UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JERÔNIMO JAIRO SILVA DE ARAÚJO

ANALISANDO VULNERABILIDADES DE SEGURANÇA EM UM PROJETO DE SOFTWARE

CAMPINA GRANDE - PB

2023

JERÔNIMO JAIRO SILVA DE ARAÚJO

ANALISANDO VULNERABILIDADES DE SEGURANÇA EM UM PROJETO DE SOFTWARE

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Professor Everton Leandro Galdino Alves

CAMPINA GRANDE - PB

2023

JERÔNIMO JAIRO SILVA DE ARAÚJO

ANALISANDO VULNERABILIDADES DE SEGURANÇA EM UM PROJETO DE SOFTWARE

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Everton Leandro Galdino Alves

Orientador – UASC/CEEI/UFCG

Professor Franklin de Souza Ramalho

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 28 de Junho de 2023.

CAMPINA GRANDE - PB

ABSTRACT

Security vulnerabilities in computer systems are often complex problems to deal with, and even with improvements in the development process they tend to persist. The lack of interest in removing these vulnerabilities during development can become a setback in the future (technical debt), result in improper access, expose user data, and revert in financial costs to the company. Thus, it is necessary that concerns with the identification and removal of these vulnerabilities exist during the entire software development process. In this work, static analysis tools (Check Marx, Black Duck and Jfrog Xray) will be used to analyze the evolution, distribution by risk class and lifetime of security vulnerabilities in a project developed by a large company in partnership with the Distributed Systems Lab. The project in question consists of a service that offers management for observability resources. From the results it was found that vulnerabilities affecting open-source components found by the Black Duck and Jfrog Xray tools were being addressed. However, security vulnerabilities affecting project code points found in the Check Marx tool were not being addressed.

Analisando Vulnerabilidades de Segurança em um Projeto de Software

Jerônimo Jairo Silva de Araújo
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
jeronimo.araujo@ccc.ufcg.edu.br

Everton Leandro Galdino Alves
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
everton@computacao.ufcg.edu.br

RESUMO

Vulnerabilidades de segurança em sistemas computacionais são problemas geralmente complexos de tratar e que mesmo com a melhora no processo de desenvolvimento tendem a persistir. A falta de interesse na remoção dessas vulnerabilidades durante o desenvolvimento pode se tornar um contratempo no futuro (débito técnico), resultar em acessos indevidos, expor dados dos usuários e reverter em custos financeiros para a empresa. Desta forma, faz-se necessário que preocupações com a identificação e remoção dessas vulnerabilidades existam durante todo o processo de desenvolvimento do software. Neste trabalho, serão utilizadas ferramentas de análise estática (Check Marx, Black Duck e Jfrog Xray) para analisar a evolução, distribuição por classe de risco e tempo de vida de vulnerabilidades de segurança em um projeto desenvolvido por uma empresa de grande porte em parceria com o Laboratório de Sistemas Distribuídos. O projeto em questão consiste de um serviço que oferece gerenciamento para recursos de observabilidade. A partir dos resultados obtidos foi constatado que vulnerabilidades que afetam componentes open-source encontradas pelas ferramentas Black Duck e Jfrog Xray estavam sendo tratadas. No entanto, as vulnerabilidades de segurança que afetam pontos de código do projeto encontrados na ferramenta Check Marx, não estavam sendo abordadas.

Palavras-chave

Vulnerabilidades, segurança, SAST, SCA, desenvolvimento.

1. INTRODUÇÃO

Com o mercado cada vez mais competitivo e exigindo uma maior velocidade para entrega de produtos. Um processo de desenvolvimento que consiga se adequar a esse cenário é de suma importância [1]. Por essa razão, práticas de desenvolvimento ágil foram introduzidas, incorporando valores e princípios que devem ser seguidos, a fim de entregar cada vez mais rápido o produto, priorizando a satisfação do cliente [1].

Entretanto, devido a pressão nos desenvolvedores e a falta de ferramentas e boas práticas voltadas a segurança para aplicações, vulnerabilidades acabam sendo ignoradas e descuidadas [2].

Existem muitas definições para o conceito de vulnerabilidade, cobrindo várias concepções, incluindo "conhecimento, ataque, risco, intenção, ameaça, escopo e tempo" [3]. Contudo, nesta pesquisa consideramos uma vulnerabilidade como sendo uma imperfeição que permite que usuários violem as políticas de segurança de um sistema [4].

Dessa forma, as vulnerabilidades podem ser observadas como um problema antigo de sistemas de software, sendo, geralmente, bem complexas de tratar. Por essa razão, a descoberta e a resolução destas, deve ser feita de forma rápida e com precisão [2], pois quando não tratados podem deixar os sistemas suscetíveis a invasões, quebra de integridade de informações, falta de autenticidade e confiabilidade, gerando muitos contratempos para os usuários e custos financeiros para a empresa [5].

Para isso, algumas ferramentas foram criadas para analisar e detectar estaticamente possíveis vulnerabilidades de segurança em um software. Dessas, as ferramentas de Teste de Segurança em Aplicação Estática (SAST) [9] que realizam uma análise estática do código, em busca de vulnerabilidades conhecidas, como SQL Injections, Autorização imprópria, Injeção de código, entre outras. Um exemplo dessa categoria é o Check Marx [6], que detecta vulnerabilidades listadas no OWASP Top 10 [21] e no CWE/SANS Top 25 [22]. Além disso, existem as ferramentas de Análise de Composição de Software (SCA) [10], que têm como foco a localização de componentes open-source, analisando-os e mapeando vulnerabilidades conhecidas, como aquelas listadas no National Vulnerability Database (NVD) [20]. Como exemplos, temos as ferramentas Black Duck [7] e Jfrog Xray [8].

Quando essas ferramentas são integradas no processo de desenvolvimento, as vulnerabilidades passam a ser monitoradas, deixando o time de desenvolvimento ciente das suas existências.

Para analisar como as vulnerabilidades são tratadas em um contexto real, realizamos um estudo de caso. Esse estudo toma como base o projeto Observability Hub. Trata-se de uma parceria entre o Laboratório de Sistemas Distribuídos com uma empresa de grande porte.

Em determinado momento, o projeto incorporou as ferramentas citadas acima em sua pipeline, sendo possível realizar uma análise do código e componentes open-source, e utilizar relatórios e artefatos gerados por estas a fim de evoluir e incorporar políticas de segurança.

No processo de desenvolvimento desse projeto foi incorporado algumas ferramentas de análise estática que tem foco nas vulnerabilidades e que fazem o monitoramento delas. Entretanto, mesmo que os desenvolvedores estejam cientes da sua existência, não existe um processo ou atividade que foque em resolver essas vulnerabilidades.

2. FUNDAMENTAÇÃO TEÓRICA

Esta seção tem como objetivo descrever brevemente acerca de desenvolvimento ágil de software, análise estática e ferramentas de segurança SCA e SAST, a fim de introduzir os conceitos usados neste trabalho.

2.1 Desenvolvimento ágil de software

Os métodos de desenvolvimento ágil de software utilizam processos de desenvolvimento para produzir softwares úteis de forma ágil. Ao contrário dos modelos clássicos, o software não é

desenvolvido completamente como uma única unidade, mas como uma série de incrementos [16].

Os processos de especificação, projeto e implementação que estão presentes no desenvolvimento ágil [16], são intercalados de forma que se reduza a rigidez, característica dos modelos clássicos.

Dentro do processo de implementação, outras atividades são incorporadas, como as atividades de verificação e validação. Elas são atividades essenciais para garantir a segurança e qualidade do software.

No desenvolvimento ágil, a utilização de frameworks de testes automatizados é amplamente utilizada, e é uma prática recomendada [16]. A integração contínua é um processo comumente utilizado durante a implementação, no qual os desenvolvedores integram novos trechos de código com frequência e com pouco atraso na base de código [17]. Durante a integração contínua, são aplicadas etapas de teste automatizados e análise estática no novo código que está sendo integrado.

2.2 Análise estática

Análise estática de código consiste em inspecionar o código-fonte de um software em busca de erros e vulnerabilidades comuns [18], sem a necessidade de haver execução do código.

Existem diversas ferramentas de análise estática de código que tem o objetivo de identificar problemas no código-fonte do software. Algumas dessas ferramentas têm o foco em aspectos de segurança, buscando por falhas de segurança no código ou identificando componentes open-source que têm vulnerabilidades conhecidas.

2.2.1 Security Application Static Testing (SAST)

A metodologia de teste conhecida como Teste de Segurança de Aplicação Estática (SAST) envolve a análise do código-fonte do software com o objetivo de encontrar vulnerabilidades de segurança conhecidas que possam deixar a aplicação suscetível a ataques. [19]

As ferramentas SAST são comumente integradas no início do ciclo de desenvolvimento, pois não requerem a execução da aplicação, pois a análise é feita no código fonte. Além disso, essas ferramentas oferecem um apoio e ajuda para que os desenvolvedores consigam resolver os problemas encontrados [19].

2.2.2 Software Composition Analysis (SCA)

O processo automatizado conhecido como Análise de Composição de Software (SCA) tem como objetivo identificar os componentes open-source presentes no código-fonte do software [20]. Essa identificação é realizada inspecionando os arquivos de gerenciamento de pacotes, arquivos de manifesto, arquivos binários, imagens de containers, código-fonte e mais.

Os componentes open-source identificados são adicionados no Bill of Materials (BOM). Em seguida, os componentes guardados no BOM são analisados, comparando-os com uma variedade de bancos de dados que mapeiam vulnerabilidades para componentes open-source, incluindo à National Vulnerability Database (NVD) [20]. Assim, caso alguma vulnerabilidade seja detectada durante a análise, ela é mapeada para o componente.

3. METODOLOGIA

Esta pesquisa utilizou como objeto de estudo o projeto Observability Hub, que incorporou as ferramentas de análise estática segurança Black Duck [7], Check Marx [6] e Jfrog Xray [8] na sua pipeline de integração contínua no final de novembro de 2021. Elas foram adicionadas para seguir uma norma de controle de qualidade e segurança nos projetos definida pela empresa, e tem como essas ferramentas o padrão para segurança nos projetos.

O projeto trata-se de uma parceria entre o Laboratório de Sistemas Distribuídos e uma empresa de grande porte e tem por objetivo desenvolver o projeto Observability Hub.

Ao todo, a equipe do projeto é composta por dez pessoas do Laboratório de Sistemas Distribuídos, e cinco pessoas da empresa. A equipe utiliza a metodologia de desenvolvimento ágil Scrum e implementa atividades para garantia de qualidade: métrica de cobertura de testes em, no mínimo, 75% das linhas, revisão de código antes de merge request, reunião de retrospectiva para coleta de aspectos bons e ruins da Sprint, ambientes separados para desenvolvimento, validação e produção. Atualmente, o projeto conta com aproximadamente 83.669 linhas de código (LOC) e possui 272 dependências com componentes open-source, considerando também dependências transitivas.

Durante o período de novembro de 2021 até 27 de maio de 2023, as ferramentas de análise estática foram executadas para cada commit criado, como parte da pipeline de integração contínua do projeto, mas versões anteriores a essa data não foram escaneadas e por isso, não sabemos como estava a segurança do projeto naquela época.

Para isso, considerando que o projeto iniciou em agosto de 2021, foi determinado um intervalo da data do início do projeto até o dia 17 de maio de 2023, onde seriam criadas versões mensais do projeto para serem analisadas pelas ferramentas.

Com isso, foram geradas informações por essas ferramentas que foram coletadas e com elas queremos analisar as vulnerabilidades presentes no projeto. Para isso, algumas perguntas foram elaboradas e estas direcionarão essa pesquisa:

1. Como se deu a evolução das vulnerabilidades ao longo do tempo?
2. Qual o tempo de vida médio das vulnerabilidades no projeto?
3. Quais foram as classes de risco das vulnerabilidades encontradas?
4. Existe alguma correlação entre o tempo de vida das vulnerabilidades e as classes de risco?

3.1 Execução das ferramentas

A integração das ferramentas de segurança foi implementada na pipeline de integração contínua, a fim de permitir a análise da segurança do código e componentes open-source do projeto durante o ciclo de desenvolvimento. Para isso, foi definida uma etapa que era executada em todas as branches e continha tarefas responsáveis por executar as ferramentas no código enviado para o repositório. Cada tarefa foi configurada para executar uma ferramenta específica, juntamente com as regras para determinar se ela falharia ou seria bem-sucedida.

No escopo deste estudo, foi considerado o intervalo de agosto de 2021 a maio de 2023, no qual foram criadas versões do projeto para cada mês desse intervalo. Consequentemente, foram geradas 22 versões que continham o código do projeto referente a cada mês e ano específico.

Foi criada uma branch específica no projeto, junto com a pipeline contendo somente as tarefas que rodavam as ferramentas de análise estática. As versões foram enviadas para essa branch em ordem cronológica, da versão mais antiga para a mais nova, permitindo a execução da pipeline para essas versões, e gerando as informações que serão utilizados na pesquisa.

3.2 Coleta de dados

Para realizar a coleta dos dados utilizados nesta pesquisa, foi feito um estudo nas documentações disponibilizadas pelas ferramentas a fim de encontrar as informações necessárias. Durante esse processo, foi descoberto que todas as três ferramentas dispõem de REST APIs, por meio das quais é possível consultar e coletar as informações necessárias.

O objetivo era encontrar informações sobre as vulnerabilidades que foram identificadas na versão analisada, assim como as suas classes de risco. Especificamente para a ferramenta de Análise Estática de Segurança (Static Application Security Testing - SAST), procuramos identificar quais vulnerabilidades estavam afetando trechos de código do projeto. Já para as ferramentas de Análise de Composição de Software (Software Composition Analysis - SCA), buscamos identificar quais vulnerabilidades estavam afetando componentes open-source do projeto.

Para facilitar a interação com as REST APIs das ferramentas e coleta das informações entregues como resposta, foi utilizada a ferramenta com interface amigável, que permitisse organizar facilmente as informações obtidas. O Postman [15] foi a ferramenta escolhida, e nela foi criado uma coleção[1] contendo todas as requisições que foram criadas para as APIs das ferramentas, como também as respostas obtidas.

Após executar todas as requisições necessárias foram obtidas respostas em formato JSON, que foram armazenadas no mesmo formato na própria ferramenta.

3.3 Pré-processamento

Durante a coleta de dados, foi observado que as respostas obtidas continham informações desnecessárias que não seriam utilizadas na análise exploratória de dados. Dentre essas informações desnecessárias, podemos citar:

- Sobre a vulnerabilidade: descrição detalhada, solução recomendada, links para as páginas das vulnerabilidades, entre outras.
- Sobre o projeto: identificação, nome, link do projeto na ferramenta, proprietário do projeto, entre outras.
- Sobre o scan: identificação e usuário que executou o scan.
- Informações sensíveis: nomes de usuário, nome do projeto, links para as ferramentas dentro da empresa, informações sobre o projeto, entre outras.

Portanto, foi necessário realizar um pré-processamento a fim de selecionar apenas informações relevantes para a análise. Os seguintes dados foram selecionados:

- Data do scan
- Identificação da vulnerabilidade
- Classe de risco
- Nome, versão e gerenciador de dependências do componente open-source afetado (no caso das ferramentas de Análise de Composição de Software - SCA)
- Trecho e classe do código afetado (no caso das ferramentas de Teste de Segurança de Aplicação Estática - SAST)

Com base nesses critérios, foram criados três conjuntos de dados (datasets) correspondentes a cada ferramenta de análise de segurança com as informações necessárias sobre as vulnerabilidades que foram analisadas e descobertas no projeto. Esses conjuntos de dados[2] foram estruturados em formato CSV, pois facilitou a manipulação e a posterior análise.

[1] A coleção pode ser encontrada em: https://documenter.getpostman.com/view/17010614/2s93sf4CJw

[2] Os conjuntos de dados estão disponíveis em: https://drive.google.com/drive/folders/12Nml-TxYen0y5_JOAcNgXAZ9GUJC3NEp?usp=sharing

3.4 Análise dos dados

Após a coleta e pré processamento dos dados obtidos, realizamos uma análise dos dados para responder às perguntas direcionadoras da pesquisa.

Para isso, utilizamos bibliotecas do Python, como Pandas [12] e Numpy [13], para manipulação e análise dos dados. Além disso, utilizamos a ferramenta Seaborn [14] para gerar visualizações das análises feitas.

Para analisar a evolução das vulnerabilidades ao longo do tempo, acumulamos o número total de vulnerabilidades por data da versão e representamos essas informações utilizando um gráfico de linhas. Também investigamos os dez dias que mais tiveram vulnerabilidades registradas, representando essas informações em um gráfico de barras em ordem decrescente do número total de vulnerabilidades.

Para entender o tempo de vida das vulnerabilidades no projeto, calculamos as datas da primeira e última aparição delas. Isso nos permitiu calcular o tempo de vida, em dias, de cada vulnerabilidade no projeto. Utilizamos gráficos boxplot para visualizar a distribuição do tempo de vida das vulnerabilidades.

A fim de compreender as classes de risco presentes no projeto, foram acumuladas o número de vulnerabilidades por dia e por classe de risco, para que pudéssemos visualizar como se deu a evolução das vulnerabilidades por classe de risco. Além disso, visualizamos a quantidade de vulnerabilidades para classe de risco através de um gráfico de barras. Também, analisamos o tempo de vida das vulnerabilidades dessas classes e geramos visualizações de boxplots para cada classe.

Por fim, criamos uma matriz de correlação entre as datas da primeira e última aparição, tempo de vida em dias e a classe de risco das vulnerabilidades no projeto. Essa matriz foi visualizada através de um heatmap, permitindo uma análise das relações entre essas variáveis.

É importante ressaltar que as visualizações citadas anteriormente foram criadas separadamente para cada ferramenta. Assim, foram feitas análises descritivas e exploratórias para cada ferramenta.

4. RESULTADOS

Com base nas análises descritivas e exploratórias das informações obtidas e processadas das ferramentas de análise estática, serão apresentadas as visualizações criadas para ajudar o entendimento das perguntas direcionadoras desta pesquisa.

4.1 Evolução das vulnerabilidades ao longo do tempo

A primeira parte da análise busca compreender a evolução da quantidade de vulnerabilidades entre 17 de agosto de 2021 a 27 de maio de 2023. Para isso, foram criadas visualizações em gráficos de linhas para cada versão do projeto e para cada ferramenta.

Iniciando com as ferramentas de Análise de Composição de Software (SCA) foram examinadas a quantidade total de vulnerabilidades detectadas para componentes open-source por versão para a ferramenta Black Duck (Figura 1). Observou-se que, no início do projeto, a quantidade de vulnerabilidades encontradas para componentes open-source utilizados foi a maior, e ao longo do tempo houve uma tendência de redução, chegando a zero e, posteriormente, apresentando um leve aumento.

Figura 1. Gráfico de linha representando a quantidade de vulnerabilidades detectadas por versão na ferramenta Black Duck.

Para a ferramenta Jfrog Xray (figura 2), foi observado uma curva semelhante a da ferramenta Black Duck, no qual no início do projeto foi identificado uma grande quantidade de vulnerabilidades, e ao longo do tempo houve uma tendência de redução, chegando a zero na última versão analisada.

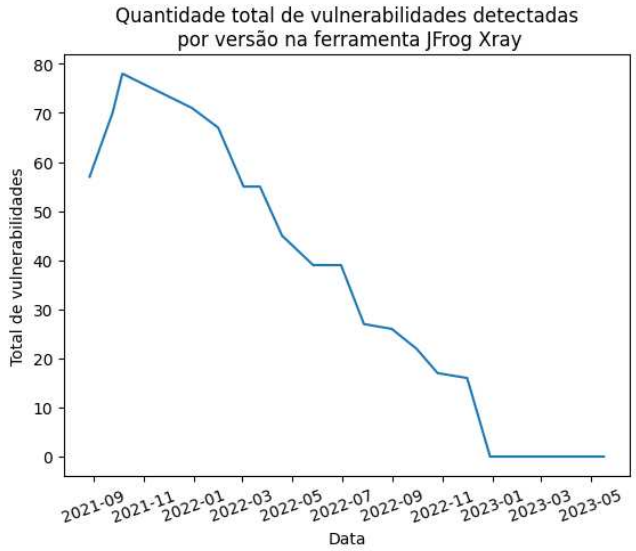

Figura 2. Gráfico de linha representando a quantidade de vulnerabilidades detectadas por versão na ferramenta Jfrog Xray.

Nesta etapa foi analisado o gráfico de linhas gerado para os resultados da ferramenta Check Marx (Figura 3), uma ferramenta de Teste de Segurança de Aplicação Estática (SAST). A quantidade total de pontos de código afetados por vulnerabilidades no início do projeto é menor, e com ao longo do tempo houve tendência de aumento, chegando a maior quantidade registrada nas últimas versões analisadas.

Figura 3. Gráfico de linha representando a quantidade de vulnerabilidades que estão afetando pontos de código detectadas por versão na ferramenta Check Marx.

Diferentemente do que foi diagnosticado para as ferramentas do tipo SCA, a ferramenta do tipo SAST revela que as vulnerabilidades aumentaram com o passar do tempo. Dessa forma, as vulnerabilidades que afetam componentes open-source parecem estar recebendo mais atenção que as detectadas em pontos de código.

Uma possível explicação para a diminuição das vulnerabilidades em componentes open-source, foi a criação, periódica, de atividades para remoção de algumas vulnerabilidades monitoradas. Esse processo se tornou periódico logo após a descoberta de uma vulnerabilidade de altíssimo nível para o componente open-source Log4J [15], em 13 de dezembro de 2021, que trouxe preocupação sobre o monitoramento e resolução de vulnerabilidades em componentes open-source.

Por outro lado, uma vez que não ocorreu nenhum evento relacionado a vulnerabilidades que afetam os pontos de código que despertasse grande preocupação, como ocorreu com as vulnerabilidades em componentes open-source descritas anteriormente, não foi implementada uma atividade integrada nas Sprints para resolver esse tipo de vulnerabilidade.

4.2 Vulnerabilidades por classe de risco

Na segunda parte da análise, buscou-se compreender a distribuição por classe de risco e sua evolução ao longo do tempo, entre 17 de agosto de 2021 a 27 de maio de 2023. Essa análise nos ajudará a evidenciar se houve alguma priorização na resolução das vulnerabilidades.

Começando pela ferramenta Black Duck, inicialmente buscamos entender como se dava a distribuição das vulnerabilidades por classe de risco. Estruturou-se um gráfico de barras (figura 4) que indica uma maior quantidade de vulnerabilidades encontradas pertence a classe de risco “Média”, representando 51.27% das ocorrências. Em seguida temos a classe “Alta” com 27.81%, seguida pela classe "Crítica" com 13.48% e, por fim, a classe “Baixa” com 7.43%. Assim, 41.29% das vulnerabilidades encontradas ao longo do tempo estão concentradas nas classes de risco “Alta” e “Crítica”, que têm um grau de vulnerabilidade preocupante para a aplicação.

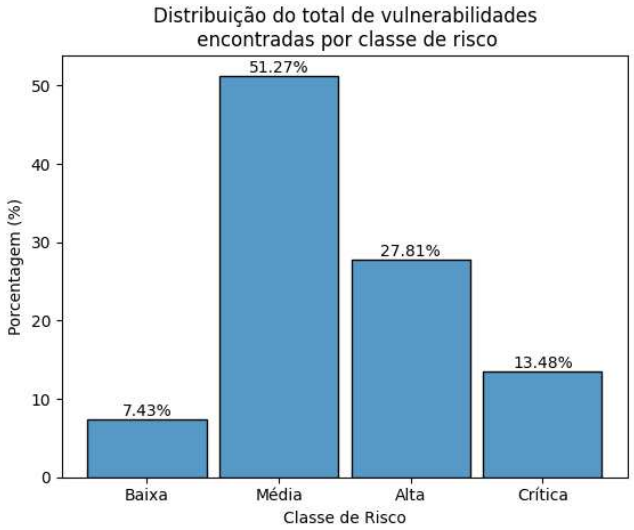

Figura 4. Gráfico de barras representando a distribuição das vulnerabilidades por classe de risco na ferramenta Black Duck.

Em seguida, o gráfico de linhas (figura 5) representa a evolução das vulnerabilidades por classe de risco ao longo do tempo para a ferramenta Black Duck. Observamos os seguintes comportamentos para cada classe de risco:

- Classe “Crítica”: Houve uma tendência de queda seguida de crescimento, seguida mais uma vez de queda nas vulnerabilidades ao longo do tempo.
- Classe “Alta”: Apresentou uma tendência de queda na maior parte do tempo.
- Classe “Média”: No início, apresentou a maior quantidade de vulnerabilidades encontradas, com uma tendência de diminuição, seguida de um aumento que manteve praticamente a mesma quantidade de vulnerabilidades encontradas no início. Posteriormente, houve uma tendência de diminuição para a restante parte do tempo.
- Classe “Baixa”: Inicialmente em baixa quantidade, houve um rápido crescimento seguida de queda e estagnação em aproximadamente zero vulnerabilidades.

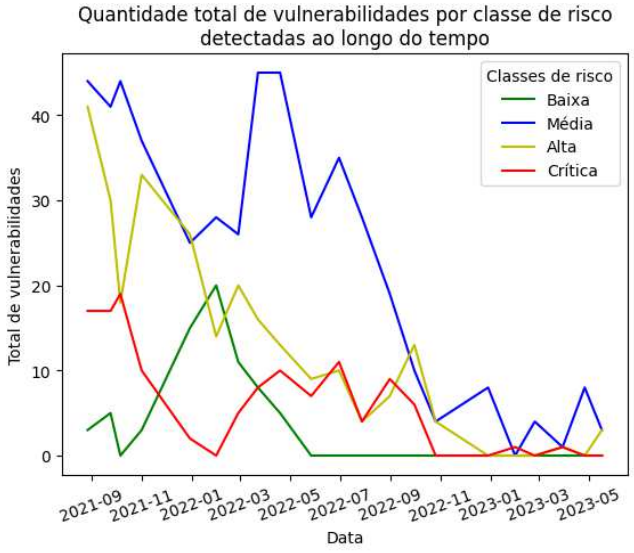

Figura 5. Gráfico de linhas representando a quantidade de vulnerabilidades por classe de risco ao longo do tempo para a ferramenta Black Duck.

Em relação a ferramenta Jfrog Xray, construiu-se o gráfico de barras (figura 6) com a distribuição das vulnerabilidades por classe de risco. Observamos que a maior quantidade de vulnerabilidades encontradas pertencentes a classe "Alta" é de 45,76%, seguida pela classe "Média" com 35,68%, a classe "Crítica" com 16,71% e, por fim, a classe "Baixa" com 1,86%. Assim, há uma concentração de 62.47% de vulnerabilidades nas classes de risco “Alta” e “Crítica”, que representam um grau de vulnerabilidade preocupante para a aplicação.

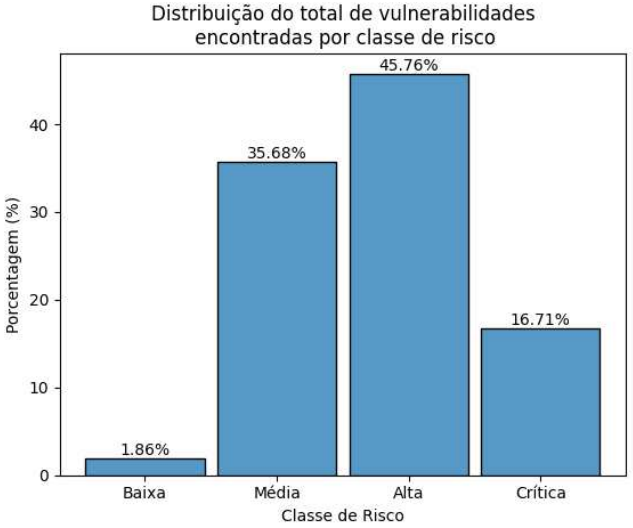

Figura 6. Gráfico de barras representando a distribuição das vulnerabilidades por classe de risco na ferramenta Jfrog Xray.

Analisando a evolução das vulnerabilidades por classe de risco ao longo do tempo para a ferramenta Jfrog Xray, foi gerado um gráfico de linhas (figura 7) e observou-se os seguintes comportamentos para cada tipo:

- Classes “Crítica”, “Média” e “Alta”: Houve um rápido aumento seguido de uma queda expressiva até chegar a zero.
- Classe “Baixa”: Manteve tendência constante, com quantidades próximas de zero.

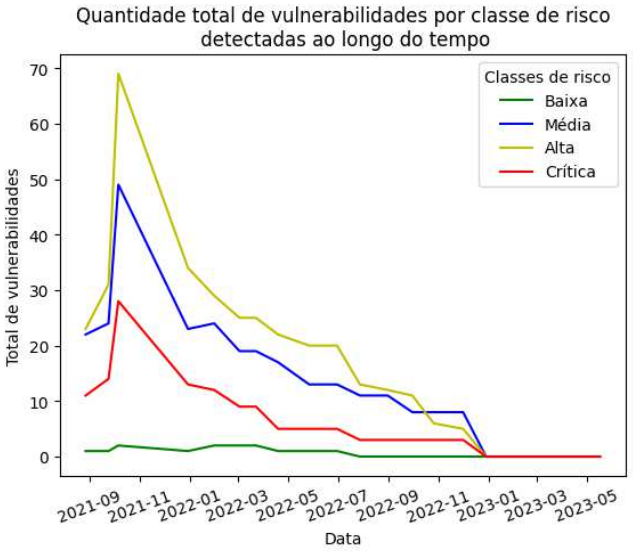

Figura 7. Gráfico de linhas representando a quantidade de vulnerabilidades por classe de risco ao longo do tempo para a ferramenta Jfrog Xray.

Por fim, para a ferramenta Check Marx, gerou-se um gráfico de barras (figura 8) com a distribuição das vulnerabilidades por classe de risco. Observamos que a maior quantidade de vulnerabilidades encontradas pertence a classe “Baixa” com 61.57% das ocorrências, seguida pela classe

“Média” com 37.89% e, por último, a classe “Alta” com apenas 0.53%.

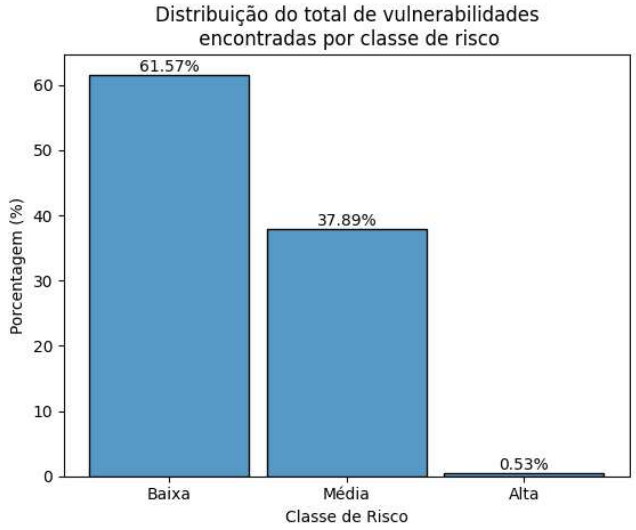

Figura 8. Gráfico de barras representando a distribuição das vulnerabilidades por classe de risco na ferramenta Check Marx.

Ao analisar a evolução das vulnerabilidades por classe de risco ao longo do tempo para a ferramenta Check Marx, utilizando um gráfico de linhas (Figura 9), observou-se os seguintes comportamentos para cada classe de risco:

- Classe “Alta”: Manteve uma tendência constante, com quantidades próximas de zero.
- Classe "Média": Apresentou tendência de crescimento até início de 2023, seguida de uma queda seguida por uma tendência constante.
- Classe “Baixa”: Mostrou curva semelhante a classe de risco “Média” até o início de 2023, quando teve um aumento significativo seguido por uma tendência constante.

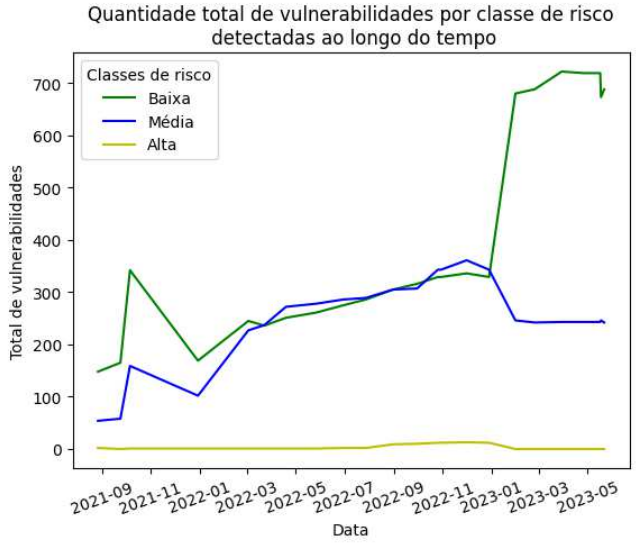

Figura 9. Gráfico de linhas representando a quantidade de vulnerabilidades por classe de risco ao longo do tempo para a ferramenta Check Marx.

Ao observar as distribuições das vulnerabilidades, notou-se que, para as ferramentas SCA, as distribuições são semelhantes, com concentração nas classes de risco “Crítica”, “Alta” e “Média”. Já para a ferramenta SAST, a concentração está nas classes de risco “Baixa” e “Média”.

Ao analisar a evolução das vulnerabilidades por classe de risco ao longo do tempo, observamos que, para as ferramentas SCA, as vulnerabilidades com diferentes classes de riscos apresentaram queda em intervalos diferentes, mas, no geral, houve uma queda indicando que houve uma priorização para resolvê-las. Para a ferramenta SAST, não houve priorização na resolução das vulnerabilidades da classe de risco "Baixa", uma vez que houve um aumento expressivo na quantidade de vulnerabilidades durante quase todo o intervalo analisado.

Devido a concentração de 61.57% das vulnerabilidades encontradas pela ferramenta SAST na classe de risco "Baixo", essa situação pode parecer menos preocupante em comparação com as ferramentas SCA, onde as ferramentas Black Duck e JFrog Xray apresentaram concentrações de 41,29% e 62,47%, respectivamente, de vulnerabilidades com classes de risco “Alta” e “Crítica”, que representam um grau de vulnerabilidade preocupante para a aplicação.

Embora possa parecer menos crítico, é importante ressaltar que o custo para remover essa quantidade significativa de vulnerabilidades em pontos de código tende a aumentar a medida que são negligenciadas. Portanto, é crucial abordar essas vulnerabilidades o mais cedo possível, a fim de evitar problemas que possam acarretar em altos custos para sua remoção.

4.3 Tempo de vida das vulnerabilidades

Na terceira parte da análise, buscou-se compreender o tempo de vida das vulnerabilidades no projeto. Para isso, calculamos os tempos de vida em dias das vulnerabilidades, considerando a primeira e última data em que elas foram detectadas pelas ferramentas.

Ao observar o gráfico com boxplots (Figura 10), notou-se que não há muita variabilidade no tempo de vida das vulnerabilidades das ferramentas de Análise de Composição de Software (SCA). visto que as diferenças entre o terceiro quartil e o primeiro quartil são bem semelhantes para as duas ferramentas. A média do tempo de vida para ambas também é semelhante, em torno de 95 dias. Isso pode ser explicado pela periodicidade das atividades de remoção de vulnerabilidades, que aconteciam, quase sempre, a cada três ou quatro sprints (duas semanas para cada Sprint).

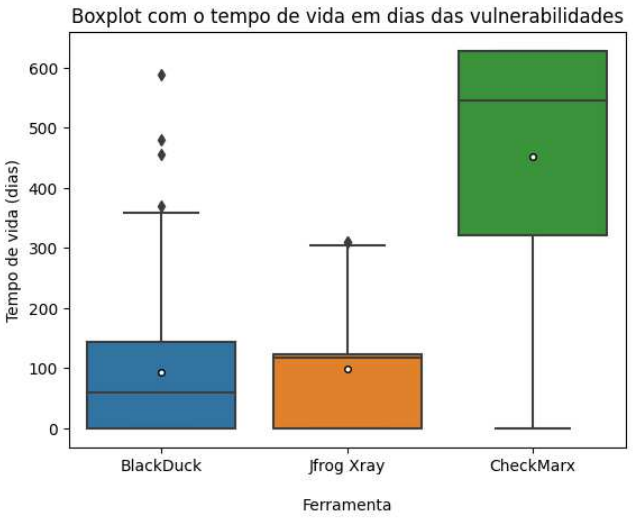

Figura 10. Gráfico com boxplots mostrando a distribuição do tempo de vida das vulnerabilidades por ferramenta.

Na mesma figura 10, para a ferramenta de Teste de Segurança Estática de Software (SAST), Check Marx, foi observado uma grande variabilidade no tempo de vida das vulnerabilidades, evidenciada pela ampla dispersão no boxplot. A

média do tempo de vida é consideravelmente grande, com 452 dias, cerca de um 1 ano e 2 meses. Isso pode ser explicado por não haver atividades de remoção de vulnerabilidades no código do projeto durante todo o intervalo de tempo analisado.

Portanto, era esperado que o tempo de vida das vulnerabilidades detectadas pelas ferramentas SCA fossem semelhantes, já que ambas encontram vulnerabilidades em componentes open-source, detectando suas remoções em intervalos semelhantes. No entanto, para a ferramenta SAST, a grande variabilidade no tempo de vida das vulnerabilidades pode indicar que as vulnerabilidades estão sendo introduzidas e mantidas, havendo pouca preocupação com à sua remoção. Conforme visto na seção 4.2, a maioria das vulnerabilidades encontradas encontradas em pontos de códigos do projeto eram das classes "Baixa" e "Média". Isso pode ser um fator que determina a pouca preocupação com a remoção das mesmas.

4.4 Correlações entre variáveis do tempo de vida e classe de risco

Na quarta parte da análise, foi possível identificar que existia alguma correlação entre as variáveis "Primeira aparição", "Última aparição", "Classe de risco" e o "Tempo de vida" em cada ferramenta. Nosso foco principal foi avaliar a correlação entre as variáveis "Tempo de vida" e "Classe de risco", pois isso poderia indicar se houve alguma priorização na remoção de vulnerabilidades com determinadas classes de risco.

Foram gerados gráficos de mapa de calor com os coeficientes de correlação de Pearson para as ferramentas Black Duck (figura 11), Jfrog Xray(figura 12) e Check Marx (figura 13).

Ao observar a figura 11, notou-se as maiores correlações entre as variáveis "Primeira aparição" x "Última aparição" e "Tempo de vida" x "Última aparição", com valores próximos de 0.6, indicando correlações moderadas. No entanto, ambas não fornecem informações úteis para nossa análise. Por exemplo, a segunda correlação nos indica que o tempo de vida aumenta com a data da última aparição, o que é óbvio.

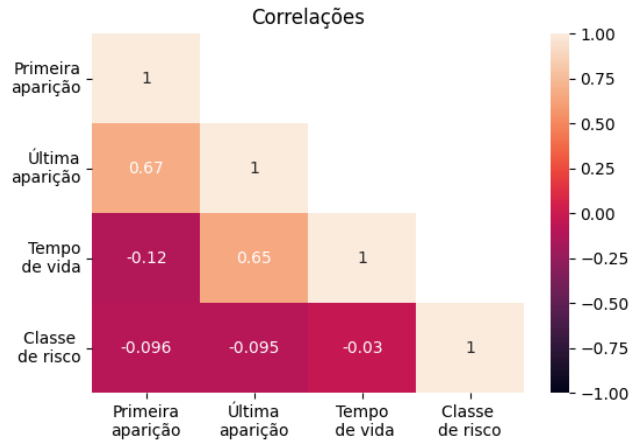

Figura 11. Gráfico mapa de calor mostrando as correlações entre as variáveis para a ferramenta Black Duck.

Na figura 12, encontrou-se duas correlações muito fortes entre as variáveis "Primeira aparição" x "Última aparição" e "Tempo de vida" x "Última aparição", que são relações semelhantes a ferramenta anterior, portanto, não fornecem informações úteis, pois são óbvias.

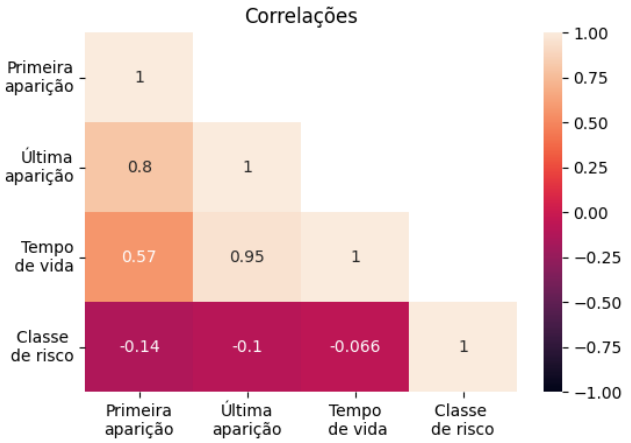

Figura 12. Gráfico mapa de calor mostrando as correlações entre as variáveis para a ferramenta Jfrog Xray.

Por último, na Figura 13, há as correlações entre "Tempo de vida" x "Primeira aparição" e "Tempo de vida" x "Última aparição", sendo uma correlação negativa forte e correlação positiva mediana, respectivamente. Para a correlação entre "Tempo de vida" x "Primeira aparição", ao contrário do que foi observado na correlação da ferramenta anterior, indica que quanto mais antiga for a primeira aparição maior será o tempo de vida, que é óbvio.

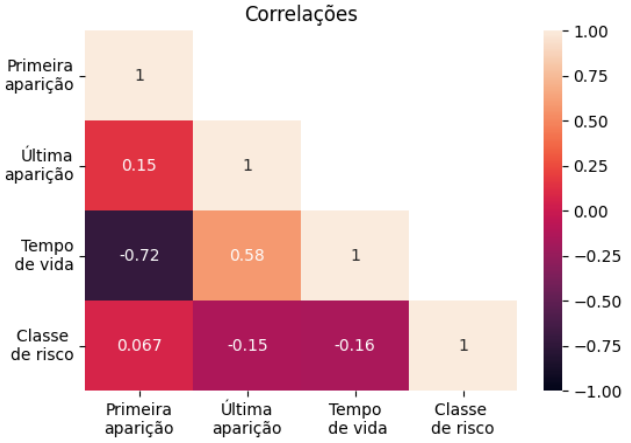

Figura 13. Gráfico mapa de calor mostrando as correlações entre as variáveis para a ferramenta Check Marx.

Como observado, as correlações mais significativas foram encontradas entre variáveis que não forneciam informações úteis e eram esperadas, como a correlação entre as variáveis "Primeira aparição" e "Última aparição". Já a correlação que a priori se tinha maior expectativa ("Tempo de vida" x "Classe de risco") apresentou valores próximos de zero para todas as ferramentas, sendo consideradas correlações fracas. Portanto, é pouco provável que não tenha havido priorização na remoção de vulnerabilidades por classe de risco.

5. LIMITAÇÕES

Durante a realização desta pesquisa, algumas limitações foram identificadas. Na análise da ferramenta de Teste de Segurança de Aplicação (SAST), Check Marx, não foi possível verificar a existência de falsos positivos para as vulnerabilidades identificadas por esta. Além disso, devido ao intervalo de tempo selecionado para geração das versões do projeto, é possível que algumas vulnerabilidades que surgiram e foram resolvidas nesse intervalo da versão (mês), e também pode haver imprecisão no cálculo do tempo de vida das vulnerabilidades.

6. CONSIDERAÇÕES FINAIS

Concluindo a análise realizada, observou-se que as vulnerabilidades presentes em componentes open-source, identificadas pelas ferramentas de Análise de Composição de Software (SCA), estavam sendo removidas e priorizadas em relação às identificadas pela ferramenta de Teste de Segurança de Aplicação Estática (SAST), que afetam pontos de código.

Portanto, como uma medida para mitigar o crescimento contínuo das vulnerabilidades em pontos de código, é necessário criar e integrar atividades focadas na remoção dessas. Utilizando a mesma periodicidade de duas a três Sprints, utilizada para remoção de vulnerabilidades em componentes open-source, é provável que, gradualmente, a quantidade significativa de vulnerabilidades seja reduzida.

6.1 Trabalhos futuros

Para superar as limitações apresentadas na pesquisa, realizar outra análise, considerando intervalos menores, por exemplo, considerando os intervalos entre Sprints, que normalmente são de duas semanas. Isso permitirá uma melhor captura das vulnerabilidades que surgem e são resolvidas ao longo do tempo.

Além disso, para entender o motivo pelo qual as vulnerabilidades em pontos de código não foram removidas, seria interessante investigar se isso se deve pelo fato de ser uma atividade mais trabalhosa, o que pode fazer com que haja resistência por parte dos desenvolvedores. Assim, um levantamento (survey) com os integrantes da equipe de desenvolvimento pode ajudar a identificar os motivos e obstáculos relacionados a essa questão.

7. REFERÊNCIAS

[1] PRESSMAN, Roger S.; MAXIM, Bruce R.. Engenharia de Software: uma abordagem profissional. 8. ed. New York, New York: Amgh Editora Ltda., 2016.

[2] OWASP (org.). OWASP Top 10 - 2017: the ten most critical web application security risks. The Ten Most Critical Web Application Security Risks. Disponível em: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf. Acesso em: 12 dez. 2022.

[3] BLACK, Paul e et al. Dramatically reducing software vulnerabilities: report to the white house office of science and technology policy. National Institute Of Standards And Technology, [S.L.], v. 3, p. 1-64, 08 jul. 2016. National Institute of Standards and Technology. http://dx.doi.org/10.6028/nist.ir.8151. Disponível em: https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8151.pdf. Acesso em: 08 dez. 2022.

[4] BISHOP, Matt. Vulnerabilities Analysis. 1999. 14 f. Monografia (Especialização) - Curso de Computer Science, Department Of Computer Science, University Of California At Davis, Davis, California, 1999. Disponível em: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=46861ecf991406f68dc007fdc0a014aaa929807b. Acesso em: 03 fev. 2023.

[5] LAUDON, Kenneth; LAUDON, Jane. Management Information Systems: managing the digital firm. 10. ed. New York City , United States: Pearson, 2007.

[6] CHECK MARX. Disponível em: https://checkmarx.com/. Acesso em: 02 fev. 2023.

[7] BLACK DUCK. Disponível em: https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html. Acesso em: 02 fev. 2023.

[8] JFROG Xray. Disponível em: https://jfrog.com/xray/. Acesso em: 02 fev. 2023.

[9] DAVE WICHERS. **OWASP Foundation:** Source Code Analysis Tools. Disponível em: https://owasp.org/www-community/Source_Code_Analysis_Tools. Acesso em: 16 mai 2023.

[10] STEVE SPRINGETT. **OWASP Foundation**: Component Analysis. Disponível em: https://owasp.org/www-community/Component_Analysis. Acesso em: 16 mai 2023.

[11] Pandas. Disponível em: https://pandas.pydata.org/. Acesso em: 25 maio. 2023.

[12] Numpy. Disponível em: https://numpy.org/. Acesso em: 25 maio. 2023.

[13] Seaborn. Disponível em: https://seaborn.pydata.org/. Acesso em: 25 maio. 2023.

[14] Postman. Disponível em: https://seaborn.pydata.org/. Acesso em: 25 maio. 2023.

[15] HAROLD BLANKENSHIP. **OWASP Foundation:** Project Update Request - Log4J. Disponível em: https://owasp.org/blog/2021/12/13/Log4J-Alert. Acesso em: 2 junho 2023.

[16] SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Education do Brasil, 2011.

[17] O que é integração contínua? IBM. Disponível em: https://www.ibm.com/br-pt/topics/continuous-integration. Acesso em: 06 jun. 2023.

[18] O que é análise de código estático? JetBrains. Disponível em: https://www.jetbrains.com/pt-br/teamcity/ci-cd-guide/concepts/static-code-analysis/. Acesso em: 06 jun. 2023.

[19] What is SAST and How Does Static Code Analysis Work? Synopsys. Disponível em: https://www.synopsys.com/glossary/what-is-sast.html. Acesso em: 07 jun. 2023.

[20] What is Software Composition Analysis and How Does it Work? Synopsys. Disponível em: https://www.synopsys.com/glossary/what-is-software-composition-analysis.html. Acesso em: 07 jun. 2023.

[21] OWASP Top Ten | OWASP Foundation. Disponível em: https://owasp.org/www-project-top-ten/. Acesso em: 10 jun. 2023.

[22] TOP 25 Software Errors | SANS Institute. Disponível em: https://owasp.org/www-project-top-ten/. Acesso em: 10 jun. 2023.