no âmbito do estágio mencionado, sendo estas ferramentas:

- OWASP Dependency Check
- OWASP Dependency Track

A pesquisa de informação sobre as ferramentas, juntamente ao verificar-se aquilo que é possível saber pela documentação e informações de utilização, é o que auxiliou o determinar dos critérios abaixo apresentados para este procedimento.

Como parâmetros de avaliação utilizados para comparar e apresentar as ferramentas de SCA foram baseados no "*An Open Guide To Evaluating Software Composition Analysis Tools*" da "*The Linux Foundation*". Como explicação mais aprofundada do significado de cada um dos critérios a este documento tem-se o Apêndice A com a tabela de informação relevante traduzida do inglês para português do documento referido [6]. Como tal os parâmetros de avaliação definidos para o procedimento deste trabalho são baseados nos seguintes tópicos [5]:

- Base de conhecimentos:
  - Tamanho da base de conhecimentos;
  - Frequência de atualização da base de conhecimentos.
- Capacidades de deteção:
  - Habilidade de deteção de fragmentos de código fonte;
  - Habilidade para auto identificar código fonte (componentes e fragmentos).
- Facilidade de utilização:
  - Intuição;
  - Requer o mínimo de treino possível.
- Capacidades operacionais:
  - Velocidade de scans;
  - Habilidade para usar scans M&A (*Mergers and Acqusitions*) - sem bloqueio de licença em modelos de utilização;
  - Dar suporte a diferentes modelos de auditoria;
  - Agnóstico a linguagens de programação.
- Capacidades de integração:
  - Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*).
- Base de dados de vulnerabilidades de segurança:

    - Tamanho da base de dados;
    - Frequência de atualização;
    - Fontes da informação;
    - Investigação da validação de alertas de vulnerabilidades.
- Métodos de descoberta avançados:
    - Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente).
- Custos associados:
    - Custo de infraestrutura;
    - Custo operacional;
    - Custo de licenciamento;
    - Custo de integração;
    - Custo *lock-in*
    - Custo da personalização de engenharia.
- Modelos de implementação:
    - *On-site* (conhecido também por *On premises*);
    - Nuvem;
    - Híbrido.
- Outros:
    - Instalação modular;
    - Geração de avisos necessários;
    - Capacidades de comunicação.

Estes parâmetros foram modificados de acordo com a informação disponibilizada sobre as ferramentas a analisar. Como não existem métricas padrão quantitativas para as ferramentas de SCA, estas foram definidas ao serem analisadas as ferramentas de acordo com os parâmetros existentes. Na listagem abaixo, os critérios com "Removido:" foram os critérios excluídos e os critérios com "Adicionado:" foram os critérios acrescentados. Os critérios adicionados e removidos apresentam a justificação dos mesmos na submarca de cada um.

Após a recolha de informação feita a lista de parâmetros sofreu as seguintes alterações:

- Base de conhecimentos;
    - Tamanho da base de conhecimentos;
    - Frequência de atualização da base de conhecimentos;
- Capacidades de deteção
    - **Removido:** Habilidade de deteção de fragmentos de código fonte;
        - Justificação: Esta informação não foi encontrada em nenhuma documentação das ferramentas avaliadas;
    - **Removido:** Habilidade para auto identificar código fonte (componentes e fragmentos);
        - Justificação: Esta informação não foi encontrada em nenhuma documentação das ferramentas avaliadas;
    - **Adicionado:** Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado

        - Justificação: Esta capacidade é relevante, pois diminui os falsos positivos gerados pela ferramenta, aumentando assim a sua eficácia.
- Facilidade de utilização
    - Intuição;
    - Requer o mínimo de treino possível;
- Capacidades operacionais;
    - Velocidade de scans;
    - Habilidade para usar scans M&A (*Mergers and Acqusitions*) - sem bloqueio de licença em modelos de utilização;
    - Dar suporte a diferentes modelos de auditoria;
    - Agnóstico a linguagens de programação;
- Capacidades de integração
    - Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*);
- Base de dados de vulnerabilidades de segurança
    - Tamanho da base de dados;
    - Frequência de atualização;
    - Fontes da informação;
    - **Removido:** Investigação da validação de alertas de vulnerabilidades;
        - Justificação: Esta informação não foi encontrada em nenhuma documentação das ferramentas avaliadas;
    - **Adicionado:** Capacidade de priorizar vulnerabilidades
        - Justificação: Esta informação foi destacada em várias ferramentas, sendo algo importante para a organização de um desenvolvedor;
    - **Adicionado:** Remediação automatizada (exemplo, criação pull requests que permite aos programadores atualizar o package recomendado com um único clique)
        - Justificação: Esta capacidade apesar de não ser comum em todas as ferramentas foi aqui adicionada, porque agiliza o trabalho do desenvolvedor;
- Métodos de descoberta avançados;
    - Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente);
- Custos associados;
    - Custo de infraestrutura;
    - Custo operacional;
    - Custo de licenciamento;
    - Custo de integração;
    - Custo *lock-in;*
    - Custo da personalização de engenharia;
- Modelos de implementação;
    - *On-site;*
    - Nuvem;
    - **Removido:** Híbrido;
        - Justificação: Foi removido, pois é a presença do modelo de implementação *on-site* e em nuvem;

- Outros;
  - Instalação modular;
  - Geração de avisos necessários;
  - **Removido:** Capacidades de comunicação.
    - Justificação: Este critério foi retirado, pois era muito ambíguo (a procura de informação era de uma vasta gama) e era necessário algo mais objetivo;
  - **Adicionado:** Capacidade de geração de relatórios
    - Justificação: Critério adicionado para compensar a remoção do anterior dentro do mesmo tópico.

4.1.1. Considerações dos parâmetros

Esta secção explica como foram tomadas as decisões na recolha da informação apresentadas na Tabela IV.

Tabela IV - Considerações dos parâmetros

Base de conhecimentos	
Tamanho	Considera-se o uso de **SBOMs**, a quantidade de **package managers**, ***lockfiles***, ***package ecosystems***, **componentes** ***open-source***, **licenças** ***open-source***, **repositórios e projetos** ***open-source***.
Frequência de atualização	Este parâmetro não se aplica ao uso de SBOMs. Dependendo da informação encontrada consideram-se as seguintes frequências de atualização: • Informação não disponibilizada • De hora em hora • Diária • Semanal • Mensal
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Este parâmetro considera-se **Sim** apenas quando a informação da ferramenta em relação ao tópico é explícita. Caso contrário considera-se **Não**.
Facilidade de utilização	
Intuição	Consideram-se as opiniões de utilizadores e comparações da facilidade de utilização entre ferramentas para ambos os critérios. É utilizada a seguinte escala de avaliação **(1- Muito pouco intuitivo; 5- Muito intuitivo)**
Requer o mínimo de treino possível	
Capacidades operacionais	
Velocidade de scans	SBOMS não realizam scans, pelo que este parâmetro só é válido para a criação de *Bill*

	Of Materials que recorram ao scan de código. Dependendo da informação encontrada consideram-se as seguintes velocidades de scans: • Informação não disponibilizada • Alguns segundos • Alguns minutos
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Considera-se como **Sim** as ferramentas que usarem a licença **Apache License 2.0** e **MIT**. "The Apache 2.0 license is a particular type of *open-source*, permissive software license that ensures that end-users are granted a license to any patent that is covered by the software in question. An Apache 2.0 license ensures the security and availability of safe and powerful *open-source* software." [39] "The MIT license gives users express permission to reuse code for any purpose, sometimes even if code is part of proprietary software. As long as users include the original copy of the MIT license in their distribution, they can make any changes or modifications to the code to suit their own needs." [40]
Dar suporte a diferentes modelos de auditoria	Consideram-se os seguintes modelos de auditoria: • Tradicional • Blind • DIY O modelo de auditoria **Tradicional** é o modelo utilizado normalmente pelas ferramentas. O modelo de auditoria **DIY** é um modelo onde a ferramenta permite o utilizador personalizar a sua auditoria. O modelo de auditoria **Blind** é o modelo onde se realiza auditorias de hashes criptográficos, ou seja, os códigos-fonte nunca são acedidos ou transferidos, totalmente realizado remotamente e é fácil de configurar e administrar e facilita a parte legal envolvida.

Agnóstico a linguagens de programação	Considera-se que algumas ferramentas que usam **SBOM são agnósticas** a linguagens de programação se não houver mais informação sobre a ferramenta neste tópico. Se a ferramenta não for agnóstica a linguagens de programação considera-se a **quantidade de linguagens de programação** que a ferramenta suporta.
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI	Consideram-se os seguintes sistemas de construção: • APIs • CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	Considera-se se a ferramenta permite ter **vulnerabilidades apenas do projeto a ser analisado** ou de um **largo conjunto de projetos**. Considera-se também a **quantidade de componentes vulneráveis**, **vulnerabilidades conhecidas** e **vulnerabilidades únicas** que a base de dados de vulnerabilidades de segurança possui.
Frequência de atualização	Consideram-se as seguintes frequências de atualização: • Informação não disponibilizada • Cada vez que a ferramenta corre (faz verificação dos *data feeds*) • Várias vezes ao dia • Diária • Semanal • Mensal
Fontes de informação	Consideram-se todas as fontes de informação de vulnerabilidades utilizadas como: • NVD • Alertas de Segurança • Centros de investigação de cibersegurança • Bases de dados próprias da ferramenta
Capacidade de priorizar vulnerabilidades	Considera-se como **Sim** ou **Não** a existência desta capacidade nas ferramentas**.**
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores	Capacidade de criar *pull requests* com remediações simples de atualizar as versões

atualizar o package recomendado com um único clique)	de *packages* e vulnerabilidades de remediação fácil. Considera-se como **Sim** ou **Não.**
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	No caso de a informação não estar disponibilizada considera-se como **Não**. Considera-se como **Sim** ou **Não.**
Custos associados	
Infraestrutura	São os custos de manutenção do hardware. Considerar os custos com base nos custos das máquinas e dos planos de pagamento existentes em cada ferramenta. Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** No caso de não possuir qualquer custo considera-se **Sem custos.**
Operacional	Relativo aos falsos positivos. Há mais esforço dedicado das pessoas para interpretar os relatórios da ferramenta, bem como suprimir todos os falsos positivos a cada análise. Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** No caso de não possuir qualquer custo considera-se **Sem custos.**
Licenciamento	São os custos associados á utilização de licenças durante a utilização da ferramenta. Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** No caso de não possuir qualquer custo considera-se **Sem custos.**
Integração	Capacidade de se inserirem em pipelines automáticas e em package managers. Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** No caso de não possuir qualquer custo considera-se **Sem custos.**

Lock-in	Relativo a ferramentas serem *open-source*, que se baseiam nas principais bases de dados de vulnerabilidades (não têm uma base de dados própria) e são integradas agilmente nos *package managers*. **(**Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** No caso de não possuir qualquer custo considera-se **Sem custos.**
Personalização de engenharia	Relativo ao esforço e custo de personalização das ferramentas. Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** No caso de não possuir qualquer custo considera-se **Sem custos.**
Métodos de implementação	
On-site	O modelo *on-site* é necessário instalação na máquina No caso de a ferramenta ser híbrida no modelo de implementação este parâmetro é considerado como **Sim**. Considera-se como **Sim** ou **Não.**
Nuvem	O modelo em nuvem é a maneira que não é necessário descarregar software. A instalação numa *Virtual Machine* não é considerada em nuvem. Ter em consideração capacidade do uso de plugins. No caso de a ferramenta ser híbrida no modelo de implementação este parâmetro é considerado como **Sim**. Considera-se como **Sim** ou **Não.**
Outros	
Instalação modular	Considera-se como **Sim** ou **Não** a capacidade de instalação modular numa ferramenta.
Geração de avisos necessários	Considera-se como **Sim** ou **Não** a capacidade de a ferramenta gerar avisos necessários.
Capacidade de geração de relatórios	Considera-se como **Sim** ou **Não.**

	No caso da resposta ser **Sim**, consideram-se os **tipos de ficheiro** onde o relatório pode ser apresentado

4.2 Proposta AHP

Dada a limitação e dificuldade de definir pesos fixos e conciliar vários critérios para a metodologia que poderia resultar em possuir diferentes pesos consoante o avaliador, com a devida revisão bibliográfica foi achado o *Analytic Hierarchy Process*.

O AHP (Analytic Hierarchy Process) é um processo de tomada de decisões que compara múltiplas alternativas, cada uma com vários critérios, para determinar a melhor opção. O AHP combina matemática e psicologia para comparar várias opções e selecionar a melhor. Faz isto utilizando um conceito chamado *pairwise comparisons*. Em vez de comparar vários critérios de uma só vez, são comparados dois de cada vez. O AHP utiliza álgebra linear para avaliar os resultados de cada *pairwise comparison*. Cada critério recebe o seu próprio peso de importância e quanto maior for o peso, mais importante é o critério para a decisão global [41].

Isto permite que cada avaliador decida os pesos dos critérios de acordo com a sua opinião, enquadrando-se no procedimento desenvolvido no âmbito desta dissertação, de modo a não haver pesos fixos e pré-definidos onde opiniões podem diferir e podendo eventualmente acompanhar desenvolvimento tecnológico onde determinados critérios possam ganhar ou perder importância.

O AHP consiste em quatro etapas [42]:

(1) Identificar a decisão, opções e critérios.
(2) Realizar as comparações em pares.
(3) Calcular o peso da importância de cada critério.
(4) Identificar a melhor opção calculando a utilidade.

4.2.1. Identificar a decisão, opções e critérios

Para decidir os critérios é necessário efetuar-se a seguinte pergunta: "qual é a alternativa certa para mim?". Neste caso concreto queremos decidir qual das ferramentas SCA será a melhor para uma situação específica dentro de uma determinada gama de ferramentas. Uma vez entendida a questão e as opções, decidir-se-á os critérios a considerar. O procedimento criado vem atender esta etapa do AHP, pois os critérios foram deliberadamente escolhidos como é dilucidado neste trabalho. No entanto, existe sempre espaço para acréscimo de critérios caso um avaliador assim o decida. Esta fase torna-se assim flexível com o uso do AHP juntamente com o procedimento criado. Após identificados os critérios, poderá iniciar-se o processo de comparação de pares [42].

4.2.2. Realizar as comparações em pares

Nesta etapa são utilizados os critérios definidos na etapa anterior e serão então organizados em matrizes. Nas matrizes AHP, cada critério tem a sua própria linha e coluna. As matrizes quadradas resultantes permitem a comparação em pares de todas as combinações possíveis de critérios.

Na Tabela V é apresentada a Matriz A, que é uma matriz 3x3 exemplificativa. Mostra como as comparações em pares são organizadas e referenciadas utilizando subscritos: por exemplo, x12 refere-se ao espaço da grelha na primeira linha, segunda coluna.

Na Tabela VI é apresentada a Matriz B, que também é uma matriz 3x3 exemplificativa. Esta matriz exemplifica uma situação com 3 critérios: custo, funcionalidades e utilização. Neste exemplo, cada espaço da grelha contém uma pontuação das comparações em pares. Estas pontuações de amostra mostram que o custo é o fator de decisão mais importante, seguido pelas funcionalidades e, por último, a utilização.

Tabela V - Matriz A

A	Crit. 1	Crit. 2	Crit. 3
Crit. 1	x_{11}	x_{12}	x_{13}
Crit. 2	x_{21}	x_{22}	x_{23}
Crit. 3	x_{31}	x_{32}	x_{33}

Tabela VI - Matriz B

B	Custo	Utilização	Funcionalidades
Custo	1	7	5
Utilização	1/7	1	1/3
Funcionalidades	1/5	3	1

Para se fazer as comparações em pares, primeiramente, é necessário atribuir um número a cada espaço da grelha. Este número é a importância relativa dos dois critérios. Por exemplo, uma pontuação de 1 significa que ambos os critérios são igualmente importantes. Quando um critério é comparado consigo mesmo, a sua importância relativa é 1, pois os critérios que estão a ser comparados são os mesmos. Números maiores mostram que um critério é cada vez mais importante, sendo 9 a pontuação mais alta. Na linha 1 da Tabela VI, o custo é sete vezes mais importante ($x_{12} = 7$) para a decisão final do que a utilização e cinco vezes mais importante ($x_{13} = 5$) do que as funcionalidades. Este processo de pontuação é repetido para todos os espaços da grelha à direita da diagonal (a metade superior direita da matriz). Como o custo é sete vezes mais importante do que a utilização, podemos também dizer que a localização é 1/7 tão importante como o custo (x_{21}). Da mesma forma, uma vez que o custo é cinco vezes mais importante do que as funcionalidades, as funcionalidades devem ser 1/5 tão importante como o custo (x_{31}). Este processo de utilização de inversos (1/x), para descrever a relação oposta entre dois critérios repete-se para todos os espaços da grelha à esquerda da diagonal (a metade inferior esquerda da matriz). Se o critério 1 for x vezes tão importante como o critério 2, então o critério 2 deve ser 1/x tão importante como o critério 1 [42].

4.2.3. Calcular o peso da importância de cada critério

Após a complementação da matriz a mesma é utilizada para calcular os pesos de importância, que dizem o quanto cada critério terá em conta na decisão final. Quanto maior for o peso de importância de um determinado critério, mais influência terá na sua decisão final. Esta parte do AHP depende de álgebra linear. O primeiro passo para determinar o peso

de um critério é encontrar a média geométrica. O valor médio de um conjunto de números encontrados utilizando o produto dos seus valores, em vez da soma (V) da linha. Pode-se encontrar a média geométrica multiplicando todas as pontuações de importância relativa da linha (x) e tomando a enésima raiz deste produto (onde n = número total de critérios). Esta equação de amostra mostra como calcular a média geométrica do Critério 1 da matriz genérica 3 × 3 (Tabela V):

$$V_1 = \sqrt[3]{x_{11} * x_{12} * x_{13}}$$

Em seguida, divide-se a média geométrica do critério pela soma das médias geométricas de todos os critérios. O decimal resultante é o peso (W) desse critério. Este método chama-se normalização, porque assegura a soma de todos os pesos igual a 1, ou 100%. A soma dos pesos é igual a 1, porque cada critério representa uma parte de toda a decisão. Esta equação de amostra mostra como calcular o peso do critério 1 da matriz genérica 3 × 3 (Tabela V):

$$W_1 = \frac{V_1}{V_1 + V_2 + V_3}$$

Se se fizerem os mesmos cálculos para o critério do custo, utilizando as pontuações de importância relativa da Tabela VI:

$$V_{custo} = \sqrt[3]{1 * 7 * 5} = 3.27$$

$$W_{custo} = \frac{3.27}{3.27 + 0.36 + 0.84} = 0.73$$

Isto diz-nos que o custo representa 73% da decisão global da ferramenta a utilizar. Utilizando este mesmo método, verificamos que o peso calculado para a utilização é de 8% e as funcionalidades é de 19% [42].

4.2.4. Identificar a melhor opção calculando a utilidade

A etapa final da AHP é determinar a utilidade. A utilidade é um valor numérico que fornece informação sobre a utilidade de algo, e que o ajudará a selecionar a melhor opção. Quanto mais benéfico ou útil for um critério, maior será a sua utilidade [42]. A utilidade pode ser medida de forma diferente para cada critério. Como o procedimento realizado neste trabalho apresenta as possíveis respostas para cada um dos critérios a sua utilidade é definida por cada resposta considerada. A pontuação atribuída a cada resposta será definida pelo avaliador da maneira que melhor entender.

No Apêndice C encontra-se a tabela de apoio à realização do AHP de acordo com os tópicos/critérios definidos para o procedimento desta dissertação.

Capítulo 5 Aplicação do procedimento

Na primeira Secção são atribuídos os pesos aos tópicos, de seguida, aos critérios de cada tópico, e, por fim, às respostas dentro de cada critério.

Após as atribuições dos pesos, as restantes secções apresentam a avaliação individual de uma ferramenta de SCA utilizando o procedimento criado neste trabalho.

5.1 Atribuições de pesos

Nesta secção podemos entender como foram atribuídos os pesos dos parâmetros para a posterior avaliação das ferramentas. As atribuições foram decididas pela equipa onde me encontro inserida no âmbito deste estágio.

Apesar da proposta AHP, as atribuições já tinham sido deliberadas anteriormente à proposta, sendo que se manteve o trabalho feito. No entanto, é de notar que as atribuições feitas são exemplificativas (demonstram um determinado cenário) para o uso do procedimento, sendo importante referir que os critérios é que são o foco do mesmo.

A Tabela VII apresenta as atribuições de pesos por tópicos, explanando o motivo da cotação atribuída.

Tabela VII - Atribuições de pesos por tópicos

Tópicos	**% Atribuída**	**Motivo**
Base de conhecimentos	10%	Não é de tanta importância quanto a base de vulnerabilidades, pois existem maneiras simples de facilitar a criação da base de conhecimentos (*lockfiles*, *package managers* e SBOM).
Capacidades de deteção	10%	Este tópico apenas possui um parâmetro de relevância média relativo a falsos-positivos e a capacidade de a ferramenta detetá-lo, evitando assim gasto de tempo do desenvolvedor.
Facilidade de utilização	5%	A facilidade de utilização é um dos tópicos com menos peso na avaliação, pois é feita com base em opiniões e as ferramentas possuem suportes e guias de utilização para apoiar o utilizador.
Capacidades operacionais	10%	Tópico com parâmetros de relevância média relativo a scans, auditoria e agnóstico a linguagens de programação.

Capacidades de integração	10%	Tópico de relevância média devido a uso de APIs e possuir CLI para integração, agilização e facilitação do trabalho.
Base de dados de vulnerabilidades de segurança	20%	Considerei o tópico mais importante da ferramenta, pois é o que permite detetar as vulnerabilidades e remediá-las, sendo o ponto crucial de uma boa ferramenta de SCA.
Métodos de descoberta avançados	5%	Este tópico apenas inclui apenas um parâmetro e o mesmo não é de baixa relevância, pois pode ser contornado por outros métodos.
Custos associados	15%	Os custos associados é o segundo tópico com maior peso na avaliação pois, tem de possuir uma importância alta a ser considerada em relação aos custos de uso da ferramenta comparativamente a tópicos de funcionalidades.
Métodos de implementação	10%	O método de implementação torna-se um fator de relevância média da ferramenta em termos de agilidade ou em aplicação a diferentes projetos dentro de uma empresa.
Outros	5%	Os parâmetros a serem considerados neste tópico são de menor relevância comparativamente aos restantes.
Total	100%	

A Tabela VIII apresenta a atribuição de pesos por parâmetros, sendo que o peso atribuído a cada critério é cotado de 0 a 100% dentro do tópico pertencente. Ou seja, cada tópico vale 100% de modo que seja possível a um diferente avaliador ajustar o peso dos critérios a diferentes pesos dos tópicos.

Tabela VIII - Atribuições de pesos por parâmetros

Parâmetros	**% Atribuída**	**Motivo**
Base de conhecimentos		
Tamanho	60%	O tamanho da base de conhecimentos é o parâmetro mais importante deste tópico, pois é onde a informação do código é reconhecida.
Frequência de atualização	40%	Este tópico é de relevância mais baixa, pois, dependendo da base de conhecimentos pode nem haver atualização deste parâmetro.
Capacidades de deteção		

Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	100%	A percentagem atribuída ao tópico aplica-se apenas a um parâmetro, logo tem a cotação total.
Facilidade de utilização		
Intuição	25%	Parâmetros de igual relevância, pois são atribuídos com base em opiniões e são parâmetros que se encontram interligados.
Requer o mínimo de treino possível	25%	
Capacidades operacionais		
Velocidade de scans	10%	Atribuída a menor percentagem, pois este parâmetro é de pouca relevância devido à velocidade de scans ser geralmente baixa e porque esta informação nem sempre se encontra disponível na documentação das ferramentas.
Habilidade para usar scans M&A	40%	Parâmetro de maior relevância para não existir bloqueio de licenças em modelos de utilização.
Dar suporte a diferentes modelos de auditoria	20%	Parâmetro de importância menor à velocidade de scans, mas maior do que ser agnóstico a linguagens de programação, pois praticamente todas as ferramentas de SCA dão suporte a modelos de auditoria tradicional, apenas algumas diferindo no usa dos modelos Blind e DIY.
Agnóstico a linguagens de programação	30%	Parâmetro de 2ª maior importância deste tópico, pois é útil para analisar múltiplos projetos a usar diferentes linguagens de programação ou quando há necessidade de se alterar a linguagem de programação de um projeto.
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI	100%	A percentagem atribuída ao tópico aplica-se apenas a um parâmetro, logo tem a cotação total.
Base de dados de vulnerabilidades de segurança		
Tamanho	40%	Parâmetro de relevância média, pois considera-se apenas conseguir avaliar um projeto, ou múltiplos e considera-se também a quantidade vulnerabilidades relevantes indicadas pelas ferramentas.

Frequência de atualização	30%	Parâmetro de relevância média baixa, pois é importante atualizar a informação que se vai buscar às fontes de informação, no entanto, por norma, a frequência de atualização é relativamente baixa.
Fontes de informação	80%	Parâmetro de relevância mais alta neste tópico, pois é são as fontes onde se vão buscar as vulnerabilidades a serem detetadas no código.
Capacidade de priorizar vulnerabilidades	30%	Parâmetro de relevância média baixa, pois é de alguma importância no suporte ao desenvolvedor em avisar o que é prioritário resolver.
Remediação automatizada	20%	Parâmetro de relevância mais baixa, pois não é uma função necessária a uma ferramenta de SCA, no entanto, facilita o trabalho do desenvolvedor e é uma funcionalidade útil para poupar tempo.
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	50%	A percentagem atribuída ao tópico aplica-se apenas a um parâmetro, logo tem a cotação total.
Custos associados		
Infraestrutura	40%	Custo de maior importância, associado ao custo da ferramenta
Operacional	30%	2º Custo de maior importância pois, é relativo aos falsos positivos das ferramentas e ao esforço dedicado a analisar e suprimir os mesmo
Licenciamento	15%	Custo de menor relevância, pois poucas ferramentas necessitam de custos de licenciamento adicionais.
Integração	30%	Custo de relevância média, pois refere-se à capacidade de se inserirem em pipelines automáticas e em *package managers*, no entanto praticamente todas as ferramentas possuem algum mecanismo de integração.
Lock-in	20%	Custo de relevância média baixa, pois é relativo a ferramentas serem *open-*

		source, que se baseiam nas principais bases de dados de vulnerabilidades (ou seja, não têm uma base de dados própria). No entanto, praticamente todas as ferramentas possuem uma base de dados de vulnerabilidades *open-source* (como a NVD), até mesmo as que possuem uma base de dados própria.
Personalização de engenharia	15%	Custo de menor relevância, pois poucas empresas cobram os custos de personalização das ferramentas (muitas ferramentas já incluem opções de personalização sem custos).
Métodos de implementação		
On-site	50%	Nestes parâmetros os 10% dividem-se pelos 2, pois o melhor resultado é o resultado híbrido (*on-site* e nuvem).
Nuvem	50%	
Outros		
Instalação modular	15%	A instalação modular é permitir aplicar a ferramenta em diferentes módulos de um projeto, pelo que a sua relevância é ligeiramente menor do que a geração de avisos, mas considerada igual em relação à geração de relatórios.
Geração de avisos necessários	20%	Dentro deste tópico este é o parâmetro de maior relevância, visto que é a geração de avisos de vulnerabilidades.
Capacidade de geração de relatórios	15%	A geração de relatórios é um fator de igual relevância à instalação modular, pois é um fator de alguma importância de organização e de suporte.

A Tabela IX apresenta a atribuição de pesos consoante as possíveis respostas, sendo que o peso atribuído a cada critério também é cotado de 0 a 100% dentro do tópico pertencente. Ou seja, cada tópico vale 100% de modo que seja possível a um diferente avaliador ajustar o peso dos critérios a diferentes pesos dos tópicos.

Tabela IX - Atribuição de pesos por respostas

Base de conhecimentos	**Respostas**	**Motivo**
Tamanho **(60%)**	A avaliação deste parâmetro é deixada ao critério do avaliador	Depende do que o avaliador procura numa ferramenta de SCA. Neste caso pode haver

	consoante as suas preferências **(60%)**.	preferência em SBOM ou em determinados *packages*.
Frequência de atualização **(40%)**	Este parâmetro não se aplica ao uso de SBOMs pelo que a percentagem atribuída é de **40%**. Dependendo da informação encontrada consideram-se as seguintes frequências de atualização: • Informação não disponibilizada **(0%)** • De hora em hora **(40%)** • Várias vezes ao dia **(37.5%)** • Diária **(35%)** • Semanal **(30%)** • Mensal **(20%)**	Relativo ao uso de SBOMS não se aplica este parâmetro, logo possui a cotação máxima. Em relação a informação não disponibilizada, como a ferramenta não apresenta a informação apenas se pode considerar 0%. Hora em hora possui a pontuação máxima também, pois é a melhor frequência de atualização que se pode ter. As restantes opções decaem apenas em 5% dando espaço para acrescentar maiores frequências de atualização, visto que a mensal apesar de não ser uma boa opção também não é a pior comparativamente a nem se saber esse tipo de informação.
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado **(100%)**	Este parâmetro considera-se **Sim** **(100%)** apenas quando a informação da ferramenta em relação ao tópico é explícita. Caso contrário considera-se **Não** **(0%)**.	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Facilidade de utilização		
Intuição **(25%)** Requer o mínimo de treino possível **(25%)**	É utilizada a seguinte escala de avaliação **(1- Muito pouco intuitivo; 5- Muito intuitivo)** **1 - 5%** **2 - 10%** **3 – 15%** **4 – 20%**	A escala de avaliação considerada adiciona 5% (1/5 de 25%) à medida que a intuição aumenta.

	5 – 25%	
Capacidades operacionais		
Velocidade de scans **(10%)**	SBOMS não realizam scans, logo será atribuída classificação máxima **10%**. Dependendo da informação encontrada consideram-se as seguintes velocidades de scans: • Informação não disponibilizada **0%** • Alguns segundos **10%** • Alguns minutos **5%**	SBOMs possuem a pontuação máxima por não possuírem *scans*.
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização **(40%)**	Considera-se como **Sim (40%)** ou **Não (0%)**.	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Dar suporte a diferentes modelos de auditoria **(20%)**	Consideram-se os seguintes modelos de auditoria: • Tradicional **6.5%** • Blind **6.8%** • DIY **6.7%**	A atribuição das percentagens foi feita de modo a permitir que a ferramenta suporte os 3 modelos de auditoria simultaneamente. Como praticamente todas as ferramentas possuem o modelo tradicional este ficou com a menor percentagem. Como o modelo de auditoria Blind é o mais seguro foi atribuída a maior percentagem.
Agnóstico a linguagens de programação **(30%)**	Considera-se que algumas ferramentas que usam **SBOM são agnósticas (30%)** a linguagens de programação se não houver mais informação sobre a ferramenta neste tópico. Se a ferramenta não for agnóstica a linguagens de programação considera-se a **quantidade de linguagens de programação** que a	Quando as ferramentas utilizam SBOM são consideradas agnósticas, se não houver mais informação sobre a ferramenta. Logo é atribuída a cotação máxima. No caso da quantidade e quais as linguagens de programação fica a critério do avaliador, consoante as linguagens de programação que

	ferramenta suporta. Deixando a avaliação ao critério do avaliador de acordo com as suas preferências **(30%)**.	pretende ter no seu escopo.
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI **(100%)**	Consideram-se os seguintes sistemas de construção: • APIs **(50%)** • CLI **(50%)**	Neste parâmetro dividiu-se a pontuação para caso a ferramenta possua os dois sistemas de construção.
Base de dados de vulnerabilidades de segurança		
Tamanho **(40%)**	Considera-se se a ferramenta permite ter **vulnerabilidades apenas do projeto a ser analisado (10%)** ou de um **largo conjunto de projetos (30%)**. Considera-se também a **quantidade de componentes vulneráveis**, **vulnerabilidades conhecidas** e **vulnerabilidades únicas** que a base de dados de vulnerabilidades de segurança possui (Avaliador atribui valor até **10%**).	Considera-se 10% para a ferramenta permitir apenas analisar um projeto, pois consegue analisar um projeto inteiro. Um pior cenário seria analisar ficheiro a ficheiro. Assim deixa espaço para outras opções que o avaliador possa encontrar (ou seja, pode atribuir abaixo de 10% para esses casos). Considera-se 30% para a análise de um largo conjunto de projetos e deixa-se espaço para o restante 10% para a informação sobre vulnerabilidades que é deixada a atribuição ao critério do avaliador sobre a relevância dessa informação em relação àquilo que pretende ter.
Frequência de atualização **(30%)**	Consideram-se as seguintes frequências de atualização: • Informação não disponibilizada **(0%)** • Cada vez que a ferramenta corre (**30%** - faz verificação dos *data feeds*)	A informação não disponibilizada é novamente avaliada a 0%. A cada vez que a ferramenta corre também é atribuída a percentagem máxima, pois a informação encontra-se sempre

	• Várias vezes ao dia **(30%)** • Diária **(20%)** • Semanal **(10%)** • Mensal **(5%)**	atualizada quando se usa a ferramenta. A percentagem máxima também é atribuída para "várias vezes ao dia" e decai 10% nas restantes respostas, exceto a mensal, que fica com 5% para deixar espaço para outro cenário que o avaliador possa ter de lidar.
Fontes de informação **(80%)**	Ao critério das preferências do avaliador **(0 a 80%)**	Fica ao critério do avaliador, consoante as fontes de informação que achar de maior relevância.
Capacidade de priorizar vulnerabilidades **(30%)**	Considera-se como **Sim (30%)** ou **Não (0%).**	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Remediação automatizada **(20%)**	Considera-se como **Sim (20%)** ou **Não (0%).**	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente) **(50%)**	No caso de a informação não estar disponibilizada considera-se como **Não (0%)**. Considera-se como **Sim (50%)** ou **Não (0%).**	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Custos associados		
Infraestrutura **(40%)**	Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** **1 – 35%** **2 – 26.25%** **3 – 17.5%** **4 – 8.75%** **5 – 0%**	Sem custos obtém a pontuação máxima. Quando passa para a escala de avaliação decai 5%, pois é para garantir a importância de não possuir custos já que a base de preços de

	No caso de não possuir qualquer custo considera-se **Sem custos (40%).**	ferramentas de SCA é um preço ligeiramente elevado, principalmente para empresas e organização. As restantes percentagens são divididas uniformemente, sendo que a escala em 5 fica a 0%.
Operacional **(30%)**	Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** **1 – 25%** **2 – 18.75%** **3 – 12.5%** **4 – 6.25%** **5 – 0%** No caso de não possuir qualquer custo considera-se **Sem custos (30%).**	
Licenciamento **(15%)**	Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** **1 – 10%** **2 – 7.5%** **3 – 5%** **4 – 2.5%** **5 – 0%** No caso de não possuir qualquer custo considera-se **Sem custos (15%).**	
Integração **(30%)**	Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** **1 – 25%** **2 – 18.75%** **3 – 12.5%** **4 – 6.25%** **5 – 0%** No caso de não possuir qualquer custo considera-se **Sem custos (30%).**	
Lock-in **(20%)**	Relativo a ferramentas serem *open-source*, que se baseiam nas principais bases de dados de vulnerabilidades (não têm uma base de dados própria) e são integradas agilmente nos package managers.	

	(Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** **1 – 15%** **2 – 11.25%** **3 – 7.5%** **4 – 3.75%** **5 – 0%** No caso de não possuir qualquer custo considera-se **Sem custos (20%).**	
Personalização de engenharia **(15%)**	Utiliza-se a seguinte escala de avaliação: **(1 = Mais barato; 5 = Mais caro)** **1 – 10%** **2 – 7.5%** **3 – 5%** **4 – 2.5%** **5 – 0%** No caso de não possuir qualquer custo considera-se **Sem custos (15%).**	
Métodos de implementação		
On-site **(50%)**	Considera-se como **Sim (50%)** ou **Não (0%).**	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Nuvem **(50%)**	Considera-se como **Sim (50%)** ou **Não (0%).**	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Outros		
Instalação modular **(15%)**	Considera-se como **Sim (15%)** ou **Não (0%).**	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Geração de avisos necessários **(20%)**	Considera-se como **Sim (20%)** ou **Não (0%).**	Como este parâmetro é apenas de resposta Sim ou Não a pontuação máxima vai para a resposta de Sim.
Capacidade de geração de relatórios **(15%)**	Considera-se como **Sim (10%)** ou **Não (0%).**	Para a resposta Sim é atribuído 10% e para Não 0%. Deixando

	No caso da resposta ser **Sim**, consideram-se os **tipos de ficheiro** onde o relatório pode ser apresentado. Deixando os **5%** restantes a critério do avaliados com base nas suas preferências.	espaço para 5% para os tipos de ficheiro, sendo que serão atribuídos a critério das preferências do avaliador quanto aos tipos de ficheiro utilizados.

Neste capítulo são feitas as avaliações de cada ferramenta, para tal, são feitas as seguintes considerações gerais:

- Em relação às **fontes de informação das bases de vulnerabilidades** é dada maior pontuação às ferramentas que usem mais fontes *open-source*.
- Na avaliação da **capacidade de geração de relatórios** é atribuída a pontuação máxima apenas com o Sim, pois em relação à percentagem atribuída aos tipos de ficheiro depende do avaliador, que no caso deste trabalho é irrelevante.

5.2 OWASP Dependency-Check

Na Tabela X é efetuada a avaliação da ferramenta OWASP Dependency-Check de acordo com o procedimento descrito.

Tabela X - Avaliação da ferramenta OWASP Dependency-Check

Base de conhecimentos	Recolha de informação	%
Tamanho	Analisadores de tipos de ficheiros: -Archive "Zip archive format (*.zip, *.ear, *.war, *.jar, *.sar, *.apk, *.nupkg); Tape Archive Format (*.tar); Gzip format (*.gz, *.tgz); Bzip2 format (*.bz2, *.tbz2); RPM format (*.rpm)" -Assembly "NET Assemblies (*.exe, *.dll)" -Jar "Java archive files (*.jar); Web application archive (*.war)" -RetireJS "JavaScript files" -Node.js "NPM package specification files (package.json)" -Node Audit	4%

	"Uses the npm audit APIs to report on known vulnerable node.js libraries. This analyzer requires an Internet connection." -Nugetconf "Nuget packages.config file" -Nuspec "Nuget package specification file (*.nuspec)" -OpenSSL "OpenSSL Version Source Header File (opensslv.h)" -OSS Index "Uses the OSS Index APIs to report on vulnerabilities not found in the NVD. This analyzer requires an Internet connection." - Ruby bundler-audit "Ruby Gemfile.lock files"	
Frequência de atualização	Mensal	2%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Não	0%
Facilidade de utilização		
Intuição	4/5 (1-Muito pouco intuitivo; 5-Muito bom)	2%
Requer o mínimo de treino possível	4/5 (1-Muito treino; 5-Pouco treino)	2%
Capacidades operacionais		
Velocidade de scans	Informação não disponibilizada	0%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Tradicional	0.65%
Agnóstico a linguagens de programação	Não. Apenas suporta Java, Javascript e .NET (Total de 3 linguagens de programação)	0.5%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	CLI	5%

Base de dados de vulnerabilidades de segurança		
Tamanho	Só permite ter vulnerabilidades do projeto a ser analisado	1%
Frequência de atualização	Sempre que a ferramenta corre	3%
Fontes de informação	NVD, NPM Audit API, OSS Index, RetireJS e Bundler Audit (Total 5)	7%
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Não	0%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Não	0%
Custos associados		
Infraestrutura	Sem Custos	4%
Operacional	4/5 (1 = Mais barato; 5 = Mais caro)	0.625%
Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	1/5 (1 = Mais barato; 5 = Mais caro)	1.5%
Personalização de engenharia	1/5 (1 = Mais barato; 5 = Mais caro)	1%
Métodos de implementação		
On-site	Sim	5%
Nuvem	Não	0%
Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Não	0%
Capacidade de geração de relatórios	Sim. HTML, XML, JSON e CSV	1.5%
Total		**53.275%**

5.3 OWASP Dependency-Track

Na Tabela XI é efetuada a avaliação da ferramenta OWASP Dependency-Track de acordo com o procedimento descrito.

Tabela XI - Avaliação da ferramenta OWASP Dependency-Track

Base de conhecimentos	Recolha de informação	%
Tamanho	SBOM (CycloneDX)	6%
Frequência de atualização	Não aplicável	4%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se	Não	0%

o componente vulnerável é utilizado		
Facilidade de utilização		
Intuição	4/5 (1-Muito pouco intuitivo; 5-Muito bom)	2%
Requer o mínimo de treino possível	4/5 (1-Muito treino; 5-Pouco treino)	2%
Capacidades operacionais		
Velocidade de scans	Não aplicável (não faz scans ao código)	1%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Tradicional	0.65%
Agnóstico a linguagens de programação	Sim, usa SBOM	3%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	API e CLI	10%
Base de dados de vulnerabilidades de segurança		
Tamanho	Permite ter vulnerabilidades de um largo conjunto de projetos e permite ter um repositório interno com vulnerabilidades dentro da organização.	3.5%
Frequência de atualização	Sempre que a ferramenta corre (verifica os *data feeds* da NVD)	3%
Fontes de informação	NVD, NPM Audit API, the OSS Index, RetireJS e Bundler Audit (Total 5)	7%
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Não	0%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Não	0%
Custos associados		
Infraestrutura	2/5 (1 = Mais barato; 5 = Mais caro)	2.625%
Operacional	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	1/5 (1 = Mais barato; 5 = Mais caro)	1.5%
Personalização de engenharia	1/5 (1 = Mais barato; 5 = Mais caro)	1%

Métodos de implementação		
On-site	Sim	5%
Nuvem	Não	0%
Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	O dependency-track não gera relatórios, as métricas estão embebidas na plataforma. Contudo, como tem uma API simples e intuitiva, é fácil criar um script que nos gere relatórios com o formato que quisermos.	1% (0.5% + 0.5%)
Total		**70.275%**

Considerações:

- Na Capacidade de geração de relatórios é apenas atribuído 0.5% à geração de relatórios, pois a geração não é direta, no entanto a ferramenta permite ser capaz de gerar relatórios através de configurações.

5.4 Gitlab

Na Tabela XII é efetuada a avaliação da ferramenta Gitlab de acordo com o procedimento descrito.

Tabela XII - Avaliação da ferramenta do Gitlab

Base de conhecimentos	Recolha de informação	%
Tamanho	Os analisadores do GitLab obtêm informações de dependência utilizando um dos seguintes métodos: 1. Analisar diretamente os ficheiros de localização. 2. Executar um gestor de pacotes ou ferramenta de construção para gerar um ficheiro de informação de dependência que é depois analisado. Os seguintes gestores de pacotes utilizam ficheiros de localização que os analisadores GitLab são capazes de analisar diretamente: • Bundler • Composer • Conan • Go • NuGet • npm • yarn • Poetry	4.5%

	Para obter informações de dependência executa-se um gerenciador de pacotes para gerar um arquivo analisável. Para oferecer suporte aos seguintes gerenciadores de pacotes, os analisadores do GitLab realizam duas etapas: 1. Executar o gerenciador de pacotes ou uma tarefa específica, para exportar as informações de dependência. 2. Analisar as informações de dependência exportadas. Gerenciadores de Pacotes: • sbt • Maven • Gradle • setuptools • pip • Pipenv	
Frequência de atualização	Diária	3.5%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Não	0%
Facilidade de utilização		
Intuição	5/5 (1-Muito pouco intuitivo; 5-Muito bom)	2.5%
Requer o mínimo de treino possível	5/5 (1-Muito treino; 5-Pouco treino)	2.5%
Capacidades operacionais		
Velocidade de scans	Informação não disponibilizada	0%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Tradicional e DIY	1.32
Agnóstico a linguagens de programação	Não. 11 linguagens de programação - Ruby, PHP, C, C++, Go, Java, JavaScript, .NET, C#, Python, Scala)	1%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	APIs e CLI	10%
Base de dados de vulnerabilidades de segurança		

Tamanho	Permite ter vulnerabilidades de um largo conjunto de projetos	3%
Frequência de atualização	Diariamente	2%
Fontes de informação	NVD, ruby-advisory-db, GitHub Advisory Database	5.5%
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Sim	2%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim	5%
Custos associados		
Infraestrutura	3/5 (1 = Mais barato; 5 = Mais caro)	1.25%
Operacional	3/5 (1 = Mais barato; 5 = Mais caro)	1.25%
Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	2/5 (1 = Mais barato; 5 = Mais caro)	1.125%
Personalização de engenharia	1/5 (1 = Mais barato; 5 = Mais caro)	1%
Métodos de implementação		
On-site	Sim	5%
Nuvem	Não	5%
Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	JSON e uma CycloneDX SBOM para cada *lock* ou *build file* que detetar.	1.5%
Total		73.445%

5.5 Github

Na Tabela XIII é efetuada a avaliação da ferramenta Github de acordo com o procedimento descrito.

Esta ferramenta só permite ter vulnerabilidade do projeto a ser analisado, mas pode ser facilmente configurada para analisar múltiplos projetos. Logo, foi cotado 2% para o tamanho da base de dados de vulnerabilidades, devido a ser o valor intermédio das respostas definidas.

Tabela XIII - Avaliação da ferramenta do Github

Base de conhecimentos	Recolha de informação	%
Tamanho	Ecossistemas de pacotes suportados (segurança de cadeias de fornecimento):	4.5%

	• Composer • NuGet • Go modules • Maven • npm • pip • RubyGems • Yarn O GitHub habilita automaticamente as atualizações de segurança do Dependabot para cada repositório que atende esses pré-requisitos. 1. Pré-requisito de habilitação automática; 2. O repositório não é um *fork*; 3. O repositório não está arquivado; 4. O repositório é público ou privado e ativou-se a análise somente de leitura pelo GitHub, gráfico de dependência e alertas de vulnerabilidade nas configurações do repositório; 5. O repositório contém o arquivo de manifesto de dependência de um ecossistema de pacotes que o GitHub suporta; 6. As atualizações de segurança do Dependabot não estão desabilitadas para o repositório	
Frequência de atualização	Diária. O utilizador pode personalizar a frequência de atualização para diária, semanal ou mensal.	3.5%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Não	0%
Facilidade de utilização		
Intuição	4/5 (1-Muito pouco intuitivo; 5-Muito bom)	2%
Requer o mínimo de treino possível	4/5 (1-Muito treino; 5-Pouco treino)	2%
Capacidades operacionais		
Velocidade de scans	Alguns minutos	0.5%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%

Dar suporte a diferentes modelos de auditoria	Tradicional	0.65%
Agnóstico a linguagens de programação	Não. Linguagens de programação suportadas: C#, Go, Java, JavaScript, PHP, Python, Ruby, Scala, TypeScript	0.8%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	CLI	5%
Base de dados de vulnerabilidades de segurança		
Tamanho	Só permite ter vulnerabilidade do projeto a ser analisado, mas pode ser facilmente configurado para analisar múltiplos projetos.	2%
Frequência de atualização	Diária. O utilizador pode personalizar a frequência de atualização para diária, semanal ou mensal.	2%
Fontes de informação	• NVD; • Uma combinação de *machine learning* e análise humana para detetar vulnerabilidades em *commits* públicos no GitHub • Alertas de Segurança reportados no GitHub (*GitHub Advisory Database*) • A base de dados de Alertas de Segurança da npm	7%
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Sim	2%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim	5%
Custos associados		
Infraestrutura	3/5 (1 = Mais barato; 5 = Mais caro)	1.75%
Operacional	5/5 (1 = Mais barato; 5 = Mais caro)	0%
Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	2/5 (1 = Mais barato; 5 = Mais caro)	1.125%
Personalização de engenharia	3/5 (1 = Mais barato; 5 = Mais caro)	0.5%
Métodos de implementação		
On-site	Não	0%
Nuvem	Sim	5%

Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	Não	0%
Total		59.825%

5.6 WhiteSource Software (Mend)

Na Tabela XIV é efetuada a avaliação da ferramenta WhiteSource Software (Mend) de acordo com o procedimento descrito.

Durante a realização do trabalho esta empresa alterou o nome de "WhiteSource" para "Mend". No seguimento disto, poderá haver informação recolhida com ambos os nomes.

Tabela XIV - Avaliação da ferramenta Mend

Base de conhecimentos	Recolha de informação	%
Tamanho	• 3 Milhões de componentes *open-source*; • Mais de 200 licenças diferentes *open-source*. Possui uma SBOM produzida pela Mend SCA (Identifica todas as bibliotecas *open-source*) Possui cobertura completa sobre a utilização de código aberto com a maior base de dados de vulnerabilidades da indústria. Com mais de 270 milhões de componentes de código *open-source* e 13 mil milhões de ficheiros.	6%
Frequência de atualização	Usa SBOM	4%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Sim	10%
Facilidade de utilização		
Intuição	4/5 (1-Muito pouco intuitivo; 5-Muito bom)	2%
Requer o mínimo de treino possível	4/5 (1-Muito treino; 5-Pouco treino)	2%
Capacidades operacionais		
Velocidade de scans	Alguns segundos	1%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Tradicional	0.65%

Agnóstico a linguagens de programação	Sim. Possui mais de 200 linguagens de programação e SBOM	3%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	API e CLI	10%
Base de dados de vulnerabilidades de segurança		
Tamanho	Mais de 300 000 componentes vulneráveis. Permite ter vulnerabilidades de um largo conjunto de projetos.	4%
Frequência de atualização	Diária	2%
Fontes de informação	• CVE/NVD • Rastreador de Problemas do Github • Alertas de Segurança • Rastreadores de Problemas de projetos *open-source*. A base de dados de vulnerabilidades monitoriza continuamente múltiplos recursos, incluindo o NVD e uma vasta gama de Alertas de Segurança e Rastreadores de Problemas. CVE Numbering Authority - Mend é uma Autoridade de Numeração de CVE, que lhes permite revelar responsavelmente novas vulnerabilidades de segurança encontradas através da sua própria investigação.	7%
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Sim	2%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Não	0%
Custos associados		
Infraestrutura	5/5 (1 = Mais barato; 5 = Mais caro)	0%
Operacional	2/5 (1 = Mais barato; 5 = Mais caro)	1.875%
Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	3/5 (1 = Mais barato; 5 = Mais caro)	0.75%
Personalização de engenharia	1/5 (1 = Mais barato; 5 = Mais caro)	1%
Métodos de implementação		

On-site	Sim	5%
Nuvem	Sim	5%
Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	Sim. Excel e XML (alguns relatórios também suportam a exportação para formatos HTML ou JSON).	1.5%
Total		83.275%

5.7 CAST Highlight

Na Tabela XV é efetuada a avaliação da ferramenta CAST Highlight de acordo com o procedimento descrito.

Tabela XV - Avaliação da ferramenta CAST Highlight

Base de conhecimentos	Recolha de informação	%
Tamanho	Os dados da SBOM podem ser automaticamente exportados em vários formatos, incluindo normas da indústria e opções flexíveis, tais como: • CycloneDX • Word • Excel • PPT • XML • REST API Deteta automaticamente todas as *frameworks* de código aberto e componentes de terceiros a partir de uma base de conhecimento proprietária de mais de 100 milhões de componentes. A CAST consolida uma base de dados única feita de 94M+ componentes o*pen-source* e 9B+ impressões digitais de ficheiros. Para constituir a base de conhecimentos sobre código *open-source*, rastreiam continuamente diferentes plataformas, seja para código fonte (atualmente suportado: Github, GitLab, Salsa Debian, Framagit) ou pacotes/binários (atualmente suportados: Maven, NPM, NuGet, PyPi, Packagist, RubyGem...).	5.5%

	Ferramentas e ficheiros de gestão de dependência atualmente suportados: • Ant (build.xml) • Bauer (bauer.json) • Go (Go.mod, Go.sum) • Composer (composer.json) • Go (Go.mod, Go.sum) • Gradle (build.gradle, dependencies.gradle) • Maven (pom.xml) • NPM (package.json e package-lock.json) • Python (requirements.txt, setup.py) • Ruby (Gemfile.lock) • Visual Studio (.vcproj, .csproj) • Yarn (yarn.lock) Os dados da SBOM podem ser automaticamente exportados em vários formatos, incluindo padrões industriais e opções flexíveis tais como CycloneDX, Word, Excel, PPT, XML, e REST API. O CAST Highlight deteta mais de 350 licenças *open-source*.	
Frequência de atualização	Várias vezes ao dia	3.75%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Sim	10%
Facilidade de utilização		
Intuição	5/5 (1-Muito pouco intuitivo; 5-Muito bom)	2.5%
Requer o mínimo de treino possível	4/5 (1-Muito treino; 5-Pouco treino)	2%
Capacidades operacionais		
Velocidade de scans	Alguns minutos	0.5%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Tradicional	0.65%
Agnóstico a linguagens de programação	Não. 46 Linguagens de programação	2.5%
Capacidades de integração		

Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	API e CLI	10%
Base de dados de vulnerabilidades de segurança		
Tamanho	Por volta de 150 mil vulnerabilidades conhecidas. Permite ter vulnerabilidades de um largo conjunto de projetos.	4%
Frequência de atualização	Informação não disponibilizada	0%
Fontes de informação	Dependendo do nome do componente e da versão detetada num scan da aplicação, o CAST Highlight encontra possíveis vulnerabilidades (CVEs) cruzando a Base de Dados Nacional de Vulnerabilidade (NVD) do NIST, através de mais de 150 mil vulnerabilidades conhecidas. Os utilizadores normalmente executam uma primeira análise da sua aplicação para estabelecer uma linha de base CVE e dar prioridade a ações de mitigação de risco.	3%
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Não	0%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Não	0%
Custos associados		
Infraestrutura	4/5 (1 = Mais barato; 5 = Mais caro)	0.875%
Operacional	3/5 (1 = Mais barato; 5 = Mais caro)	1.25%
Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	3/5 (1 = Mais barato; 5 = Mais caro)	0.75%
Personalização de engenharia	2/5 (1 = Mais barato; 5 = Mais caro)	0.75%
Métodos de implementação		
On-site	Não	0%
Nuvem	Sim	5%
Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	Sim. CSV	1.5%
Total		69.525%

5.8 Contrast Security

Na Tabela XVI é efetuada a avaliação da ferramenta Contrast Security de acordo com o procedimento descrito.

Apesar desta ferramenta possuir SBOM, a documentação da mesma contradiz-se, pois apresenta 11 linguagens de programação suportadas. Logo, a avaliação do critério "Agnóstico a linguagens de programação" foi cotada com 2.5% pelas dúvidas levantadas.

Tabela XVI - Avaliação da ferramenta Contrast Security

Base de conhecimentos	Recolha de informação	%
Tamanho	SBOM	6%
Frequência de atualização	Não aplicável	4%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Sim	10%
Facilidade de utilização		
Intuição	5/5 (1-Muito pouco intuitivo; 5-Muito bom)	2.5%
Requer o mínimo de treino possível	4/5 (1-Muito treino; 5-Pouco treino)	2%
Capacidades operacionais		
Velocidade de scans	Alguns minutos	0.5%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Tradicional	0.65%
Agnóstico a linguagens de programação	11 linguagens de programação suportadas: Java, Javascript, .NET Framework, .NET Core, Node.js, Python, Ruby, Go, PHP, Scala e Kotlin	2.5%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	CLI e API	10%
Base de dados de vulnerabilidades de segurança		
Tamanho	Não há limite de aplicações que podem ser analisadas simultaneamente, continuamente e em tempo real.	3%
Frequência de atualização	Diária	2%
Fontes de informação	Os fundadores e funcionários de contraste são os fundadores do OWASP e de muitos dos projetos OWASP de maior sucesso, incluindo os Top 10 do OWASP, ESAPI, WebGoat, XSS Prevention Cheat Sheet, e outros. Todos os funcionários da Contrast gastam tempo na	6.5%

	investigação de segurança, que partilhamos abertamente através da OWASP. Além disso, a Contrast estabeleceu uma parceria com a Aspect Security, uma empresa líder em segurança de aplicações, para partilhar conhecimentos de investigação/consultoria. A Aspect tem várias dezenas de investigadores/consultores que analisam centenas de milhões de linhas de código e descobrem milhares de vulnerabilidades todos os anos para as maiores e mais importantes instituições financeiras, governamentais e comerciais de todo o mundo. Através desta relação, a Contrast tem acesso às vulnerabilidades em aplicações que conduzem a novas regras úteis.	
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Não	0%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim	5%
Custos associados		
Infraestrutura	3/5 (1 = Mais barato; 5 = Mais caro)	1.75%
Operacional	3/5 (1 = Mais barato; 5 = Mais caro)	1.25%
Licenciamento	3/5 (1 = Mais barato; 5 = Mais caro)	0.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	1/5 (1 = Mais barato; 5 = Mais caro)	1.5%
Personalização de engenharia	5/5 (1 = Mais barato; 5 = Mais caro)	0%
Métodos de implementação		
On-site	Sim	5%
Nuvem	Sim	5%
Outros		
Instalação modular	Não	0%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	Sim. PDF	1.5%
Total		**82.65%**

5.9 Snyk

Na Tabela XVII é efetuada a avaliação da ferramenta Snyk de acordo com o procedimento descrito.

Tabela XVII - Avaliação da ferramenta Snyk

Base de conhecimentos	Recolha de informação	%
Tamanho	São suportados os seguintes gestores de pacotes: 1. deb 2. gomodules 3. gradle 4. maven 5. nuget 6. paket 7. pip 8. rpm 9. rubygems 10. cocoapods 11. npm 12. Yarn Gestores de pacotes| ferramentas de construção: • .NET (C#, F#, Visual Basic)| Nuget, Paket • Bazel| Ver documentos da API. • C / C++| N/A • Elixir| hex • Go| Go Modules, dep, govendor • Java| Gradle, Maven • JavaScript| npm, yarn • PHP| Composer • Python| pip, Poetry, pipenv • Ruby| Bundler • Scala| sbt • Swift e Objective-C| CocoaPods	4%
Frequência de atualização	Informação não disponibilizada	0%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Sim	10%
Facilidade de utilização		
Intuição	5/5 (1-Muito pouco intuitivo; 5-Muito bom)	2.5%
Requer o mínimo de treino possível	5/5 (1-Muito treino; 5-Pouco treino)	2.5%
Capacidades operacionais		

Velocidade de scans	Informação não disponibilizada	0%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Blind	0.68%
Agnóstico a linguagens de programação	15 linguagens de programação	2%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	CLI e API	10%
Base de dados de vulnerabilidades de segurança		
Tamanho	Permite ter vulnerabilidade de um largo conjunto de projetos	3%
Frequência de atualização	Várias vezes ao dia	3%
Fontes de informação	A maioria das vulnerabilidades da base de dados provem de uma destas fontes: - Monitorização de outras bases de dados de vulnerabilidades, tais como CVEs do NVD e muitas outras. - Monitorização das atividades dos utilizadores no GitHub, incluindo questões, PRs e mensagens de *commits* que podem indicar uma vulnerabilidade. - Pesquisa em massa, utilizando ferramentas que procuram erros repetidos de segurança pelo código de pacotes *open-source*. - Investigação manual, ao investir o tempo dos investigadores a auditar manualmente pacotes mais amplamente utilizados por falhas de segurança.	7.5%
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Sim	2%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Não	0%
Custos associados		
Infraestrutura	4/5 (1 = Mais barato; 5 = Mais caro)	0.875%
Operacional	2/5 (1 = Mais barato; 5 = Mais caro)	1.875

Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	3/5 (1 = Mais barato; 5 = Mais caro)	0.75%
Personalização de engenharia	5/5 (1 = Mais barato; 5 = Mais caro)	0%
Métodos de implementação		
On-site	Não	5%
Nuvem	Sim	5%
Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	Sim. PDF	1.5%
Total		**71.68%**

5.10 BlackDuck Software

Na Tabela XVIII é efetuada a avaliação da ferramenta BlackDuck Software de acordo com o procedimento descrito.

Tabela XVIII - Avaliação da ferramenta BlackDuck Software

Base de conhecimentos	Recolha de informação	%
Tamanho	• A deteção *open-source* de multifator da BlackDuck e a KnowledgeBase™ com mais de 4 milhões de componentes fornece uma BoM precisa para qualquer aplicação ou contentor; • Possui mais de 2750 licenças *open-source*; • Cobre mais de 5,1 milhões de componentes *open-source* de mais de 26.000 forjas e repositórios; • Mais de 5 100 000 projetos *open-source*; • 4,5 milhões de projetos *open-source* únicos. Gestores de Pacotes: • Bazel • Cargo • C/C++ (Clang) • Conan • Conda • Dart • Docker image • GoLang	5%

	• Gradle • Erlang/Hex/Rebar • Lerna • Maven • npm • NuGet • Python • The Pip detector • sbt • Execução em plano de fundo (A linha de comandos sbt) • Yarn • Yocto (BitBake)	
Frequência de atualização	De hora em hora	4%
Capacidades de deteção		
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Não	0%
Facilidade de utilização		
Intuição	4/5 (1-Muito pouco intuitivo; 5-Muito bom)	2%
Requer o mínimo de treino possível	4/5 (1-Muito treino; 5-Pouco treino)	2%
Capacidades operacionais		
Velocidade de scans	Alguns segundos	1%
Habilidade para usar scans M&A – sem bloqueio de licença em modelos de utilização	Sim	4%
Dar suporte a diferentes modelos de auditoria	Tradicional	0.65%
Agnóstico a linguagens de programação	Não. Mais de 90 linguagens de programação.	2.8%
Capacidades de integração		
Integração com sistemas de construção através de APIs e CLI (*Command Line Interface*)	API e CLI	10%
Base de dados de vulnerabilidades de segurança		
Tamanho	Mais de 174 000 vulnerabilidades únicas e permite ter múltiplos projetos.	4%
Frequência de atualização	Diária	2%
Fontes de informação	• NVD • Os Alerta de Segurança da Black Duck (*Black Duck Security Advisories* - BDSAs), dados que são pesquisados e analisados pela	5%

	Synopsys Cybersecurity Research Center (CyRC).	
Capacidade de priorizar vulnerabilidades	Sim	3%
Remediação automatizada	Sim	2%
Métodos de descoberta avançados		
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim	5%
Custos associados		
Infraestrutura	3/5 (1 = Mais barato; 5 = Mais caro)	1.75%
Operacional	2/5 (1 = Mais barato; 5 = Mais caro)	1.875%
Licenciamento	Sem custos	1.5%
Integração	1/5 (1 = Mais barato; 5 = Mais caro)	2.5%
Lock-in	3/5 (1 = Mais barato; 5 = Mais caro)	0.75%
Personalização de engenharia	4/5 (1 = Mais barato; 5 = Mais caro)	0.25%
Métodos de implementação		
On-site	Sim	5%
Nuvem	Sim	5%
Outros		
Instalação modular	Sim	1.5%
Geração de avisos necessários	Sim	2%
Capacidade de geração de relatórios	Sim. Ficheiro de texto, HTML, Excel e PDF.	1.5%
Total		**76.075%**

5.11 Discussão de resultados

Nesta secção são discutidos os resultados provenientes das secções anteriores, de modo a entender a efetividade do procedimento criado.

Na Tabela XIX podemos observar os resultados finais por ordem decrescente das suas classificações.

Tabela XIX - Classificações das ferramentas (por ordem decrescente)

Ferramenta	Classificação
WhiteSource Software (Mend)	83.275 %
BlackDuck Software	76.075%
Gitlab	73.445%
Snyk	71.68%
Dependency-Track	70.275%
CAST Highlight	69.525%
Github	59.825%
Dependency-Check	53.275%

Começando pelas ferramentas pedidas pela empresa onde se realiza este estágio, ambas são ferramentas totalmente gratuitas. No entanto, a OWASP Dependency-Check possui muito menos funcionalidades e capacidades que as restantes ferramentas. Relativamente à OWASP Dependency-Track é uma ferramenta mais completa, comparativamente às restantes existentes no mercado, ficando assim a ocupar a posição 5 desta classificação, tendo também ganho pontos no critério de custos por ser uma ferramenta gratuita que dispõe de poucos gastos adicionais ao utilizador. As posições acima desta ferramenta possuem mais funcionalidades, possuindo também bases de vulnerabilidades e de conhecimentos bastante completas, sendo que algumas incluem centros de investigação de vulnerabilidades próprios, fontes de informação privadas e *open-source*. É de notar que as ferramentas acima da OWASP Dependency-Track são pagas, logo perdem pontos no critério de custos, sendo que as suas funcionalidades se averiguam superiores e, portanto, compensam essa perda. No entanto, é de salientar que a diferença percentual entre a Dependency-Track, Snyk e Gitlab é baixa o que pode indicar preferência pela ferramenta de menor custo.

Visto isto, as ferramentas que mais se destacam é a BlackDuck Software e WhiteSource Software (Mend). A grande discrepância entre as duas deve-se ao facto de a ferramenta da BlackDuck ter uma fonte de conhecimentos mais limitada que a WhiteSource, não usar SBOM e suportar menos linguagens de programação (90 vs. 200) sendo que a WhiteSource ainda suporta SBOM. A ferramenta da WhiteSource é capaz de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado, reduzindo assim a quantidade de falsos positivos, contrariamente à da BlackDuck. Apesar de os custos de infraestrutura da WhiteSource serem ligeiramente mais elevados, em contrapartida, a BlackDuck possui elevados custos para a personalização de engenharia. Contudo, os resultados obtidos são provenientes de um determinado conjunto de pesos exemplificativos. Se o mesmo for alterado, consequentemente os resultados irão diferir consoante as preferências estipuladas pelo avaliador.

Para além dos resultados obtidos das ferramentas, é de notar o esforço dedicado na aplicação do procedimento. Para ponderar e definir os pesos de critérios do procedimento (usando ou não o AHP) leva um a dois dias, sendo que esta etapa dentro de uma organização deverá ser feita em grupo, podendo levar menos tempo ainda. Para a recolha de informação de cada ferramenta tem-se uma média de 2-3 dias para cada uma (feita individualmente), visto que a documentação das ferramentas pode ser complexa e pouco explícita em determinados critérios, mas também ter informações bastante objetivas noutros, dependendo da ferramenta a ser analisada.

Os critérios onde se notaram mais dificuldades em achar a sua respetiva informação foram os seguintes:

- Frequência de atualização da base de conhecimentos;
- Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado;
- Dar suporte a diferentes modelos de auditoria;
- Frequência de atualização da base de dados de vulnerabilidades de segurança;
- Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente).

Capítulo 6 Conclusão

Em suma, ao longo do trabalho desenvolvido pode-se constatar que a Análise de Composição de Software é um tema de segurança informática recente, que se encontra em constante evolução e que proporciona grandes benefícios ao mercado atual, onde existem dependências diretas e indiretas entre variados tipos de software e as cadeias de fornecimento de software estão sujeitas a ataques devastadores que podem desencadear graves repercussões. Na linha de seguimento deste trabalho é de notar que não foi encontrado nenhum procedimento exato e objetivo, sem limitações, de outros entendidos do tema para a avaliação das ferramentas de SCA, no entanto, caminha-se para o encontro da mesma como se pôde observar pela revisão de literatura.

O objetivo deste trabalho é de ampla importância para qualquer desenvolvedor de software, na medida em que providencia um procedimento de avaliação para que se possa escolher uma ferramenta de SCA adequada ao software com requisitos específicos a serem atendidos, de modo a proteger as dependências com o seu software. Este procedimento foi deliberadamente construído de modo flexível, com o intuito de não limitar avaliadores que pretendam utilizá-lo, tanto pelos pesos de critérios definidos neste trabalho, como à evolução tecnológica constante que poderá alterar a relevância dos critérios. Para além disto, inclui-se também uma proposta de trabalho futuro para este procedimento focado em critérios práticos, leia-se, critérios a notar durante a utilização das ferramentas. Para além do procedimento apresentado também é possível estendê-lo utilizando critérios práticos. Isto é, testar as ferramentas e comparar resultados do seu funcionamento. Para tal, são propostas *benchmarks* com vulnerabilidades conhecidas como a "OWASP Benchmark Project" [43] e "Go Test Bench" [44]. Aplicando-se estas *benchmarks* às ferramentas de SCA podem ser definidos novos critérios de utilização como:

- Quantidade de vulnerabilidades detetadas (sabendo o total das mesmas pela informação disponibilizada nas *benchmarks*)
- Quantidade de falsos positivos
- Facilidade de utilização da ferramenta
- Velocidade de scans
- Qualidade das métricas disponibilizadas
- Qualidade de remediações automáticas

Verificou-se a efetividade do procedimento proposto através das avaliações realizadas a uma gama de ferramentas de SCA atualmente utilizadas no mercado. Através dos resultados obtidos foi possível comparar as ferramentas tendo em conta as suas bases de conhecimentos, capacidades de deteção, operacionais e de integração, a sua facilidade de utilização, as suas bases de dados de vulnerabilidades, se possuem métodos de descoberta avançados, os custos associados, os seus modelos de implementação, entre outros. Por fim, estes resultados foram discutidos de modo a entender quais as ferramentas são as mais adequadas para serem aplicadas na proteção de cadeias de fornecimento dentro dos critérios definidos.

Ao se realizar este procedimento notaram-se determinadas limitações e dificuldades, sobretudo ao definir os critérios e a procurar informações dos mesmos para cada uma das ferramentas. Como tal, a recolha de informação sobre as ferramentas foi a fase mais demorada, pois à medida que ia sendo feita, foram-se alterando os critérios conforme as informações encontradas e não disponibilizadas. O que mais dificultou esta fase foi uma

vasta, mas pouco conclusiva gama de informação, a que acrescia o facto de se encontrar demasiadamente dispersa e, portanto, dificultar, ao avaliador, a sua apreensão e recolha em tempo útil.

Referências

[1] Scott Treacy, «Software Composition Analysis and Defense-in-Depth», *Journey Notes*, 2021. https://blog.barracuda.com/2021/08/20/software-composition-analysis-and-defence-in-depth/ (acedido 2 de dezembro de 2021).

[2] ichi.pro, «Análise de composição de software (SCA)», *ICHI.PRO*, 2020. https://ichi.pro/pt/analise-de-composicao-de-software-sca-138912967469454 (acedido 20 de dezembro de 2021).

[3] Philippe Ombredanne, «Free and Open Source Software License Compliance: Tools for Software Composition Analysis», *Computer*, vol. 53, n. 10, pp. 105–109, out. 2020, doi: 10.1109/MC.2020.3011082.

[4] Nasif Imtiaz, Seaver Thorn, e Laurie Williams, «A comparative study of vulnerability reporting by software composition analysis tools», em *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Bari Italy, out. 2021, pp. 1–11. doi: 10.1145/3475716.3475769.

[5] Ibrahim Haddad, «Open Source Compliance In The Enterprise». 2018.

[6] Ibrahim Haddad, «An Open Guide To Evaluating Software Composition Analysis Tools», p. 13, nov. 2020.

[7] Aleksandar Dimov e Vladimir Dimitrov, «Classification of Software Security Tools», p. 11, 2021.

[8] Darius Foo, Jason Yeo, Hao Xiao, e Asankhaya Sharma, «The Dynamics of Software Composition Analysis», *ArXiv190900973 Cs*, set. 2019, Acedido: 12 de janeiro de 2022. [Em linha]. Disponível em: http://arxiv.org/abs/1909.00973

[9] Yang Chen, Andrew E. Santosa, Asankhaya Sharma, e David Lo, «Automated identification of libraries from vulnerability data», em *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, Seoul South Korea, jun. 2020, pp. 90–99. doi: 10.1145/3377813.3381360.

[10] Steve Mansfield-Devine, «DevOps: finding room for security | Elsevier Enhanced Reader», 2018. https://reader.elsevier.com/reader/sd/pii/S1353485818300709?token=FFD7E5929999F0486748A0DA70F6A9AB833EAC23562F5DAABF1BF1E359ABD513165E9FB81982660869911EEC3FE6C01B&originRegion=eu-west-1&originCreation=20220110232934 (acedido 12 de janeiro de 2022).

[11] Daniel Berman, «Software composition analysis | SCA Guide 2021 | Snyk», 27 de janeiro de 2021. https://snyk.io/blog/what-is-software-composition-analysis-sca-and-does-my-company-need-it/ (acedido 12 de janeiro de 2022).

[12] Veracode, «The Dangers of Open Source Risk», *Veracode*, 2020. https://www.veracode.com/security/dangers-open-source-risk (acedido 2 de abril de 2022).

[13] Stuart Millar, «Vulnerability Detection in Open Source Software: The Cure and the Cause», p. 6, 2017.

[14] FOSSA Editorial Team, «The Complete Guide to Software Composition Analysis - FOSSA», 2021. https://fossa.com//complete-guide-software-composition-analysis (acedido 12 de janeiro de 2022).

[15] Julie Peterson, «AST: Application Security Testing», *WhiteSource*, 27 de agosto de 2020. https://www.whitesourcesoftware.com/resources/blog/ast-application-security-testing/ (acedido 12 de janeiro de 2022).

[16] Simon Roe, «SAST, DAST, SCA: What's best for application security testing? | Outpost 24 blog», 30 de março de 2021. https://outpost24.com/blog/SAST-DAST-SCA-what-s-best-for-application-security-testing (acedido 12 de janeiro de 2022).
[17] Revenera, «What is Software Composition Analysis?», *Revenera Blog*, 6 de outubro de 2020. https://www.revenera.com/blog/what-is-software-composition-analysis/ (acedido 14 de janeiro de 2022).
[18] «ISO/IEC/IEEE International Standard - Software engineering - Recommended practice for software acquisition», IEEE, 2019. doi: 10.1109/IEEESTD.2019.8645777.
[19] Wikipedia, «Typosquatting», *Wikipedia*. 27 de dezembro de 2021. Acedido: 10 de outubro de 2021. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Typosquatting&oldid=1062305263
[20] FOSSA Editorial Team, «SolarWinds, Supply Chain Attacks, and Software Composition Analysis - FOSSA», *Dependency Heaven*, 23 de dezembro de 2020. https://fossa.com/blog/solarwinds-supply-chain-attacks-software-composition-analysis/ (acedido 2 de dezembro de 2021).
[21] Stephen Pritchard, «Software supply chain attacks – everything you need to know», *The Daily Swig | Cybersecurity news and views*, 11 de fevereiro de 2021. https://portswigger.net/daily-swig/software-supply-chain-attacks-everything-you-need-to-know (acedido 2 de dezembro de 2021).
[22] Zach Schneider, «event-stream vulnerability explained», *event-stream vulnerability explained*, 27 de outubro de 2018. https://schneider.dev/blog/event-stream-vulnerability-explained/ (acedido 4 de dezembro de 2021).
[23] GReAT e AMR, «Operation ShadowHammer», 25 de março de 2019. https://securelist.com/operation-shadowhammer/89992/ (acedido 4 de dezembro de 2021).
[24] Joseph Cox, «The NSA and CIA Use Ad Blockers Because Online Advertising Is So Dangerous». https://www.vice.com/en/article/93ypke/the-nsa-and-cia-use-ad-blockers-because-online-advertising-is-so-dangerous (acedido 9 de dezembro de 2021).
[25] Brad Smith, «A moment of reckoning: the need for a strong and global cybersecurity response», *Microsoft On the Issues*, 18 de dezembro de 2020. https://blogs.microsoft.com/on-the-issues/2020/12/17/cyberattacks-cybersecurity-solarwinds-fireeye/ (acedido 4 de dezembro de 2021).
[26] mindsecblog, «Entenda o ataque hacker da SolarWinds que assombrou o mundo», 20 de dezembro de 2020. https://minutodaseguranca.blog.br/entenda-o-ataque-hacker-da-solarwinds-que-assombrou-o-mundo/ (acedido 11 de dezembro de 2021).
[27] mimecast, «Important Update from Mimecast», *Mimecast*, 12 de janeiro de 2021. https://www.mimecast.com/blog/important-update-from-mimecast/ (acedido 11 de dezembro de 2021).
[28] Aaron Kraus, «Mimecast hacked: How the Microsoft 365 email attack impacts users | Coalition», 14 de janeiro de 2021. https://www.coalitioninc.com/blog/mimecast-certificate-hacked-how-the-microsoft-365-email-attack-impacts-users (acedido 28 de dezembro de 2021).
[29] Jessica Haworth, «Researcher hacks Apple, Microsoft, and other major tech companies in novel supply chain attack», *The Daily Swig | Cybersecurity news and views*, 10 de fevereiro de 2021. https://portswigger.net/daily-swig/researcher-hacks-apple-microsoft-and-other-major-tech-companies-in-novel-supply-chain-attack (acedido 12 de dezembro de 2021).
[30] Microsoft, «3 Ways to Mitigate Risk When using Private Package Feeds - v1.0». 22 de março de 2021. Acedido: 12 de dezembro de 2021. [Em linha]. Disponível em: https://azure.microsoft.com/mediahandler/files/resourcefiles/3-ways-to-mitigate-risk-using-private-package-

feeds/3%20Ways%20to%20Mitigate%20Risk%20When%20Using%20Private%20Package%20Feeds%20-%20v1.0.pdf
[31] James Wetter e Nicky Ringland, «Understanding the Impact of Apache Log4j Vulnerability», *Google Online Security Blog*, 17 de dezembro de 2021. https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html (acedido 20 de dezembro de 2021).
[32] Google, «GHSA-jfh8-c2jp-5v3q | GHSA | Open Source Insights». https://deps.dev/advisory/GHSA/GHSA-jfh8-c2jp-5v3q (acedido 20 de dezembro de 2021).
[33] Google, «GHSA-7rjr-3q55-vv33 | GHSA | Open Source Insights». https://deps.dev/advisory/GHSA/GHSA-7rjr-3q55-vv33 (acedido 20 de dezembro de 2021).
[34] Nicole Sganga, «Nightmare before Christmas: What to know about the Log4j vulnerability». https://www.cbsnews.com/news/log4j-vulnerability-breach-patch/ (acedido 20 de dezembro de 2021).
[35] Y. Striem-Amit, «UPDATED: Cybereason Log4Shell Vaccine Offers Permanent Mitigation Option for Log4j Vulnerabilities (CVE-2021-44228 and CVE-2021-45046)». http://www.cybereason.com/blog/cybereason-releases-vaccine-to-prevent-exploitation-of-apache-log4shell-vulnerability-cve-2021-44228 (acedido 20 de dezembro de 2021).
[36] Flavia, «O que é SCA - Software Composition Analysis? - Nova8 O que é SCA - Software Composition Analysis?», *Nova8*, 3 de dezembro de 2020. https://www.nova8.com.br/2020/12/03/o-que-e-sca-software-composition-analysis/ (acedido 10 de outubro de 2021).
[37] Tomas Statkus, «How Do Software Composition Analysis Tools Work?», 26 de julho de 2021. https://reviewedbypro.com/how-do-software-composition-analysis-tools-work/ (acedido 20 de dezembro de 2021).
[38] Gareth Rushgrove, «Introduction to Software Bill of Materials and CycloneDX», *DevSecCon*, 7 de dezembro de 2020. https://www.devseccon.com/blog/a-quick-introduction-to-software-bill-of-materials-and-cyclonedx-secadvent-day-7/ (acedido 6 de março de 2022).
[39] Vagisha Arora, «What Does "Apache 2.0 License" Mean?», *Planet Crust*, 30 de novembro de 2021. https://www.planetcrust.com/what-does-apache-2-0-license-mean/?utm_campaign=blog (acedido 7 de julho de 2022).
[40] Snyk, «What is the MIT License? Top 10 questions answered | Snyk», 26 de julho de 2020. https://snyk.io/learn/what-is-mit-license/ (acedido 7 de julho de 2022).
[41] Alessio Ishizaka e Ashraf Labib, «Review of the main developments in the analytic hierarchy process», *Expert Syst. Appl.*, p. S0957417411006701, mai. 2011, doi: 10.1016/j.eswa.2011.04.143.
[42] Jeneé A. Jagoda, Steven J. Schuldt, e Andrew J. Hoisington, «What to Do? Let's Think It Through! Using the Analytic Hierarchy Process to Make Decisions», *Frontiers for Young Minds*, 18 de junho de 2020. https://kids.frontiersin.org/articles/10.3389/frym.2020.00078 (acedido 5 de setembro de 2022).
[43] OWASP, «OWASP Benchmark | OWASP Foundation». https://owasp.org/www-project-benchmark/ (acedido 4 de agosto de 2022).
[44] Contrast Security OSS, «Go Test Bench». Contrast Security OSS, 21 de julho de 2022. Acedido: 4 de agosto de 2022. [Em linha]. Disponível em: https://github.com/Contrast-Security-OSS/go-test-bench

Apêndices

Apêndice A

Neste apêndice são apresentadas as métricas de avaliação de ferramentas de SCA criadas pela OWASP, demonstradas na Tabela XVIII.

Tabela XX – Métricas de avaliação de ferramentas de SCA

Métrica	Descrição
1. Base de conhecimento	▪ Medir a dimensão da base de conhecimentos a partir do número de projetos *open source* e o número de ficheiros detetados. Quanto maior a base de conhecimento sobre informações de software *open-source*, mais código *open source* será identificado no *scanning*. ▪ Listar os principais repositórios detetados (por exemplo, todos do NPM, SourceForge). ▪ Determinar quais os ecossistemas a serem detetados (por exemplo, R, Delphi). ▪ Identificar quais idiomas pertencem ao escopo da ferramenta (com base na extensão e o tipo de repositório). O scanner deve ser agnóstico na linguagem, na prática, poucos fornecedores possuem este suporte, daí a necessidade de esclarecer as linguagens suportadas. ▪ Distinguir da deteção a nível de pacote (exemplo, Maven) e de suporte "Java" (por exemplo, é possível encontrar dependências jar mas não realizar o *scan* dos ficheiros fonte .java para direitos autorais e informações de licenças). ▪ Identificar a frequência de atualização da base de conhecimento. As atualizações regulares são necessárias para acompanhar o desenvolvimento de código *open source*. Os serviços de conformidade e fornecedores de ferramentas atualizam as suas bases de dados regularmente. Algumas empresas atualizam 3 a 4 vezes por ano, outras realizam com ainda mais frequência (até diariamente). O ideal é conseguir a maior e mais atualizada base de dados para aumentar as chances de identificar código *open source* recentemente criado. ▪ Responder às seguintes questões: o Quanto tempo leva o pedido de um cliente a ser adicionado à base de conhecimento?

	○ Existe um SLA (*Service-Level Agreement*) para pedidos? ○ Qual é o processo usado?
2. Capacidades de deteção	▪ Identificar os componentes detetados por completo ▪ Verificar a capacidade de corrigir/configurar o analisador. Isto é, projetos de software são complexos, com diferentes configurações de construção e é necessária uma maneira de configurar a ferramenta de modo a captar a realidade das condições do projeto. ▪ Identificar a metodologia de deteção utilizada. Diferentes abordagens de análise possuem prós e contras. O criador da ferramenta deve resumir como a deteção da ferramenta funciona para cada *scanner* utilizado. ▪ Identificar os fragmentos parciais detetados (variando de poucas linhas a um ficheiro parcial). ▪ Identificar as opções oferecidas para corrigir e verificar resultados. Suporta a capacidade de classificar resultados (como exemplo, P1 ou Serious). ▪ Determinar a capacidade de auto identificar código com a origem própria e licença sem a necessidade de um engenheiro de conformidade direcionar a ferramenta sobre o que é uma correspondência correta e o que é um falso positivo. Muitos dos motores de *scannings* do código fonte, principalmente os que suportam fragmentos, geram um número significativo de falsos positivos que necessitam de ser investigados e devem ser corrigidos manualmente. As horas de trabalho manual geradas por estes falsos positivos são um problema constante com alguns dos produtos mais conhecidos no mercado atual. ▪ Identificar o tipo de análise suportada (distinção entre deteção a nível de pacote e deteção "exata" do estilo de ficheiro utilizado para descobrir cópias ficheiros únicos da fonte, binários e ficheiros multimédia). ○ *Scanners* da fonte (código → quais pacotes OSS-*Open Source Software*?) ○ *Scanners* binários (binário → quais pacotes OSS?) ○ *Scanners* de fragmentos (código fragmentado → copiado de quais componentes OSS?) ○ *Scanners* de dependências (código → quais dependências são incluídas via gestor de pacotes?)

	○ *Scanners* de licença (código → licenças OSS?) ○ Quais linguagens são suportadas? Se uma linguagem é suportada, inclui a análise de fragmentos? Apenas a nível de pacote? Correspondência exata de ficheiros? ○ *Scanners* de segurança (código → vulnerabilidades?) ○ Outras técnicas de deteção de vulnerabilidades incluem termos de pesquisa, deteção de email/URL, deteção de serviços web, etc?
3. Facilidade de utilização	▪ Quantificar a intuição do design e interface do utilizador. ▪ Identificar se possui a disponibilidade de cliente local ou *plugin* para *browser*. ▪ Identificar se possui cliente móvel. ▪ Requer formação mínima ou nenhuma para correr a ferramenta, mas é dada formação para compreender como examinar e avaliar os resultados gerados. Nota: A facilidade de utilização é um critério subjetivo e difícil de quantificar e qualificar, no entanto, algumas ferramentas são de mais fácil utilização que outras. Este critério será avaliado através de opiniões e comentários de utilizadores das ferramentas.
4. Capacidades Operacionais	▪ Determinar a velocidade dos *scans* do código fonte. ○ A velocidade dos *scans* do código fonte é um dos maiores problemas das ferramentas atuais. ○ Como exemplo, é possível ter uma empresa que concebeu e desenvolveu a sua própria base de dados que se torna perfeitamente adequada para manipular um tipo de dados específicos. Por conseguinte, o *scan* é exponencialmente rápido do que outras ferramentas por ser dedicado a algo específico. Esta rapidez também é útil quando para quando se integra a ferramenta de *scanning* com o processo de integração contínua. ○ Outra questão a ter em consideração é se há ficheiros a serem ignorados. ○ Algo a ter em consideração para além do mencionado é saber se os direitos de autor e a deteção de licenças exatas estão a ser detetados ou se o realiza apenas o *scan* de repositórios/pacote de ficheiros de gestão como o "pom.xml".

	▪ Verificar a capacidade de utilizar a ferramenta para *scans* relacionados com atividades M&A (*Mergers and Acquisitions*) sem um bloqueio de licenças nos modelos de utilização. Alguns vendedores de ferramentas impõem limitações através do acordo de licenciamento à capacidade de usar a ferramenta em cenários para além de *scans* de código relacionado com esforços de desenvolvimento em curso. ▪ Identificar o apoio a diferentes modelos de auditoria. Podem ser: o Auditoria Tradicional o Auditoria às cegas (*blind*) – considerada a auditoria mais segura e modelo de auditoria mais privativo em modelos M&A. o DIY (*Do It Yourself*) ▪ Possuir linguagem de programação agnóstica. Algumas ferramentas são, por admissão dos seus criadores, bastante eficazes a funcionar com linguagens de programação específicas. O ideal seria possuir um motor de identificação e *scanning* agnóstico a linguagens de programação. A maioria das ferramentas não é agnóstica, mas existem algumas. ▪ Verificar a capacidade de reutilizar clarificação de *scan* por toda a organização. ▪ Possuir um sistema de construção (CD/CI – *Continuous Delivery*/*Continuous Integration*) agnóstico.
5. Capacidades de integração	▪ Fornecer APIs e interface de linha de comandos (CLI – *Command-Line Interface*) para fácil integração. A utilização de uma ferramenta de *scanning* não é limitada apenas à UI (*User Interface*). Idealmente, as empresas desejam integrar a ferramenta com o desenvolvimento existente e na construção de sistemas e processos. Este cenário é exequível se a ferramenta de *scanning* suportar APIs e CLI o que permite aos administradores do sistema interagir com a ferramenta fora da UI. ▪ Suportar as capacidades de integração da UI. ▪ Possuir a capacidade de integrar políticas de conformidade de uma organização na ferramenta e ter o "*rule flag code*" no que diz respeito às políticas e regras declaradas
6. Base de dados de	▪ Determinar a dimensão da base de dados de vulnerabilidades. O número de vulnerabilidades detetadas em todos os projetos. A base de dados contém

vulnerabilidades de segurança	informações sobre vulnerabilidades se segurança conhecidas que permitem à ferramenta detetar problemas relacionados com a segurança do código fonte, não apenas o código fonte aberto, mas código fonte completo. Isto deve-se ao facto de os programadores poderem ter copiado partes de código de componentes de código *open source* para componentes proprietários ou de terceiros. Assim, se esse código contiver uma vulnerabilidade de segurança conhecida, aquando do *scanning* do componente proprietário, o motor deverá ser capaz de assinalar a vulnerabilidade existente. ▪ Determinar a frequência de atualização da base de dados de vulnerabilidades. Os fornecedores de serviços atualizam regularmente as suas bases de dados e quanto mais frequente for o ciclo de atualização, melhor será encontrar vulnerabilidades logo a estas terem sido publicamente identificadas. ▪ Identificar o número de fontes de informação sobre vulnerabilidades. Podem ser usadas variadas fontes para enriquecer a base de dados de vulnerabilidades de segurança em componentes *open source*. Ao avaliar as ferramentas de conformidade deste serviço, é recomendado investigar este aspeto e as mecânicas das atualizações utlizadas. As múltiplas fontes (diretas e indiretas) são usadas para recolher informação sobre as vulnerabilidades e recomendações para corrigir as mesmas. ▪ Verificar se a ferramenta possui pesquisas adicionais realizadas pelo fornecedor da ferramenta para validar alertas de vulnerabilidades. ▪ Calcular/Determinar a precisão da ferramenta, a taxa de vulnerabilidade detetadas que são Verdadeiros Positivos. Para tal existem 4 níveis de Verdadeiros Positivos: o O software vulnerável foi corretamente mapeado para uma dependência que é, na verdade, utilizado no nosso software proprietário. o A dependência utilizada é utilizada num ambiente crítico (tempo de execução). o O software proprietário chama a parte vulnerável da dependência num ambiente crítico. o A vulnerabilidade é explorável.

	▪ Determinar o *Recall* (Quanto do potencial universo de Verdadeiras vulnerabilidades é encontrado e se adequa corretamente ao software proprietário?) Na prática isto é impossível de se saber, pois as comparações entre diferentes soluções servem para estimar quais soluções têm o maior *recall* para uma pilha de tecnologia específica. ▪ Indagar a capacidade de priorizar vulnerabilidades contextualmente. As pontuações gerais atribuídas à severidade da vulnerabilidade, como CVSS3, podem ser imprecisas dependendo do ambiente de software proprietário. Os utilizadores devem ser capazes de contextualizar a gravidade das vulnerabilidades para garantir maior precisão à resolução dessas ameaças de segurança.
7. Método de descoberta de vulnerabilidades avançadas	▪ Suportar a deteção avançada de vulnerabilidades. Identificar uma vulnerabilidade quando o código vulnerável foi copiado para um novo componente (requere o suporte da identificação de fragmentos de código fonte)
8. Custos associados	▪ Custo de infraestrutura: custos da infraestrutura informático relacionados com o alojamento da solução ou com a utilização via ambiente *cloud*. o Considerar a utilização de servidores que os clientes têm de comprar, instalar e manter, inclui o custo de atualização dessa infraestrutura e, dependendo do tamanho da infraestrutura, o custo de um administrador de sistema associado. ▪ Custo operacional: custo associado à gestão dos resultados que a ferramenta proporciona. Implica inspecionar e interpretar os resultados e tomar as medidas necessárias. Uma ferramenta que auto identifique os falsos positivos baixará o custo de mão de obra para identificar manualmente os milhares de falsos positivos gerados. ▪ Custo anual de licenciamento: O custo da licença anual do software para utilização da ferramenta, custo de acesso ao SDK (*Software Development Kit*) para que possa interagir as ferramentas internas com o motor de *scanning*, e possivelmente, o custo de qualquer personalização exclusiva que se queira introduzir para se adaptar às necessidades.

	▪ Custo inicial para integrar com as ferramentas e infraestruturas de engenharia/TI existentes: os custos de integração são difíceis de estimar, mas tipicamente evoluem sobre a capacidade de integrar a ferramenta em fluxos de trabalho e processos com o mínimo de disrupções. ▪ Averiguar a capacidade de exportar projetos e outras informações, quer para migração para um novo sistema, como para preservar o conhecimento se deixar o fornecedor. ▪ Custo *lock-in* (o custo de ter de abandonar uma solução e adotar outra): as empresas regularmente ignoram ou não tomam atenção suficiente ao fator "*lock-in*" e aos custos associados à construção de todo um ambiente de conformidade em torno de uma ferramenta específica. É recomendado escolher uma nova ferramenta que tenha este custo em consideração. ▪ Custo de personalização da engenharia para atender necessidades específicas
9. Suporte para modelos de implementação	▪ Oferecer suporte para vários modelos de entrega: o Apenas no local o Apenas em nuvem o Híbrido ▪ Determinar quais informações sobre código e projetos saem das suas redes. Deve ser claro para o utilizador final: o O conteúdo real da fonte e dos ficheiros binários o O conteúdo parcial do ficheiro/*string* o As *Hashes* o As listas de inventário o A informação sobre a política o O estado de conformidade ou de não conformidade
10. Capacidades de comunicação	▪ Investigar a capacidade de gerar avisos necessários de conformidade. São avisos baseados em resultados de *scan* reais ou apenas retirados da informação sobre a licença da base de conhecimentos. ▪ Determinar se os direitos de autor/licenças reais dos subcomponentes ou subficheiros estão incluídos nos avisos. ▪ Averiguar a existência de avisos para fragmentos de código *open source*.

	▪ Verificar o suporte para diferentes capacidades de relatórios – capacidade de exportar em vários formatos, como Excel/Spreadsheet (incluindo a disponibilidade de uma amostra de um relatório detalhado) ▪ Suportar formatos padrão abertos (SPDX, SARIF, CVE, CVSS, …)

Apêndice B

Neste apêndice é apresentado as tabelas de apoio, o raciocínio e a pesquisa efetuada para recolher a informação de cada ferramenta. Como tal, a apresentação das tabelas é informal com incoerências entre português e inglês, links, entre outros.

	Dependency-check
Base de conhecimentos	
Tamanho	File type analyzers: https://jeremylong.github.io/DependencyCheck/analyzers/index.html Languages supported: Java, Javascript, .NET
Frequência de atualização	A única métrica que consigo retirar é a data das releases. Podes consultá-la aqui: https://github.com/jeremylong/DependencyCheck/releases A assunção que faço aqui é que quando surgem novos ecossistemas em *open-source*, o dependency-check terá uma nova release para acompanhar as mudanças.
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Não
Facilidade de utilização	
Intuição	4/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	4/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	Não sei. Sugiro fazeres download de 4 ou 5 projetos *open-source* (java, javascript ou .net) e correres o dependency-check contra eles. Depois retiras uma métrica tipo tempo por linha de código, ou algo do género.
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim (Licença é a Apache License 2.0)
Dar suporte a diferentes modelos de auditoria	Traditional apenas

	Dependency-check
Agnóstico a linguagens de programação	Não
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	Só permite ter vulnerabilidades do projeto a ser analisado
Frequência de atualização	Sempre que a ferramenta corre, verifica os data feeds da NVD (https://nvd.nist.gov/vuln/data-feeds), o que a permite descobrir vulnerabilidades muito recentes
Fontes da informação	NVD NPM Audit API, the OSS Index, RetireJS, Bundler Audit
Capacidade de priorizar vulnerabilidades	Sim
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Não
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Não
Custos associados	
infraestrutura	No costs
operacional	4/5 (1=Cheapest;5=Most expensive) Generates too many false positives
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)

	Dependency-check
lock-in	1/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	1/5 (1=Cheapest;5=Most expensive)
Modelos de implementação	
On-site	Sim
Nuvem	Não
Outros	
Instalação modular	Sim
Geração de avisos necessários	Não
Capacidade de geração de relatórios	HTML, XML, JSON e CSV

	Dependency-track
Base de conhecimentos	
Tamanho	O dependency-track não se baseia numa base de conhecimentos própria, ele depende de ferramentas externas para gerar o SBOM (software bill of materials). O SBOM tem de estar no formato da CycloneDX. Como exemplo, tens aqui um conjunto de ferramentas que geram esse SBOM (https://cyclonedx.org/tool-center/). De realçar que há ferramentas que geram SBOM no formato SPDX, e depois é possível converter (usando outra ferramenta à parte) para CycloneDX.
Frequência de atualização	Não aplicável
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Não
Facilidade de utilização	
Intuição	4/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	4/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	Não aplicável (não faz scan ao código)
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim (Licença é a Apache License 2.0)
Dar suporte a diferentes modelos de auditoria	Traditional apenas

	Dependency-track
Agnóstico a linguagens de programação	Sim, usa SBOM (Licença é a Apache License 2.0)
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	API e CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	Permite ter vulnerabilidades de um largo conjunto de projetos. E permite ter um repositório interno com vulnerabilidades dentro da organização.
Frequência de atualização	Sempre que a ferramenta corre, verifica os data feeds da NVD (https://nvd.nist.gov/vuln/data-feeds), o que a permite descobrir vulnerabilidades muito recentes
Fontes da informação	NVD NPM Audit API, the OSS Index, RetireJS, Bundler Audit
Capacidade de priorizar vulnerabilidades	Sim
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Não
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Não
Custos associados	
infraestrutura	2/5 (1=Cheapest;5=Most expensive)
operacional	1/5 (1=Cheapest;5=Most expensive)
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	1/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	1/5 (1=Cheapest;5=Most expensive)

	Dependency-track
Modelos de implementação	
On-site	Sim
Nuvem	Não
Outros	
Instalação modular	Sim
Geração de avisos necessários	Sim
Capacidade de geração de relatórios	O dependency-track não gera reports, as métricas ficam todas embebidas na plataforma. Contudo, como tem uma API simples e intuitiva, é muito fácil criar um script que nos gere reports com o formato que quisermos.

	Gitlab
Base de conhecimentos	
Tamanho	GitLab analyzers obtain dependency information using one of the following two methods: Parsing lockfiles directly. Running a package manager or build tool to generate a dependency information file which is then parsed. The following package managers use lockfiles that GitLab analyzers are capable of parsing directly: Bundler, Composer, Conan, Go, NuGet, npm, yarn, Poetry Obtaining dependency information by running a package manager to generate a parsable file To support the following package managers, the GitLab analyzers proceed in two steps: Execute the package manager or a specific task, to export the dependency information. Parse the exported dependency information. Package Managers: sbt, Maven, Gradle, setuptools, pip e Pipenv
Frequência de atualização	This advisory database is constantly being updated
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Penso que não
Facilidade de utilização	
Intuição	5/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	5/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	Não encontrei esta informação

	Gitlab
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim (Licença é a Apache License 2.0)
Dar suporte a diferentes modelos de auditoria	Dá suporte ao Tradicional e DIY. Os reports de auditoria são altamente personalizáveis
Agnóstico a linguagens de programação	Não
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	API e CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	Permite ter vulnerabilidades de um largo conjunto de projetos.
Frequência de atualização	É atualizada diariamente
Fontes da informação	NVD, ruby-advisory-db, GitHub Advisory Database
Capacidade de priorizar vulnerabilidades	Sim
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Sim. "DS_REMEDIATE Enable automatic remediation of vulnerable dependencies." "Solutions for vulnerabilities (auto-remediation) | In Free - No | In Ultimate - Yes"
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim (não encontrei este tipo de informação)
Custos associados	
infraestrutura	3/5 (1=Cheapest;5=Most expensive) Custo do Ultimate 99$/mês, 1 188$/ano. É o preço do uso do Gitlab com acesso ao Dependency Scanning
operacional	3/5 (1=Cheapest;5=Most expensive) Não achei quase nenhuma review a falar de falsos positivos, e têm

	Gitlab
	um sistema de reportar falsos positivos. Também o utilizador pode marcar como falso positivo.
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	2/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	1/5 (1=Cheapest;5=Most expensive)
Modelos de implementação	
On-site	Sim
Nuvem	Não
Outros	
Instalação modular	Acho que sim
Geração de avisos necessários	Sim, quando se faz merge para o GitLab podemos ativar a opção de fazer o scan automaticamente
Capacidade de geração de relatórios	JSON In addition to the JSON report file, the Gemnasium Dependency Scanning tool outputs a CycloneDX Software Bill of Materials (SBOM) for each supported lock or build file it detects.

	Github
Base de conhecimentos	
Tamanho	Supported package ecosystems (supply chain security): Composer, NuGet, Go modules, Maven, npm, pip, RubyGems e Yarn. GitHub automatically enables Dependabot security updates for every repository that meets these prerequisites. 1. Automatic enablement prerequisite; 2. Repository is not a fork; 3. Repository is not archived; 4. Repository is public, or repository is private and you have enabled read-only analysis by GitHub, dependency graph, and vulnerability alerts in the repository's settings; 5. Repository contains dependency manifest file from a package ecosystem that GitHub supports; 6. Dependabot security updates are not disabled for the repository
Frequência de atualização	You specify how often to check each ecosystem for new versions in the configuration file: daily, weekly, or monthly.
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Não
Facilidade de utilização	
Intuição	4/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)

	Github
Requer o mínimo de treino possível	4/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	Code scanning uses Github Actions, and each run of a code scanning a workflow consumes minutes for Github Actions.
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim (Licença é a Apache License 2.0)
Dar suporte a diferentes modelos de auditoria	Traditional
Agnóstico a linguagens de programação	Não (C#, Go, Java, JavaScript, PHP, Python, Ruby, Scala, TypeScript)
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	Só permite ter vulnerabilidade do projeto a ser analisado, mas pode ser facilmente configurado para analisar múltiplos projetos.
Frequência de atualização	"You specify how often check each ecosystem for new versions in the configuration file: daily, weekly or monthly" Não encontrei mais nenhuma informação para além disto sobre a atualização da base de dados de vulnerabilidade.
Fontes da informação	- NVD; - A combination of machine learning and human review to detect vulnerabilities in public commits on GitHub; - Security Advisories reported on GitHub (GitHub Advisory Database); - The npm Security advisories database
Capacidade de priorizar vulnerabilidades	Sim
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Sim. "When Dependabot raises pull requests, these pull requests could be for *security* or *version* updates: *Dependabot security updates* are automated pull requests that help you update dependencies with known vulnerabilities. *Dependabot version updates* are automated pull requests that keep your dependencies updated, even when they don't have any vulnerabilities. To check the status of version updates, navigate to

	Github
	the Insights tab of your repository, then Dependency Graph, and Dependabot.”
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim (não encontrei este tipo de informação)
Custos associados	
infraestrutura	3/5 (1=Cheapest;5=Most expensive)
operacional	5/5 (1=Cheapest;5=Most expensive)
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	2/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	3/5 (1=Cheapest;5=Most expensive) - Muita personalização, mas tem de ser o cliente a configurar
Modelos de implementação	
On-site	Não
Nuvem	Sim
Outros	
Instalação modular	Acho que sim (Go Modules)
Geração de avisos necessários	Sim
Capacidade de geração de relatórios	Não achei nada em relação a isto

	WhiteSource Software
Base de conhecimentos	
Tamanho	3 Milhões de componentes *open-source* With more than 200 different open source licenses Possui uma **SBOM** produzida pela Mend SCA (Identify all open source libraries). "Gain complete coverage over your open source use with the largest vulnerability database in the industry. With more than 270 million open source components and 13 billion files”
Frequência de atualização	Não encontrei informação. Penso que deve ser atualizada regularmente como a base de dados de segurança.
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade	Sim.

	WhiteSource Software
verificando se o componente vulnerável é utilizado	"Mend offers Prioritize, a feature powered by the Effective Usage Analysis (EUA) technology, which reveals if reported vulnerabilities are effectively being referenced (directly or indirectly) from proprietary code, providing the relevant file name and line number whence the proprietary call originates. Equipped with such information, developers can consider multiple remediation approaches, including commenting out code, bypassing the code that calls the vulnerability and more. "
Facilidade de utilização	
Intuição	4/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	4/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	Poucos segundos
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Provavelmente sim. “With more than 200 different open-source licenses.” É possível adicionar licenças com o respetivo ficheiro.
Dar suporte a diferentes modelos de auditoria	Tradicional
Agnóstico a linguagens de programação	+ de 200 linguagens de programação e possui SBOM
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	API e CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	Mais de 300 000 componentes vulneráveis. Permite ter vulnerabilidades de um largo conjunto de projetos.
Frequência de atualização	Várias vezes ao dia.
Fontes da informação	CVE/NVD, Github Issue Tracker, Security advisories e projetos *open-source* issues trackers. Our vulnerability database continuously monitors multiple resources including the NVD and a wide range of security advisories and issue trackers. **CVE Numbering Authority** – Mend is a CVE Numbering Authority, which allows us to responsibly disclose new security vulnerabilities found through our own research.
Capacidade de priorizar vulnerabilidades	Sim

	WhiteSource Software
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Sim
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Acho que não (não encontrei este tipo de informação)
Custos associados	
infraestrutura	5/5 (1=Cheapest;5=Most expensive) Starting Price for Teams: $12 000/year for 20 developers (minimum) Highest Price for Teams: $300 000/year >500 developers (maximum) Starting Price for Enterprise: $32 000/year for 40 developers (minimum) Highest Price for Enterprise: $400 000/year >500 developers (maximum)
operacional	2/5 (1=Cheapest;5=Most expensive) Muito poucos falsos positivos.
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	3/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	1/5 (1=Cheapest;5=Most expensive)
Modelos de implementação	
On-site	Acho que sim
Nuvem	Sim
Outros	
Instalação modular	Sim (Go modules)
Geração de avisos necessários	Sim
Capacidade de geração de relatórios	Sim. "We offer a variety of reports that will help you monitor all of your open-source activity such as an Inventory report, due diligence

	WhiteSource Software
	report, high severity bugs report and vulnerability report and many more." Most reports can be exported to Excel and XML (some reports also support export to HTML or JSON formats). Exported reports reflect the selected scope (Organization, Product, or Project) and the defined filters.

	CAST Highlight
Base de conhecimentos	
Tamanho	SBOM data can be automatically exported in various formats including industry standards and flexible options such as CycloneDX, Word, Excel, PPT, XML, and REST API. Automatically detect all open-source frameworks and 3rd party components from a proprietary knowledgebase of 100 million+ components. CAST consolidates a unique database made of 94M+ Open-Source components and 9B+ file fingerprints. To constitute our knowledge base on Open Source, we continuously crawl the different platforms, whether it is for source code (currently supported: Github, GitLab, Salsa Debian, Framagit) or packages/binaries (currently supported: Maven, NPM, NuGet, PyPi, Packagist, RubyGem…). Currently supported dependency management tools & files: Ant (build.xml) Bauer (bauer.json) Composer (composer.json) Go (Go.mod, Go.sum) Gradle (build.gradle, dependencies.gradle) Maven (pom.xml) NPM (package.json and package-lock.json) Python (requirements.txt, setup.py) Ruby (Gemfile.lock) Visual Studio (.vcproj, .csproj) Yarn (yarn.lock) SBOM data can be automatically exported in various formats including industry standards and flexible options such as CycloneDX, Word, Excel, PPT, XML, and REST API. CAST Highlight detects 350+ Open Source licences
Frequência de atualização	Atualizada diariamente
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade verificando se o	Sim. Pode-se adicionar exclusões e a razão delas.

	CAST Highlight
componente vulnerável é utilizado	Exclusions are regularly extracted in an anonymized list by the product team to proactively improve our CVE matching algorithm and remove possible false positives from all CAST Highlight instances.
Facilidade de utilização	
Intuição	5/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	4/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	It takes less than 5 minutes to analyze a normal-sized application of 150,000 lines of code (LOC)
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim (Licença é a Apache License 2.0)
Dar suporte a diferentes modelos de auditoria	Não encontrei informação específica sobre isto. A app apresenta um assessment. Traditional
Agnóstico a linguagens de programação	46 linguagens de programação
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	API e CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	across 150+ thousand known vulnerabilities Permite ter vulnerabilidades de um largo conjunto de projetos.
Frequência de atualização	Não achei esta informação
Fontes da informação	Depending on the component name and version detected in an application scan, CAST Highlight finds possible vulnerabilities (CVEs) by cross-referencing the National Vulnerablity Database (NVD) from NIST, across 150+ thousand known vulnerabilities. Users typically run a first analysis of their application to establish a CVE baseline and prioritize risk mitigation actions.
Capacidade de priorizar vulnerabilidades	Sim
Remediação automatizada (Exp.: Cria pull requests que	Não

	CAST Highlight
permitem aos programadores atualizar o package recomendado com um único clique)	
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Acho que não (não encontrei este tipo de informação)
Custos associados	
infraestrutura	4/5 (1=Cheapest;5=Most expensive) $26k - $109K Any # of developers/Per year
operacional	3/5 (1=Cheapest;5=Most expensive) Exclusions are regularly extracted in an anonymized list by the product team to proactively improve our CVE matching algorithm and remove possible false positives from all CAST Highlight instances.
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	3/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	2/5 (1=Cheapest;5=Most expensive)
Modelos de implementação	
On-site	Acho que não
Nuvem	Sim
Outros	
Instalação modular	Sim (Go modules)
Geração de avisos necessários	Sim
Capacidade de geração de relatórios	Sim em CSVs.

	Contrast Security
Base de conhecimentos	
Tamanho	Export library versioning, vulnerability, licensing and environment data to a standardized Software Bill of Materials (SBOM)
Frequência de atualização	Não aplicável
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade	Sim

	Contrast Security
verificando se o componente vulnerável é utilizado	"Prioritize remediation efforts by accurately identifying whether vulnerable open-source libraries are actually used by the application- all the way down to the specific class, file or module"
Facilidade de utilização	
Intuição	5/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	4/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	With CodeSec developers can scan code and serverless environments in order to secure their code in **under five** minutes. "The most valuable feature is the continuous monitoring aspect: the fact that we don't have to wait for scans to complete for the tool to identify vulnerabilities. They're automatically identified through developers' business-as-usual processes."
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim "**Investors.** SBOMs can help provide documentation of critical software assets for mergers and acquisitions (M&A), venture capital fundraising, and initial public offerings (IPOs)—helping investors avoid the oft-repeated tale of "buying a breach.""
Dar suporte a diferentes modelos de auditoria	Sim, Traditional SOC2 Type II Report
Agnóstico a linguagens de programação	11 Supported languages: Java, Javascript, .NET Framework, .NET Core, Node.js, Python, Ruby, Go, PHP, Scala e Kotlin "Contrast agents are responsible for gathering security relevant data from an application, analyzing that data, and reporting findings to Contrast when necessary. In specific situations, a Contrast agent can also take actions within an application to prevent exploitation or enable a security defense. A Contrast agent gathers security relevant information using a variety of security instrumentation techniques, including code scanning, library scanning, instrumenting an application, configuration file scanning, and other techniques. Any security instrumentation technique that gathers information is a sensor."
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	CLI e API "Integrate the Contrast CLI into native CI/CD processes to populate the dependency tree and highlight potential risk"
Base de dados de vulnerabilidades de segurança	
Tamanho	"**Question: How robust is Contrast? What is the maximum number of applications that can be simultaneously monitored?** Answer: There is no limit. Any number of applications can be analyzed simultaneously, continuously, and in real-time."

	Contrast Security
Frequência de atualização	OWASP continuously monitor sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components. Use software composition analysis tools to automate the process Como a OWASP atualiza-se pela CVE e NVD estes são atualizados **diariamente**.
Fontes da informação	"**Question: How does Contrast Security stay on top of their research capabilities for new and emerging vulnerabilities?** **Answer:** Contrast founders and employees are the founders of OWASP and are the driving force behind many of the most successful OWASP projects, including the OWASP Top 10, ESAPI, WebGoat, XSS Prevention Cheat Sheet, and others. All of Contrast's employees spend some time on security research, which we share openly through OWASP. In addition, Contrast has partnered with Aspect Security, a leading application security company, to share research/consulting knowledge. Aspect has several dozen researchers/consultants analyzing hundreds of millions of lines of code and discovers thousands of vulnerabilities every year for the largest and most important financial, government and commercial institutions around the globe. See Aspect's 2013 Global Application Security Risk Report for details on this work. Through this relationship, Contrast has access to the real vulnerabilities in real applications that drive useful new rules."
Capacidade de priorizar vulnerabilidades	Sim
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Não
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim
Custos associados	
infraestrutura	3/5 (1=Cheapest;5=Most expensive) "When comparing Contrast Security to alternative systems, **on a scale between 1 to 10 Contrast Security is rated 6.2**"

	Contrast Security
operacional	3/5 (1=Cheapest;5=Most expensive) “Contrast platform automatically detects vulnerabilities while developers write code, eliminates false positives, and provides context-specific how-to-fix guidance for easy and fast vulnerability remediation.”
licenciamento	3/5 (1=Cheapest;5=Most expensive) Ronda os $10 000 anuais
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	1/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	5/5 (1=Cheapest;5=Most expensive) “Minimal customization - integrate with 1-2 systems: $2,500 Standard customization - integrate with 3-5 systems: $10,000 Fully customized system - integrate with more than 5 systems: $25,000"
Modelos de implementação	
On-site	Sim
Nuvem	Sim. Plugin
Outros	
Instalação modular	Acho que não
Geração de avisos necessários	Sim
Capacidade de geração de relatórios	PDF

	Snyk
Base de conhecimentos	
Tamanho	The following package managers are supported: deb gomodules gradle maven nuget paket pip rpm rubygems cocoapods npm yarn Package manager/build tool .NET (C#, F#, Visual Basic)/ Nuget, Paket Bazel / See API docs. C / C++ / N/A Elixir / hex Go / Go Modules, dep, govendor

	Snyk
	Java / Gradle, Maven JavaScript / npm, yarn PHP / Composer Python / pip, Poetry, pipenv Ruby / Bundler Scala / sbt Swift and Objective-C / CocoaPods
Frequência de atualização	Não achei esta informação
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade verificando se o componente vulnerável é utilizado	Sim Even if there is a vulnerable library that is in use, but maybe we not using the function itself, it's not telling us that we do use that function.
Facilidade de utilização	
Intuição	5/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	5/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	Não encontrei esta informação
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim (Possui a licença a Apache License 2.0)
Dar suporte a diferentes modelos de auditoria	Blind Audit -Audits of cryptographic hashes - source codes never accessed or transferred -Ridiculously easy to set up and administer with fewer headaches for Legal. -Performed entirely remotely
Agnóstico a linguagens de programação	15 linguagens de programação
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	CLI e API
Base de dados de vulnerabilidades de segurança	
Tamanho	Permite ter vulnerabilidades de um largo conjunto de projetos.
Frequência de atualização	Atualizada 2 a 3x por dia

	Snyk
Fontes da informação	Most of the vulnerabilities in our database originate from one of these sources: Monitoring other vulnerability databases, such as CVEs from **NVD** and many others. Monitoring user activity on GitHub, including issues, PRs and commit messages that may indicate a vulnerability. Bulk research, using tools that look for repeated security mistakes across open-source package code Manual research, investing our researchers time to manually audit more widely used packages for security flaws.
Capacidade de priorizar vulnerabilidades	Sim
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Sim
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Acho que não (não encontrei este tipo de informação)
Custos associados	
infraestrutura	4/5 (1=Cheapest;5=Most expensive) Free unlimited developers, limited tests $5 850 /year for 5 developers - $29 250 /year 25 developers (For Teams) $10 625 /year for 5 developers - $124 500 /year 75 developers (For Business) Enterprise Custom (Schedule a Demo)
operacional	2/5 (1=Cheapest;5=Most expensive) “Snyk only validates vulnerabilities that exist in dependent components, so it has a relatively low false-positive rate”
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	3/5 (1=Cheapest;5=Most expensive)
personalização de engenharia	5/5 (1=Cheapest;5=Most expensive) Possui uma versão personalizada paga
Modelos de implementação	

	Snyk
On-site	Não
Nuvem	Sim. Plugin
Outros	
Instalação modular	Sim (Go modules)
Geração de avisos necessários	Sim
Capacidade de geração de relatórios	PDFs

	BlackDuck Software
Base de conhecimentos	
Tamanho	-Black Duck's multifactor open-source detection and KnowledgeBase™ of over **4 million** components gives you an accurate Bill of Materials (BoM) for any application or container. - mais de 2750 licenças open-source -covering more than 5.1 million open-source components from over 26,000 forges and repositories. -Mais de 5 100 000 open-source projects -4.5 million unique open-source projects Package managers: Bazel Cargo C/C++ (Clang) Conan Conda Dart Docker image GoLang Gradle Erlang/Hex/Rebar Lerna Maven npm NuGet Python The Pip detector sbt Background execution (The sbt command line) Yarn Yocto (BitBake)
Frequência de atualização	The Black Duck KnowledgeBase receives **hourly** vulnerability and project data updates on the latest security, license, and quality risks affecting open source in the codebase.
Capacidades de deteção	
Capacidade de detetar uma vulnerabilidade	Informação não disponibilizada. Penso que não.

	BlackDuck Software
verificando se o componente vulnerável é utilizado	
Facilidade de utilização	
Intuição	4/5 (1=Muito pouco intuitivo; 5=Muito intuitivo)
Requer o mínimo de treino possível	4/5 (1=Muito treino; 5=Pouco treino)
Capacidades operacionais	
Velocidade de scans	Menos de 1 minuto. "**Rapid Scan** instantly analyzes open source dependencies for vulnerabilities and policy violations before code is built or merged into release branches. " "Completing a dependency analysis in under a minute, it can complete more than 30,000 scans per day. "
Habilidade para usar scans M&A (Mergers and Acqusitions) - sem bloqueio de licença em modelos de utilização	Sim (Possui a licença a Apache License 2.0 e MIT)
Dar suporte a diferentes modelos de auditoria	Traditional
Agnóstico a linguagens de programação	More than 90 programming languages BlackDuck's proprietary codeprint analysis is language agnostic. This scanning approach searches for signatures based on file and directory layouts along with other metadata that is independent of language.
Capacidades de integração	
Integração com sistemas de construção através de APIs e CLI (Command Line Interface)	API e CLI
Base de dados de vulnerabilidades de segurança	
Tamanho	Mais de 174 000 vulnerabilidades únicas e permite ter múltiplos projetos.
Frequência de atualização	Diariamente
Fontes da informação	NVD, Black Duck Security Advisories (BDSAs) data that is researched and analyzed by the Synopsys Cybersecurity Research Center (CyRC)
Capacidade de priorizar vulnerabilidades	Sim

	BlackDuck Software
Remediação automatizada (Exp.: Cria pull requests que permitem aos programadores atualizar o package recomendado com um único clique)	Acho que sim "Use the REST API to accelerate and automate essential risk mitigation and remediation tasks" "Auto-remediation"
Métodos de descoberta avançados	
Apoio à descoberta avançada de vulnerabilidades (identificação de uma vulnerabilidade quando o código vulnerável é copiado para um novo componente)	Sim
Custos associados	
infraestrutura	3/5 (1=Cheapest;5=Most expensive) $500 per team member (20-150 team members) - Security Edition ($10 000 até $75 000) Contact company - Professional Edition
operacional	2/5 (1=Cheapest;5=Most expensive) "reports with minimal false positives"
licenciamento	No costs
integração	1/5 (1=Cheapest;5=Most expensive)
lock-in	3/5 (1=Cheapest;5=Most expensive) Possui uma base de dados própria
personalização de engenharia	4/5 (1=Cheapest;5=Most expensive) Uma empresa tem de contactar a empresa para obter um plano de uso da ferramenta. Logo, deve haver alguma personalização de acordo com a empresa
Modelos de implementação	
On-site	Sim
Nuvem	Sim
Outros	
Instalação modular	Sim (Go modules)
Geração de avisos necessários	Sim
Capacidade de geração de relatórios	"You can use this report to create an attribution report for your project release or to share BOM and license information. This report is available as a **text** file or in **HTML** format." "Going forward, all reports will be Excel or PDF with no fancy embedded scripting, so you can easily share them with colleagues or advisors via email or post them to a virtual data room. We will

	BlackDuck Software
	distribute reports via Citrix ShareFile, the highly secure, industry-standard portal.”

Apêndice C

Este apêndice inclui uma tabela de apoio para o uso do AHP.
URL para o apêndice C: https://1drv.ms/x/s!AjeuwDsT4Y95lzd06JjzyMOx5j5j?e=YqfEkl