HuskyCI: um orquestrador de testes de segurança em software para ciclos ágeis de desenvolvimento

Thiago Lotufo[1], **Sérgio Câmara**[1]

[1]Globo Comunicação e Participações S/A
700 BL2, Av. das Américas, 1° Andar, Rio de Janeiro, 22640-100, RJ – Brasil

thiago.lotufo@g.globo, sergio.camara@g.globo

Abstract. *DevSecOps combines development, security, and operations to create an agile and secure software delivery process. The methodology promotes the integration of security in the development cycle, which reduces remediation costs and efforts. CI/CD pipelines automate the code flow from build to production, while SAST and SCA tools detect security vulnerabilities. In this article, we present HuskyCI, an open source tool that orchestrates security tests in CI pipelines, offering support for multiple languages and integration with existing tools. We conducted an assessment of test execution time and concluded that it does not compromise the agile development cycle of projects.*

Resumo. *DevSecOps combina desenvolvimento, segurança e operações para criar um processo ágil e seguro de entrega de software. A metodologia promove a integração da segurança no ciclo de desenvolvimento, o que diminui custos e esforços de correção. As pipelines de CI/CD automatizam o fluxo do código desde a construção até a produção, enquanto ferramentas SAST e SCA detectam vulnerabilidades de segurança. Neste artigo, apresentamos o HuskyCI, uma ferramenta de código aberto que orquestra testes de segurança em pipelines CI, oferecendo suporte a múltiplas linguagens e integração com ferramentas existentes. Conduzimos uma avaliação sobre tempo de execução dos testes e concluímos que ele não compromete o ciclo de desenvolvimento ágil dos projetos.*

1. Introdução

O termo *DevSecOps* (do inglês, *Development*, *Security*, e *Operations*) refere-se a uma metodologia de desenvolvimento, implantação e gerenciamento de ciclo de vida de software. Essa metodologia tem como princípio facilitar o desenvolvimento ágil e seguro, desde a concepção de um software e suas operações, incluindo a sua entrega, implantação e monitoramento. Dessa foma, o *DevSecOps* tem como objetivo estabelecer uma cultura e um conjunto de práticas, com auxílio de ferramentas automatizadas, a fim de promover uma maior colaboração, confiança e transparência entre as principais partes interessadas responsáveis pela entrega de um software [Chandramouli 2022].

Um componente importante utilizado nos processos de desenvolvimento ágil e implantação automática são as *pipelines* (esteiras) de integração contínua e de entrega/implantação contínua (do inglês, *continuous integration and continuous delivery/deployment*), ou CI/CD. As *pipelines* de CI/CD encaminham o software desde a fase de construção até a fase de execução/operação. Dessa forma, elas estabelecem fluxos de trabalho que levam o código-fonte do desenvolvedor por vários estágios, como

construção, testes, empacotamento, entrega e implantação, apoiados por ferramentas automatizadas em todos esses estágios do ciclo [Chandramouli 2022].

No ciclo *DevSecOps*, os aspectos de segurança do software podem ser considerados em todas as fases do ciclo de desenvolvimento. Em geral, quanto mais cedo no ciclo a segurança for abordada, menos esforço e custo serão necessários para alcançar o mesmo nível de segurança. Este princípio é conhecido como *"shift-left"* (em português, "deslocar para a esquerda"), que visa minimizar os débitos técnicos e possibilita corrigir falhas de segurança nas fases inicias ciclo do desenvolvimento, em vez deixá-las para as fases posteriores ou, ainda, após o software estar em produção [Souppaya et al. 2022]. O *shift-left* pode resultar em softwares mais seguros e resilientes, além de diminuir o tempo e esforço de correção de uma falha de segurança.

Para identificar vulnerabilidades de segurança em tempo de desenvolvimento, p.ex. durante um *commit* de código pelo desenvolvedor, as *pipelines* de CI/CD podem ser integradas com ferramentas de testes automatizados de segurança, especificamente as ferramentas de análise estática de código (*Static Application Security Tool*, SAST) e as ferramentas de análise de composição de software (*Software Composition Analysis*, SCA). As ferramentas SAST analisam o código-fonte, o *bytecode* ou o código binário de um software para encontrar possíveis fraquezas de segurança. Elas podem encontrar problemas como *buffer overflows*, código suscetíveis a ataques de injeção e não conformidades em relação às boas práticas de programação (p.ex., senhas expostas no código). Os resultados podem apontar os arquivos, números das linhas e, até mesmo, caminhos de execução afetados para ajudar na correção pelos desenvolvedores. As ferramentas SCA identificam quais bibliotecas e pacotes de código aberto são usados pelo software, verificando se estão desatualizados ou possuem vulnerabilidades de segurança conhecidas [Black et al. 2021].

De acordo com o relatório *Global State of DevSecOps 2023* [Synopsys 2023], as ferramentas SAST são usadas por 72% dos entrevistados, e as SCA usadas por 68% deles. Contudo, a maioria dos entrevistados citou uma insatisfação geral com as ferramentas de análise de segurança que utilizam, e 34% deles acham que elas são muito lentas para caber em ciclos ágeis de entrega/implantação contínua de software.

Dessa forma, percebemos que o tempo de execução total de um estágio da *pipeline* de CI/CD é crítico, uma vez que os desenvolvedores operam em um modelo ágil de desenvolvimento e, em geral, com prazos apertados. O tempo de execução de uma ferramenta de testes de segurança depende não somente da execução da análise do código-fonte em si, mas também de todas as etapas que compõe a execução do estágio, desde a inicialização, *download* de imagens dos *runners*, identificação das linguagens de programação, análise do código, armazenamento de resultados e retorno para o desenvolvedor.

O HuskyCI [Globo 2024a] é uma ferramenta de código aberto que orquestra testes de segurança dentro de *pipelines* de CI e centraliza os resultados em um banco de dados para análises e métricas adicionais. O HuskyCI possui integração com diversas ferramentas SAST e SCA de código aberto, permitindo executar análises de segurança das principais linguagens de programação mais utilizadas atualmente, e foi projetado para ser facilmente integrado às *pipelines* dos projetos de desenvolvimento.

Neste artigo, apresentamos a arquitetura do HuskyCI e a sua implementação ado-

tada pelo time de tecnologia da empresa Globo. Além disso, descrevemos uma avaliação do tempo de execução total de milhares de análises orquestradas pelo HuskyCI em repositórios dos projetos de software, de diferentes tamanhos e linguagens de programação, da empresa. Por fim, analisamos os resultados obtidos e concluímos que o HuskyCI habilita que os desenvolvedores tenham um rápido *feedback* de segurança do código produzido, sem comprometer o ciclo de desenvolvimento ágil dos produtos.

O restante do artigo está organizado da seguinte forma. A Seção 2 elenca outros trabalhos e iniciativas de ferramentas SAST/SCA. A Seção 3 descreve a arquitetura do HuskyCI, sua implementação adotada na Globo, assim como as avaliações realizadas e os resultados. Por fim, a Seção 4 apresenta as considerações finais, e indica os detalhes da demonstração e onde o código-fonte e a documentação da ferramenta podem ser obtidos.

2. Trabalhos relacionados

Nesta seção, mencionamos outras ferramentas de testes SAST e SCA que podem ser utilizadas de forma complementar ao HuskyCI ou integradas a ele.

As ferramentas *GitLab SAST* [GitLab 2024] e *GitHub Code Scanning* [GitHub 2024] oferecem funcionalidades de análise estática de código para diversas linguagens diretamente nas *pipelines* de CI/CD de seus repositórios. No entanto, ambas as ferramentas são integradas apenas às suas respectivas plataformas e suas versões mais avançadas são pagas para uso em projetos privados e empresariais. A *Snyk Code* [Snyk 2024] também é uma ferramenta paga para uso empresarial, que oferece testes SAST e SCA para diferentes linguagens de programação. Ela pode ser integrada tanto a IDEs, através de *plug-ins*, como às pipelines de CI/CD de diferentes plataformas de desenvolvimento. Por sua vez, a *Horusec* [ZupIT 2024] é uma ferramenta de código aberto inspirada inicialmente no HuskyCI e com características semelhantes a ele.

As ferramentas *Semgrep* [Semgrep 2024], *Trivy* [Trivy 2024] são ferramentas conhecidas que oferecem testes de segurança para diversos tipos de linguagens de programação. A *Gosec* [Gosec 2024] oferece testes de segurança apenas para a linguagem Golang. Todas essas três são gratuitas para uso e fazem parte, entre outras, do conjunto de ferramentas que estão integradas ao HuskyCI. Uma lista extensa de ferramentas de testes SAST e SCA, comerciais ou não, e suas características pode ser encontrada em [OWASP Foundation 2024] e [NIST 2024].

3. HuskyCI

Nesta seção, apresentamos o HuskyCI, um orquestrador de ferramentas SAST e SCA de código aberto desenvolvido pelo time de Segurança de Aplicações da Globo. Descrevemos sua arquitetura geral, incluindo seus principais componentes e como eles operam para executar um teste de segurança. Além disso, mostramos como ele foi implementado na empresa, atendendo à maioria dos softwares desenvolvidos internamente e tornando-se uma peça fundamental no programa de Segurança de Aplicações da Globo. Por fim, descrevemos uma avaliação sobre o tempo de execução dos testes orquestrados por ele.

3.1. Arquitetura

O HuskyCI foi projetado para ser integrado à *pipeline* de desenvolvimento de software, sendo uma ferramenta extensível e escalável. Destacamos algumas de suas principais vantagens:

- Multi-linguagem: Suporta a análise de código em várias linguagens de programação, como: Golang, Python, Ruby, Java, Terraform, C++, C#, JavaScript, TypeScript, Lua, VBA, entre outras. Ele integra diversas outras ferramentas de segurança, também de código aberto, que são executadas de acordo com a linguagem de programação contida no repositório analisado;
- Modularidade: Facilmente extensível para incluir novas tecnologias ou ferramentas de análise de segurança;
- Código aberto: Seu código é aberto, disponível no GitHub [Globo 2024a], permitindo que qualquer pessoa contribua e adapte a ferramenta às suas necessidades;
- Baseado em contêineres: Utiliza *containers* Docker ou *pods* Kubernetes para isolar e gerenciar as ferramentas de análise de segurança;
- Escalabilidade horizontal: Possibilita a adição de novos *containers*/*pods* conforme a demanda por testes de segurança em diferentes *pipelines* aumenta;
- Baixo custo de execução e manutenção.

A Figura 1 mostra a arquitetura do HuskyCI. O HuskyCI possui um *client* (*Husky client*) e uma API (*Husky API*) que são responsáveis por iniciar e orquestrar os testes de código respectivamente. O *Husky client* é executado pelo estágio da *pipeline* e se comunica com a *Husky API* de forma autenticada. Ao inicializar, o *Husky client* envia uma requisição HTTP para a *Husky API* com as informações, como *branch* e *url*, do repositório Git em questão. Após receber a requisição, a *Husky API* inicia um processo de pré-análise, armazenando meta-dados do teste específico em seu banco de dados.

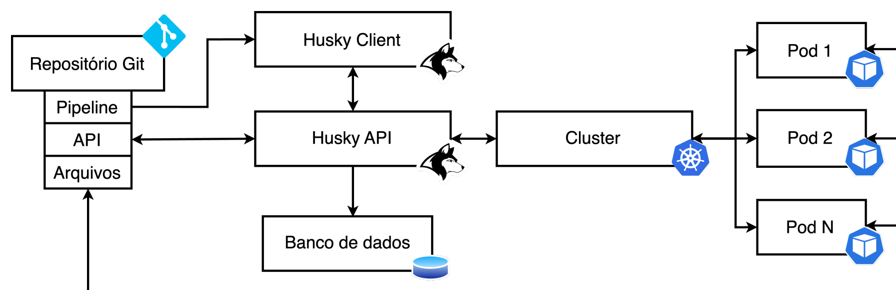


Figura 1. Arquitetura do HuskyCI.

Ainda na pré-análise, há uma etapa de identificação das linguagens utilizadas no repositório Git. Nessa etapa, a *Husky API* faz uma requisição HTTP para uma rota da plataforma Git a fim de obter as linguagens que compõe o repositório a ser analisado. Após obter esta informação, a API começa a criar os *pods* no cluster Kubernetes (ou *containers* no Docker, caso não exista um *cluster* Kubernetes), para cada análise de segurança correspondente a cada uma das linguagens encontradas no repositório. Cada *pod*, portanto, clona o código diretamente do repositório Git e inicia a etapa de análise para uma linguagem específica, executando uma ferramenta terceira para a linguagem identificada. Cabe ressaltar que as análises realizadas por cada *pod* não possuem dependência entre si.

Ao criar um *pod*, a *Husky API* estabelece um canal de comunicação com estes *pods* para que seja possível obter o estado atualizado do *pod*, identificar possíveis erros e ler a saída da análise de segurança executada. Após a finalização da análise de segurança, o resultado dessa é impresso na saída do *pod* que, através do canal de comunicação estabelecido pela *Husky API*, é lido e armazenado no banco de dados. Esses resultados ar-

mazenados podem ser utilizados para análise de dados ou para visualização de métricas, integrados a alguma ferramenta gráfica de *dashboards*.

Ao decorrer do teste, o *Husky client* realiza diversas requisições para uma rota específica da *Husky API*, informando o ID o teste, com o objetivo de buscar o seu estado atualizado. Após o teste estar enfim finalizado, o *client* recebe o resultado consolidado (de todas as análises executadas pelos *pods* para cada linguagem identificada no repositório) como retorno da *Husky API*. Este resultado é mostrado de forma ordenada para o desenvolvedor na saída do *job* no estágio na *pipeline* de CI referente ao HuskyCI.

3.2. Implementação

Na Globo, o HuskyCI foi implementado para ser escalável e adaptado à realidade do ambiente de desenvolvimento da empresa. Por este motivo, o *Husky client*, que é executado no estágio da *pipeline*, está contido em uma imagem *docker* armazenada em um repositório de artefatos interno da empresa. Para o banco de dados, implantamos o *MongoDB* devido à sua flexibilidade em armazenar grandes volumes de dados não estruturados e semi-estruturados. A *Husky API* é executada na plataforma Tsuru [Globo 2024b], o PaaS (*Platform As A Service*) da Globo, e os testes de segurança são executados em um *cluster* Kubernetes na GCP (*Google Cloud Platform*). Como a GCP é o provedor de nuvem principal da Globo, o *cluster* Kubernetes do HuskyCI é executado no GKE, *Google Kubernetes Engine*. No entanto, também é possível utilizar outros provedores de nuvem, ou até mesmo um *cluster* local para executar os testes de segurança.

Como todos esses recursos são escaláveis, o HuskyCI pode executar dezenas de testes simultâneos dadas as centenas do projetos de software atualmente ativos na empresa. A infraestrutura implantada foi planejada a fim de que não haja gargalos ou impacto negativo nos tempos de execução dos testes e, consequentemente, um aumento no tempo total de execução das *pipelines* de CI/CD de desenvolvimento.

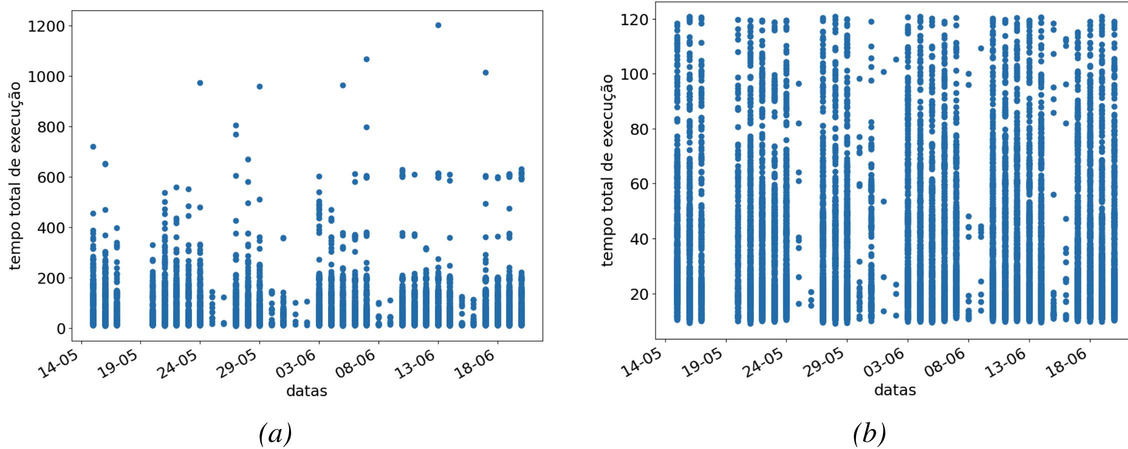
3.3. Avaliação de Desempenho

Nesta seção, apresentamos um estudo do tempo total de execução do HuskyCI, considerando todas as etapas do estágio de teste, como o tempo de preparação na pré-análise, o tempo do teste de segurança em si, e retorno para o usuário. O principal objetivo desta avaliação é identificar o desempenho geral da ferramenta e entender seu impacto, em relação ao tempo de execução, nas *pipelines* de desenvolvimento em projetos reais de software desenvolvidos dentro da Globo.

Para esta avaliação de desempenho, utilizamos algumas bibliotecas *python* para extrair, transformar e gerar as informações necessárias. Os dados de entrada para essa avaliação são definidos abaixo:

- Período observado da avaliação: **15/05/2024** à **20/06/2024**;
- Total de repositórios de código únicos: **551 repositórios**;
- Total de testes executados nesse período: **15.817 testes executados**.

Vale ressaltar que *(i)* um repositório de código pode possuir uma ou mais *branches*; *(ii)* uma *branch* pode possuir um ou mais *commits* de código; *(iii)* cada *commit* corresponde a um teste executado pelo HuskyCI. Além disso, o HuskyCI não possui, por enquanto, a funcionalidade de teste incremental de código (que considera apenas o



(a) *(b)*

Figura 2. Tempos de execução (seg) antes (a) e depois (b) da eliminação de outliers.

trecho de código adicionado no *commit*), portanto ele realiza o teste em todo o código do repositório. Em todos os testes executados, o HuskyCI realiza uma etapa pré-análise para identificar as linguagens existentes, provisiona os *pods* de acordo com as linguagens, clona o repositório em cada um dos *pods* e executa as análises estáticas nos códigos.

Em um primeiro momento, após carregar os dados e realizar as primeiras avaliações, identificamos alguns possíveis valores discrepantes no gráfico de tempo de execução do HuskyCI, conforme observado na Figura 2.a. Esses *outliers* podem ser gerados por diversos motivos, como: *(i) timeouts* nas execuções das ferramentas de segurança; *(ii)* falta de recursos no *cluster* no momento do teste; *(iii)* demora ao fazer o carregamento das imagens nos *pods*, etc. No entanto, como o foco da análise não é identificar os *outliers*, mas sim analisar o tempo de execução da ferramenta e o impacto nas pipelines de desenvolvimento, realizamos um tratamento desses valores.

O método utilizado para tratar os *outliers* foi o *Interquartile Range* (IQR) [Hleap 2024], onde consideramos os dados da amostra entre os limites inferior ($Q1 - 1,5 * IQR$) e superior ($Q3 + 1,5 * IQR$), utilizando um multiplicador = 1,5. Após a implementação do método *IQR*, os *outliers* encontrados anteriormente foram eliminados (Figura 2.b). De modo geral, a base de dados amostral foi reduzida em 1.843 testes, cerca de 11% de redução, totalizando em 13.974 testes.

A Tabela 1 mostra os principais resultados da avaliação de desempenho dos testes observados, antes e depois da eliminação de *outliers*. Os dados, após a eliminação dos casos discrepantes, representam um cenário mais realístico dos tempos de execução do HuskyCI para a grande maioria dos casos observados no período. Identificamos, portanto, que o tempo médio de execução de um teste realizado pelo HuskyCI é de 34,8 segundos, enquanto o tempo máximo de execução de um teste é de 120,8 segundos.

Nessa avaliação, identificamos também as ferramentas que tiveram os maiores tempos de execução nos testes da amostra. Para isso, calculamos o tempo médio de execução de cada uma das ferramentas utilizadas. Os resultados em segundos, da ferramenta com maior duração para menor, foram: Gosec (45,8), Semgrep (32,5), Trivy

Tabela 1. Resultados antes e depois da eliminação de *outliers*

Métricas	Antes da eliminação (seg)	Depois da eliminação (seg)
Tempo médio de execução	53,6	34,8
Tempo máximo de execução	1200,8	120,8
Tempo mínimo de execução	9,1	9,1
Mediana	28,7	25,2
Desvio padrão	67,6	24,1

(17,92), Checkov (17,2), GitLeaks (9,7), Bandit (9,2).

Para melhor entender a distribuição de tempo dos testes da amostra e identificar o impacto pelo tempo de execução, utilizamos o método *KMeans* [Developers 2024] para clusterizar os dados obtidos e agrupá-los por semelhança. Na Tabela 2, mostramos a classificação em três *clusters* para os testes executados pelo HuskyCI, e para as ferramentas SAST (Gosec) e SCA (Trivy) com maior tempo médio de execução na amostra.

Tabela 2. Clusterização dos tempos de execução do HuskyCI, Gosec e Trivy

HuskyCI clusters (seg)	Perc. da amostra	Gosec clusters (seg)	Perc. da amostra	Trivy clusters (seg)	Perc. da amostra
9.1 - 34,2	62,50%	3,6 - 29,9	43,68%	8,8 - 21,4	82,06%
34,2 - 69,8	27,34%	30,1- 74,4	29,56%	21,5 - 43,5	12,31%
69,8 - 120,8	10,15%	74,6 - 120,6	26,74%	43,5 - 118,4	5,61%

Em síntese, dos resultados dessa avaliação de desempenho, obtivemos um tempo médio de 34 segundos e tempo máximo de 2 minutos para realizar todas as etapas necessárias de um teste utilizando o HuskyCI. Da amostra, 62,50% dela foi finalizada em menos de 34 segundos e apenas 10% teve um tempo de execução entre 69 e 120 segundos. Dessa forma, podemos concluir que o HuskyCI habilita aos desenvolvedores um rápido *feedback* de segurança do código produzido, demonstrando um baixo impacto, em relação ao tempo de execução, nas *pipelines* de CI/CD de desenvolvimento de software da Globo.

4. Considerações Finais

Neste artigo, apresentamos o HuskyCI, uma ferramenta de código aberto que orquestra testes de segurança dentro de *pipelines* de CI/CD, centraliza os resultados dos testes de segurança e integra diversas ferramentas de análise. Descrevemos a sua arquitetura geral e a atual implementacão na empresa Globo. Por fim, realizamos uma avaliação de desempenho dos tempos de execução dos testes de segurança e demonstramos um baixo impacto desses testes nas pipelines de CI/CD dos projetos de desenvolvimento da empresa.

Demostração. O código fonte, documentação e instruções de instalação estão disponíveis no repositório do HuskyCI[1]. A demonstração será realizada através de um ambiente disponibilizado por um dispositivo próprio dos autores. As funcionalidades da ferramenta serão apresentadas da seguinte forma: *(i)* instalação e provisionamento da ferramenta em uma máquina local; *(ii)* configuração do teste a ser executado em um repositório específico; *(iii)* demonstração dos resultados e explicação das vulnerabilidades identificadas.

[1]https://github.com/globocom/huskyCI

Referências

Black, P. E., Okun, V., and Guttman, B. (2021). Guidelines on minimum standards for developer verification of software.

Chandramouli, R. (2022). Implementation of devsecops for a microservices-based application with service mesh. Technical report, Gaithersburg, MD.

Developers, S.-L. (2024). Kmeans. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html/. Acessado em: 04 de Julho, 2024.

GitHub (2024). Security at every step — github. https://github.com/features/security. Acessado em: 04 de Julho, 2024.

GitLab (2024). Gitlab sast. https://docs.gitlab.com/ee/user/application_security/sast/. Acessado em: 04 de Julho, 2024.

Globo (2024a). huskyci - an open source sast for devsecops. https://huskyci.opensource.globo.com/. Acessado em 30 de junho de 2024.

Globo (2024b). Tsuru: Uma plataforma de serviço aberta e extensível. https://tsuru.io/. Acessado em: 04 de Julho, 2024.

Gosec (2024). Gosec: Verificador de segurança go. https://github.com/securego/gosec. Acessado em: 04 de Julho, 2024.

Hleap, S. (2024). Unmasking the outliers: Exploring the interquartile range method for reliable data analysis. https://procogia.com/interquartile-range-method-for-reliable-data-analysis/. Acessado em: 04 de Julho, 2024.

NIST (2024). Source code security analyzers. https://www.nist.gov/itl/ssd/software-quality-group/source-code-security-analyzers. Acessado em: 21 de Agosto, 2024.

OWASP Foundation (2024). Source code analysis tools. https://owasp.org/www-community/Source_Code_Analysis_Tools. Acessado em: 21 de Agosto, 2024.

Semgrep (2024). Semgrep. https://semgrep.dev/. Acessado em: 04 de Julho, 2024.

Snyk (2024). Snyk. https://snyk.io/pt-BR/product/snyk-code/. Acessado em: 04 de Julho, 2024.

Souppaya, M., Scarfone, K., and Dodson, D. (2022). Secure software development framework (ssdf) version 1.1 : recommendations for mitigating the risk of software vulnerabilities. Technical report, Gaithersburg, MD.

Synopsys (2023). 2023 open source security and risk analysis (ossra) report. Relatório disponível online em https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/rep-DSO-23-global.pdf.

Trivy (2024). Trivy. https://trivy.dev/. Acessado em: 04 de Julho, 2024.

ZupIT (2024). Horusec. https://github.com/ZupIT/horusec/. Acessado em: 04 de Julho, 2024.