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

MAURÍCIO MARQUES DA SILVA MONTE

AUTOMAÇÃO DE TESTES DE PENETRAÇÃO EM APLICAÇÕES WEB EM AMBIENTES DE INTEGRAÇÃO CONTÍNUA

CAMPINA GRANDE - PB

2024

MAURÍCIO MARQUES DA SILVA MONTE

AUTOMAÇÃO DE TESTES DE PENETRAÇÃO EM APLICAÇÕES WEB EM AMBIENTES DE INTEGRAÇÃO CONTÍNUA

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 : Wilkerson de Lucena Andrade

CAMPINA GRANDE - PB

2024

MAURÍCIO MARQUES DA SILVA MONTE

AUTOMAÇÃO DE TESTES DE PENETRAÇÃO EM APLICAÇÕES WEB EM AMBIENTES DE INTEGRAÇÃO CONTÍNUA

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:

Wilkerson de Lucena Andrade

Orientador – UASC/CEEI/UFCG

Patrícia Duarte de Lima Machado

Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 16 de maio de 2024.

CAMPINA GRANDE - PB

RESUMO

As aplicações web são os principais alvos de ataques cibernéticos, por causa disso a realização de testes que identificam vulnerabilidades neste tipo de sistema é algo primordial. A realização destes testes costuma ser demorada se executada de forma totalmente manual, mas já existem ferramentas que automatizam partes desse tipo de teste. No entanto, ainda existe o problema de escolher qual ferramenta utilizar, dentre as várias disponíveis atualmente. Assim, foi realizado um estudo de caso para comparar várias ferramentas que identificam vulnerabilidades através de análise estática. Das ferramentas selecionadas, a ferramenta Bearer foi a que teve o melhor desempenho na detecção de vulnerabilidades descritas na lista OWASP Top Ten.

AUTOMATION OF PENETRATION TESTING IN WEB APPLICATIONS IN CONTINUOUS INTEGRATION ENVIRONMENTS

ABSTRACT

Web applications are the main targets of cyber attacks, which is why carrying out tests that identify vulnerabilities in this type of system is essential. Performing these tests tends to take a long time if carried out completely manually, but there are already tools that automate parts of this type of test. However, there is still the problem of choosing which tool to use, among the many currently available. Therefore, a case study was produced to compare several tools that identify vulnerabilities through static analysis. Of the tools selected, Bearer tool was the one that had the best performance in detecting vulnerabilities described in the OWASP Top Ten list.

Automação de Testes de Penetração em Aplicações Web em Ambientes de Integração Contínua

Maurício Marques da Silva Monte
Universidade Federal de Campina Grande
Campina Grande, Brasil
mauricio.monte@ccc.ufcg.edu.br

Wilkerson de Lucena Andrade
Universidade Federal de Campina Grande
Campina Grande, Brasil
wilkerson@computacao.ufcg.edu.br

RESUMO

As aplicações web são os principais alvos de ataques cibernéticos, por causa disso a realização de testes que identificam vulnerabilidades neste tipo de sistema é algo primordial. A realização destes testes costuma ser demorada se executada de forma totalmente manual, mas já existem ferramentas que automatizam partes desse tipo de teste. No entanto, ainda existe o problema de escolher qual ferramenta utilizar, dentre as várias disponíveis atualmente. Assim, foi realizado um estudo de caso para comparar várias ferramentas que identificam vulnerabilidades através de análise estática. Das ferramentas selecionadas, a ferramenta Bearer foi a que teve o melhor desempenho na detecção de vulnerabilidades descritas na lista OWASP Top Ten.

Palavras-Chave

Testes, Segurança, SAST, Pentest, Comparativo.

1. INTRODUÇÃO

Quando o acesso a internet se tornou comum, várias empresas decidiram migrar seus produtos para a web em detrimento de aplicações desktop nativas, que costumavam ser mais comuns. As aplicações web forneciam vantagens como não precisar ser instalada na máquina do usuário e poder ser acessada de várias plataformas, de qualquer lugar que possua acesso à internet [1].

Atualmente as aplicações web já se consolidaram no mercado, dado a sua acessibilidade e usabilidade, sendo largamente utilizadas por várias empresas. Por causa disso, tornaram-se atrativas para agentes maliciosos, fazendo com que as aplicações web sejam os principais alvos de ataques cibernéticos, sendo o foco de 17% desses ataques [2].

A realização de testes é uma prática importante no desenvolvimento de software visto que ajuda a garantir que falhas encontradas sejam corrigidas antes da implantação do software, reduzindo custos com manutenção corretiva e preservando a reputação da empresa no setor [3]. Desta forma, é possível entender também a importância dos testes de segurança, uma vez que estes ajudam a prevenir vulnerabilidades de segurança que permitem invasões que geram prejuízos, tais como roubo de senhas, instalação de vírus e programas para acesso remoto, perda de faturamento financeiro devido a ataques de negação e paralisação de serviços, dentre outros [4].

Dado este contexto, é de extrema importância que, durante a fase de desenvolvimento, haja um investimento em testes de penetração, que é um tipo de teste de segurança no qual se realiza ataques planejados a fim de identificar vulnerabilidades de segurança em um sistema computacional [6].

O processo para realizar testes de penetração é naturalmente demorado pois requer que os testadores entendam como funcionam os sistemas e as redes sendo testadas, bem como as ameaças e os métodos de ataque que estão sendo mais utilizados [5]. Se executado de forma totalmente manual, esse processo tende a ser mais vagaroso, pois envolve etapas que levam um tempo para serem executadas tais como: coleta e interpretação de dados sobre o sistema sendo testado, exploração das vulnerabilidades encontradas e por fim a correção das vulnerabilidades [4].

Já existem, atualmente, ferramentas que automatizam partes desse processo fazendo com que assim, o processo seja mais rápido e portanto, menos custoso. Em geral, cada ferramenta investiga apenas alguns tipos de vulnerabilidades, pois os processos de testagem costumam ser diferentes para cada uma. Ainda, estas ferramentas podem fazer diferentes tipos de análises: algumas fazem análises dinâmicas (DAST), que executam testes com a aplicação em execução; enquanto outras fazem análise estática (SAST), que fazem varreduras no código fonte em busca de vulnerabilidades.

Com isso, surge então a necessidade de estudos comparativos para entender quais ferramentas atuam sobre quais tipos de vulnerabilidades e quais as vantagens e desvantagens presentes entre ferramentas que verificam um mesmo tipo de brecha de segurança.

A integração contínua é um processo de automação que ajuda desenvolvedores a consolidar mudanças no código em um uma ramificação compartilhada. À medida que as atualizações são feitas, etapas de teste automatizados são executadas para garantir a confiabilidade das mudanças a serem consolidadas no código [15]. Considerando que este ambiente já é apropriado para a execução de testes automatizados, faz sentido que este seja o ambiente escolhido para se adicionar uma etapa adicional de testes para se realizar testes de segurança.

Sendo assim, o objetivo deste trabalho é investigar ferramentas open source que automatizam partes dos testes de penetração, com foco em ferramentas SAST, visando escolher uma ferramenta para ser adicionada em um ambiente de integração contínua. As ferramentas analisadas foram SonarQube Community, Bearer e Horusec. Para tal, foi realizado um estudo comparativo entre as ferramentas considerando a capacidade de detecção de vulnerabilidades, a precisão de cada ferramenta, bem como o tempo de execução e a dificuldade de instalação de cada uma.

As ferramentas foram escolhidas com base em quão ativo está o desenvolvimento e manutenção da ferramenta, na quantidade de vulnerabilidades detectadas, na quantidade de documentação que cada uma disponibiliza e também se a ferramenta é possível de ser adicionada a um ambiente de integração contínua.

2. FUNDAMENTAÇÃO TEÓRICA

Esta seção tem como objetivo descrever a respeito de testes de penetração, a fim de apresentar os conceitos que serão

necessários para compreender o conteúdo que será abordado neste trabalho.

2.1 Pentest

Testes de penetração (ou pentest) é um tipo de teste de segurança no qual se realiza ataques planejados a fim de identificar vulnerabilidades de segurança em um sistema computacional [6]. Os especialistas em segurança cibernética não só identificam vulnerabilidades que poderiam ser usadas por invasores, mas também as exploram, sempre que possível, para avaliar o que os invasores poderiam obter após uma exploração bem-sucedida das falhas [7].

O pentest pode ser executado de três formas diferentes, a depender dos objetivos do teste e do nível de informação sobre o sistema auditado que o testador terá acesso: Black-Box, White-Box e Gray-Box.

Nos testes Black-Box, o auditor de segurança realiza o procedimento do pentesting sem ter conhecimento prévio da infraestrutura e funcionamento da rede. Esse cenário é mais típico quando uma empresa quer se proteger contra um ataque de invasão externo, pois a maioria dos invasores terá vindo de fora da rede e não terá esse conhecimento [4].

No tipo White-Box, o pentester terá total conhecimento da infraestrutura da rede testada, tais como: mapeamento da rede, endereços IP usados, firewalls, código-fonte das aplicações. Esse cenário torna-se típico em casos de ataques internos [5].

Já no Gray-Box, o testador tem conhecimento parcial sobre a aplicação a ser testada. É uma combinação dos testes Black-Box e White-Box [4]. A modalidade Gray-Box simula um ataque feito por um usuário comum que possui credenciais de acesso à rede, porém com permissões limitadas [8].

2.2 OWASP

Existem várias metodologias que podem ser aplicadas em um pentest, em que cada uma define quais testes serão executados de acordo com cenário e escopo do projeto. Dentre as principais estão as metodologias OSSTMM, ISSAF, OWASP e WASC-TC. Destas metodologias, a mais consolidada e que é direcionada para testes em servidores e aplicações web é a OWASP [4] e que será a abordagem utilizada neste trabalho dado que a aplicação a ser testada se trata de uma aplicação web.

A OWASP (Open Worldwide Application Security Project) é uma organização sem fins lucrativos que trabalha para melhorar a segurança de software. Trata-se de uma comunidade aberta dedicada a permitir que organizações desenvolvam aplicações confiáveis. Para isso, a OWASP fornece literatura, ferramentas, projetos, fóruns e documentações para qualquer pessoa que esteja interessada em melhorar a segurança da sua aplicação [9].

2.3 OWASP Top Ten

A metodologia OWASP possui uma lista que contém as dez vulnerabilidades mais críticas em aplicações web, a OWASP Top Ten [10]. Essa lista foi elaborada a partir de dados coletados pela por organizações parceiras da OWASP, bem como pesquisas realizadas pela própria OWASP para assim obter os dados mais atualizados possíveis sobre novas vulnerabilidades que venham a surgir [11].

A OWASP Top Ten é atualizada a cada três ou quatro anos, conforme novas ameaças de seguranças vão surgindo. Atualmente, a versão do ano 2021 é a versão mais atual da lista [12].

2.3.1 A01:2021 - Broken Access Control

As políticas de controle de acesso previnem que usuários tenham acesso não autorizado a sistemas de software, ou, ainda, que tenham acesso, mas com ações limitadas. Essa vulnerabilidade se apresenta quando invasores são capazes de burlar políticas de controle de acesso e assim ter acesso a informações e funcionalidades que não deveriam [16].

2.3.2 A02:2021 - Cryptographic Failures

Falhas criptográficas ocorrem quando o sistema de criptografia utilizado pela aplicação não consegue proteger dados sensíveis. A falha pode ocorrer na função de criptografia em si, nas chaves criptográficas ou a nível de usuário [16].

2.3.3 A03:2021 - Injection

Injeção é um tipo de ataque em que um invasor consegue executar código malicioso em uma aplicação web. Uma aplicação pode estar vulnerável a ataques de injeção quando utiliza dados recebidos do usuário, como queries e parâmetros, sem o devido tratamento [16].

2.3.4 A04:2021 - Insecure Design

Uma aplicação com um design seguro pode ter defeitos de implementação que levam a vulnerabilidades que podem ser exploradas. Já um design inseguro é aquele que não pode ser corrigido por uma implementação perfeita, pois a forma como foi pensado resulta em uma aplicação vulnerável a ataques. Um dos fatores que contribuem para um design inseguro é a não utilização de Modelagem de Ameaças durante o processo de planejamento [17].

2.3.5 A05:2021 - Security Misconfiguration

Uma aplicação possui este tipo de vulnerabilidade quando ela contém falhas de configuração que a impedem de impor políticas de segurança de forma eficaz. Geralmente acontece quando os desenvolvedores não configuram headers de segurança ou deixam contas padrões ativadas com as credenciais padrão, por exemplo [16].

2.3.6 A06:2021 - Vulnerable and Outdated Components

Este tipo de vulnerabilidade surge quando uma aplicação utiliza algum componente ou dependência que está desatualizada e vulnerável a ataques [16].

2.3.7 A07:2021 - Identification and Authentication Failures

Essa vulnerabilidade está presente em aplicações que possuem erros na implementação de funcionalidades de autenticação e de gerenciamento de sessão, permitindo que invasores tenham acesso a conta de usuários através de ataques como força bruta, credential stuffing e identity spoofing [16].

2.3.8 A08:2021 - Software and Data Integrity Failures

Ocorre quando a aplicação faz atualizações de seus componentes, como por exemplo, em um ambiente de CI/CD, sem verificar a integridade das atualizações primeiro, para garantir de que estão vindo de fontes confiáveis [16].

2.3.9 A09:2021 - Security Logging and Monitoring Failures

Essa falha de segurança acontece quando o sistema de logging de uma aplicação não foi bem planejado e implementado. Com isso, os responsáveis por manter o sistema não conseguem

detectar ou ter informações suficientes para agir diante de um ataque vindo de um agente malicioso [16].

2.3.10 A10:2021 - Server-Side Request Forgery (SSRF)

Uma falha de SSRF acontece quando um aplicativo web tenta acessar um recurso de outro servidor sem validar a URL fornecida pelo usuário. Ele permite que um invasor force o aplicativo a enviar uma solicitação criada para um destino inesperado, mesmo quando protegido por um firewall, VPN ou outro tipo de lista de controle de acesso à rede [18].

2.4 Ferramentas de Automação de Pentest

Já existem várias ferramentas que conseguem automatizar muitos processos do pentest que se fossem feitos manualmente, demandam um tempo muito alto. Apesar desses scanners auxiliarem no processo de detecção de vulnerabilidades, cada aplicação web é escrita de forma diferente e às vezes podem ser retornados resultados contendo vulnerabilidades não existentes (falso-positivo) [13]. Desta forma, uma análise manual sobre os resultados das ferramentas ainda é necessária.

Essas ferramentas de detecção de vulnerabilidades se dividem em dois tipos: SAST e DAST.

Ferramentas do tipo SAST (Static Application Security Testing) realizam testes do tipo White-Box, varrendo o código-fonte da aplicação em busca de vulnerabilidades de segurança. São ferramentas que são executadas antes do código ser implantado, para que os desenvolvedores possam corrigir os problemas detectados ainda em fase de desenvolvimento. Pelo fato de analisarem o código fonte da aplicação, essas ferramentas são específicas para cada linguagem [14].

Ferramentas DAST (Dynamic Application Security Testing) realizam testes do tipo Black-Box, que fazem um scan de uma aplicação em execução para tentar detectar vulnerabilidades que agentes maliciosos poderiam utilizar. São ferramentas que atuam em um ambiente homologação, em que a aplicação já foi implantada, mas que ainda não está em produção. Em geral, os tipos de vulnerabilidades que as ferramentas DAST conseguem detectar, são diferentes dos tipos detectados pelas ferramentas SAST [14].

3. METODOLOGIA

Para realizar a comparação entre o desempenho das ferramentas de pentest automático, foi escolhido o TCoM Server, que se trata de uma aplicação web que é utilizada para gerenciamento de testes em dispositivos. Trata-se de um sistema real que foi desenvolvido em uma cooperação entre o SPLAB/UFCG e a Ingenico, que utiliza o TCoM Server diariamente em suas várias sedes na América Latina.

Nesta aplicação, é possível cadastrar testes e planos de testes, bem como gerenciar os usuários que podem ter acesso a esses planos. Também é possível visualizar resultados das execuções de planos de testes e gerenciar o acesso dos usuários a esta e outras aplicações do ecossistema da empresa que utiliza esta aplicação. Essa aplicação é composta por três componentes principais: frontend, backend e banco de dados.

O projeto utiliza javascript como linguagem de programação tanto para o backend como para o frontend. O backend, é composto de 576 arquivos, resultando em um total de 98354 linhas de código e utiliza o framework express[1] para implementar o servidor web. O frontend possui 390 arquivos, gerando em torno de 70150 linhas de código e utiliza React[2] como biblioteca para construir as interfaces de usuário.

Para limitar o escopo do estudo, foram utilizados alguns critérios para filtrar as ferramentas que seriam utilizadas neste trabalho.

O primeiro critério foi o tempo desde a última atualização da ferramenta. As ferramentas precisam se atualizar conforme novas formas de se explorar vulnerabilidades vão surgindo. Sendo assim, ferramentas que estiveram mais de dois anos sem atualização foram descartadas.

Outro critério utilizado foi a quantidade de vulnerabilidades que a ferramenta consegue identificar, sendo dada preferência a ferramentas que detectem mais vulnerabilidades. Com isso, o custo de manutenção dos testes do pipeline seria menor, uma vez que apenas uma ferramenta que checa várias vulnerabilidades necessita de ser configurada, ao invés de ter que configurar várias ferramentas em que cada uma verifica uma vulnerabilidade.

Ainda, foi considerada a quantidade de documentação disponibilizada, dando prioridade para aquelas que possuíam melhor documentação uma vez que isso facilita durante o processo de configuração.

A comparação entre as ferramentas foi realizada conforme os critérios apresentados na Tabela 1.

Critérios para a Análise Comparativa
Vulnerabilidades Detectadas
Precisão
Tempo de Execução
Dificuldade de Instalação

Tabela 1. Critérios utilizados na análise comparativa entre as ferramentas.

A análise foi feita avaliando as vulnerabilidades detectadas por cada ferramenta e comparando os resultados obtidos. Foi julgado também a precisão das ferramentas, gerando penalizações para as ferramentas que reportaram detecções de brechas de segurança que foram classificadas como falso positivo.

Além disso, o tempo de execução das ferramentas foi considerado, adotando-se uma margem aceitável para diferença do tempo de execução para as ferramentas. Por fim, também foi feita uma avaliação a respeito da dificuldade de instalação de cada ferramenta.

3.1 Ferramentas

As ferramentas utilizadas foram a Horusec v2.9.0-beta.3, a Bearer v1.43.1 e a SonarQube Community v10.4.1.

Bearer é uma ferramenta SAST open source de linha de comando que realiza scans no código fonte e analisa o fluxo de dados para descobrir, filtrar e priorizar riscos de segurança e

[1] Disponível em: <https://expressjs.com/pt-br/>. Acesso em: 08/04/2024.

[2] Disponível em: <https://react.dev/>. Acesso em: 08/04/2024.

privacidade. Atualmente, ela dá suporte às linguagens Ruby, Javascript/Typescript, Java, PHP, GO e Python [19].

Horusec é uma ferramenta open source que realiza análise estática no código para identificar falhas de segurança durante o processo de desenvolvimento. A ferramenta tem opções para buscar vazamentos de chaves e falhas de seguranças em todos os arquivos do projeto, bem como no histórico do git. Horusec consegue fazer análise em códigos escritos nas linguagens C#, Java, Kotlin, Python, Ruby, Golang, Terraform, Javascript, Typescript, Kubernetes, PHP, C, HTML, JSON, Dart, Elixir, Shell e Nginx [20].

SonarQube Community é a edição de código aberto do SonarQube, uma ferramenta de análise estática desenvolvida pela empresa SonarSource [21]. É capaz de detectar problemas de qualidade de código e de segurança. Funciona como uma espécie de revisão de código automática e tem suporte para mais de trinta linguagens de programação [22].

Todas as ferramentas foram executadas em containers Docker[3], disponibilizados pelos próprios desenvolvedores das ferramentas.

3.2 Coletas de Dados

Para realizar a coleta de dados utilizados neste estudo, foi realizada uma pesquisa na documentação das ferramentas utilizadas a fim de descobrir quais as opções de exportação de dados que essas ferramentas disponibilizam.

Para as ferramentas Horusec[4] e Bearer[5], foram utilizadas opções de linhas de comando que configuram a ferramenta para gerar um relatório em formato JSON assim que o scan for concluído.

Para a ferramenta SonarQube Community, os dados foram coletados através de consultas a uma API REST[6] que é disponibilizada pela ferramenta. Durante as consultas a esta API, foi utilizado um parâmetro que retorna as vulnerabilidades detectadas classificando-as de acordo com qual categoria de vulnerabilidade OWASP ela se encaixa.

Os dados de tempo de execução das ferramentas foram coletados através da ferramenta de linha de comando time[7].

3.3 Pré-processamento

Antes do processo de análise, notou-se que os relatórios das ferramentas continham informações irrelevantes e que, portanto, foram removidas dos arquivos. Com exemplo deste tipo de informação, podemos citar:

- Informações desnecessárias sobre a vulnerabilidade - solução recomendada, links com a descrição detalhada das vulnerabilidades;
- Informações desnecessárias sobre o projeto - nome do projeto, autor do código que gerou a vulnerabilidade;
- Informações sensíveis - Trechos do código, uma vez que a aplicação é proprietária.

3.4 Análise de Dados

Durante a etapa de análise, foram realizados uma série de procedimentos para facilitar a comparação do desempenho entre as ferramentas.

Para isso, utilizou-se Python e a biblioteca Pandas[8] para manipulação e análise de dados. Além disso, para gerar as visualizações, foi utilizada a biblioteca Plotnine[9], também do ecossistema de Python.

O primeiro processo realizado para se comparar melhor as ferramentas foi deixar as classificações de severidade das vulnerabilidades para uma classificação comum entre todas as ferramentas. Para isso, se utilizou como base a ferramenta SonarQube, pois além de esta já ser bem consolidada no mercado, a ferramenta Horusec já possuía uma forma de converter a sua classificação para a classificação utilizada pelo SonarQube. Logo, as categorias de vulnerabilidades detectadas foram definidas como Critical, High, Medium, Low, Warning e Info.

O próximo passo da análise foi entender como cada ferramenta desempenhou em relação a detecção de vulnerabilidades de acordo com a lista OWASP Top Ten. Porém, a SonarQube era a única que retornava essa classificação. A Horusec e a Bearer retornavam as vulnerabilidades de acordo com outra notação, a CWE.

Para a maioria dos casos, foi possível mapear as vulnerabilidades CWE para vulnerabilidades OWASP Top Ten uma vez que a OWASP disponibiliza uma lista de mapeamento de vulnerabilidades CWE para OWASP em seu site. Nos casos em que isso não foi possível, foi feita uma análise manual da mensagem com a descrição da vulnerabilidade que as ferramentas geraram a fim de corretamente alocá-las em uma vulnerabilidades listadas pela OWASP.

O terceiro passo foi filtrar problemas detectados pelas ferramentas que não eram de segurança. As ferramentas Horusec, além de detectar problemas de segurança, também detectam outros problemas com as dependências do projeto, enquanto que o SonarQube Community também detecta problemas de qualidade de código. Portanto, dos problemas detectados destas ferramentas, foram filtrados quaisquer problemas detectados que não fossem relacionados a segurança, para haver uma comparação mais justa, embora essas funcionalidades extras sejam algo positivo a ser considerado.

Por fim, para cada ferramenta, foi feita uma análise manual em cada relatório gerado com o objetivo de identificar se as vulnerabilidades detectadas pela ferramenta realmente eram verdadeiras ou se eram falsos positivos.

4. RESULTADOS

Esta seção apresenta os resultados obtidos com as ferramentas de análise estática através de visualizações. Estas visualizações foram criadas a partir de análises realizadas em cima das

[3] Disponível em: <https://docs.docker.com/get-started/overview/>. Acesso em: 05/05/2024.

[4] Disponível em: <https://docs.horusec.io/docs/cli/commands-and-flags>. Acesso em: 03/04/2024.

[5] Disponível em: <https://docs.bearer.com/reference/commands/#bearer_scan>. Acesso em: 04/04/2024.

[6] Disponível em: <https://next.sonarqube.com/sonarqube/web_api/api/hotspots/search>. Acesso em: 07/05/2024.

[7] Disponível em: <https://man7.org/linux/man-pages/man1/time.1.html>. Acesso em: 30/04/2024.

[8] Disponível em: <https://pandas.pydata.org/docs/user_guide/index.html>. Acesso em: 05/05/2024.

[9] Disponível em: <https://plotnine.org/>. Acesso em: 05/05/2024.

informações geradas por cada ferramenta e possuem o intuito de facilitar a comparação do desempenho entre elas.

4.1 Vulnerabilidades detectadas por classificação OWASP Top Ten

A primeira parte da análise busca entender a capacidade das ferramentas em detectar vulnerabilidades descritas na OWASP Top Ten, bem como a severidade dessas vulnerabilidades detectadas. Para isso, foram criadas visualizações em gráficos de barras para cada ferramenta, onde cada barra representa um uma categoria de vulnerabilidade descrita na OWASP Top Ten e cada barra é composta por várias barras de cores diferentes, com cada cor representando um nível de severidade detectado dentro daquela categoria.

Das vinte e sete vulnerabilidades de segurança que a ferramenta Horusec detectou, treze foram classificadas como A09 (Security Logging and Monitoring Failures), oito como A07 (Identification and Authentication Failures), cinco como A05 (Security Misconfiguration) e uma como A02 (Cryptographic Failures).

Os problemas de segurança encontrados pelo Horusec se distribuíram nas categorias Critical, High, Low e Info. Destas, a mais que foi mais detectada pela ferramenta foi da categoria Info, constituindo a grande maioria dos casos. Em seguida, a severidade de vulnerabilidade mais detectada foi a Critical, depois a High e por último a Low, conforme mostra a Figura 1.

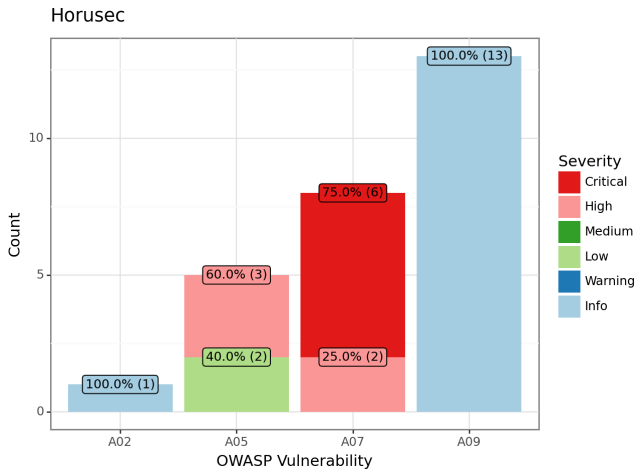

Figura 1. Gráfico de Barra representando a quantidade de vulnerabilidades detectadas pela ferramenta Horusec na aplicação TCoM Server, classificadas por severidade.

Do total de quarenta e nove vulnerabilidades detectadas pela ferramenta Bearer, três vulnerabilidades foram predominantes em relação a outras, possuindo uma quantidade de ocorrências bem próxima entre si. São essas: A03 (Injection), com treze ocorrências; A04 (Insecure Design), com doze ocorrências; e A09 (Security Logging and Monitoring Failures), também com treze ocorrências. Em seguida, tem-se a vulnerabilidade A07 (Identification and Authentication Failures), com sete detecções, e as vulnerabilidades A01 (Broken Access Control) e A05 (Security Misconfiguration), ambas com duas detecções.

As vulnerabilidades detectadas foram classificadas pela ferramenta nas categorias High, Medium e Low. A severidade mais presente foi a High, estando presente em várias categorias diferentes de vulnerabilidades. Em seguida tem-se as de severidade Low e por fim as de gravidade Medium como menos frequentes, conforme mostra a Figura 5.

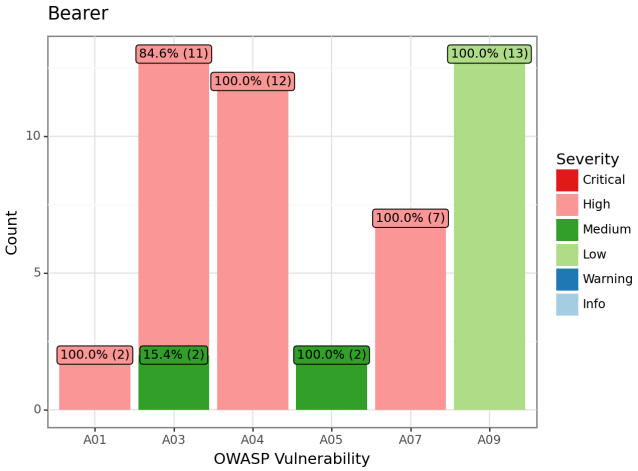

Figura 2. Gráfico de Barra representando a quantidade de vulnerabilidades detectadas pela ferramenta Bearer na aplicação TCoM Server, classificadas por severidade.

A ferramenta SonarQube Community detectou um total de dezenove vulnerabilidades, sendo que a mais presente foi da categoria A03 (Injection), com nove ocorrências. Em seguida, temos as vulnerabilidades A07 (Identification and Authentication Failures), com cinco ocorrências; as A05 (Security Misconfiguration) com quatro detecções e por último a A08 (Software and Data Integrity Failures) com apenas uma detecção e que, até então, não havia sido detectada por nenhuma outra ferramenta.

Em relação a severidade das vulnerabilidades, a maioria foi da categoria High. O segundo tipo mais detectado foi o Low e por último foi reportado duas vulnerabilidades de severidade Medium, conforme mostra a Figura 3.

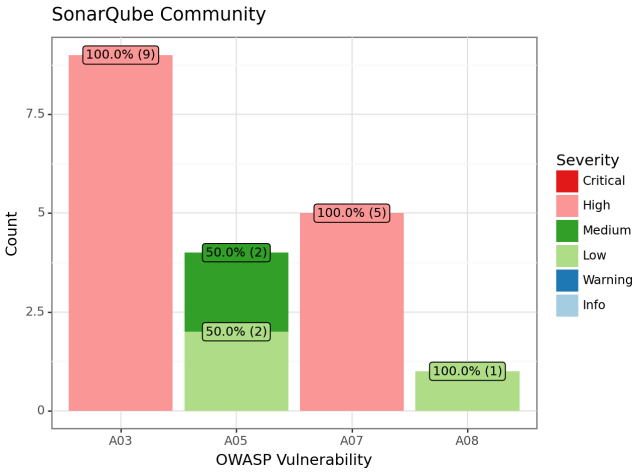

Figura 3. Gráfico de Barra representando a quantidade de vulnerabilidades detectadas pela ferramenta SonarQube Community na aplicação TCoM Server, classificadas por severidade.

Nota-se que, analisando apenas a severidade das vulnerabilidades detectadas, as ferramentas Bearer e SonarQube Community obtiveram um desempenho similar, classificando a maioria das vulnerabilidades como High e as restantes como Medium e Low.

Quando se analisa a performance em relação à capacidade de detectar vulnerabilidades categorizadas na lista OWASP Top Ten, percebe-se que as ferramentas apresentaram um

desempenho diferente entre si. A ferramenta Bearer se destacou neste ponto, uma vez que foi capaz de detectar mais vulnerabilidades nas categorias A03, A04 e A09 do que as outras ferramentas, embora nas outras categorias tenha detectado menos.

4.2 Precisão

Após a análise manual dos dados coletados pelas ferramentas, foi possível definir as vulnerabilidades detectadas como sendo verdadeiros positivos ou falsos positivos. Essa parte da análise visa comparar a precisão de cada ferramenta na detecção de vulnerabilidades verdadeiras. A tabela 2 apresenta os resultados obtidos.

Ferramenta	Total	Verdadeiros Positivos	Falsos Positivos	Precisão
Horusec	86	27	59	31,40%
Bearer	89	49	40	55,06%
SonarQube	28	19	9	67,86%

Tabela 2. Precisões obtidas pelas ferramentas na detecção de vulnerabilidades verdadeiras.

Inicialmente, a ferramenta Horusec havia detectado um total de oitenta e seis vulnerabilidades. Porém, destas, apenas vinte e sete eram reais, resultando em um total de cinquenta e nove falsos positivos, o que representa uma precisão de 31,40% por parte da ferramenta.

Já a ferramenta Bearer havia detectado oitenta e nove vulnerabilidades, sendo que deste total, apenas quarenta e nove eram reais. Portanto, a ferramenta apresentou quarenta falsos positivos, resultando em uma precisão de 55,06%.

Por fim, a ferramenta SonarQube Community havia detectado um total de vinte e oito vulnerabilidades. Entretanto, deste número, apenas dezenove eram casos reais, o que resultou em nove falsos positivos e em uma precisão de 67,86%.

Considerando as precisões obtidas, a ferramenta que teve o melhor resultado foi a SonarQube Community, registrando 67,86% para esta métrica. Em seguida vem a ferramenta Bearer com 55,06% e por último a ferramenta Horusec com 31,40%.

4.3 Tempo de Execução

Nesta etapa da análise, se registrou o tempo médio de execução das ferramentas, para entender qual o impacto da execução destas ferramentas em um ambiente de integração contínua.

Cada ferramenta foi executada cinco vezes e o tempo de execução para cada execução foi registrado. Em seguida, foi feita a média desses registros para se obter o tempo médio para cada ferramenta.

A ferramenta Horusec obteve um tempo médio de execução de 20.35 segundos. A ferramenta Bearer obteve um tempo médio de execução de 29.75 segundos. Por fim, a SonarQube obteve um tempo médio de execução de 80.16 segundos.

É importante ressaltar que a ferramenta Horusec, além de fazer scans por vulnerabilidades de segurança, também faz scans por dependências desatualizadas, bem como a ferramenta SonarQube Community também faz scans de qualidade de código.

Considerando este critério, nenhuma ferramenta possui um tempo demasiadamente longo, embora a ferramenta SonarQube Community tenha um tempo de execução bem maior que as demais.

4.4 Dificuldade de Instalação

Foi analisada a dificuldade de instalação das ferramentas com o intuito de compreender melhor qual a carga de trabalho que a ferramenta impõe aos desenvolvedores que desejam adotá-la e, consequentemente, mantê-la no seu processo de desenvolvimento.

Para isso, se considerou a quantidade de documentação disponível para realizar a configuração da ferramenta bem como a quantidade de componentes necessários para a instalação da ferramenta.

A qualidade da documentação foi avaliada de acordo com a documentação fornecida pelos desenvolvedores da ferramenta e por dúvidas respondidas em fóruns e sites de Q & A. Documentações definidas com qualidade “Boa” são aquelas que foram bem escritas pelos desenvolvedores, mas que não possuem muitas dúvidas em fóruns e sites de Q & A. Documentações definidas como “Muito Boa” se saírem bem nos dois aspectos. A tabela 3 apresenta os resultados obtidos.

Ferramenta	Qualidade da Documentação	Quantidade de Componentes
Horusec	Boa	1 (instalação mínima, apenas CLI) 9 - 12 (instalação com interface web) [23]
Bearer	Boa	1 (apenas CLI) [19]
SonarQube Community	Muito Boa	3 (Scanner, Web Server, Database) [24]

Tabela 3 - Comparativo entre Qualidade de Documentação e Quantidade de Componentes para as ferramentas Horusec, Bearer e SonarQube Community.

Dado este critério, as ferramentas Bearer e SonarQube Community são consideradas como de fácil instalação. A ferramenta Horusec, se usada com uma instalação mínima, também não é difícil de configurar. Porém, ao se optar pela instalação com a interface web, a dificuldade de instalação aumenta consideravelmente. Além disso, a Horusec oferece menos funcionalidades para filtrar quais diretórios devem ser examinados, em relação às outras ferramentas.

5. LIMITAÇÕES

Durante a medição do tempo de execução das ferramentas, devido ao fato de todas estarem configuradas dentro um container docker, o tempo de inicialização do container é incluído junto do tempo de execução da ferramenta.

Além disso, o desempenho das ferramentas neste trabalho pode não representar todos os casos uma vez que a performance de cada ferramenta pode variar de acordo com a linguagem de programação utilizada no projeto alvo a ser escaneado.

6. CONSIDERAÇÕES FINAIS

Este artigo tem o objetivo de comparar ferramentas SAST open source, visando escolher uma ferramenta para ser adicionada em uma etapa de testes dentro de um ambiente de integração

contínua. As ferramentas SonarQube Community, Bearer e Horusec foram selecionadas para o estudo, no qual os seguintes critérios de comparação foram considerados: Vulnerabilidades Detectadas, Precisão, Tempo de Execução e Dificuldade de Instalação.

Em relação ao critério de vulnerabilidades detectadas, o melhor desempenho foi da ferramenta Bearer, detectando quarenta e nove vulnerabilidades. Já a melhor precisão foi obtida pela ferramenta SonarQube Community, com 67,86%. Em relação ao tempo de execução, nenhuma ferramenta apresentou um tempo demasiadamente longo, embora a ferramenta SonarQube Community tenha demorado bem mais do que as outras. Já sobre a facilidade de instalação, todas as ferramentas são fáceis de instalar, embora a instalação completa da ferramenta Horusec seja mais complicada de se lidar.

Portanto, dado que a ferramenta Bearer obteve um desempenho superior às outras ferramentas na detecção de vulnerabilidades OWASP e teve uma precisão próxima à SonarQube Community, esta é mais recomendada caso o objetivo do usuário seja estar mais seguro contra vulnerabilidades OWASP.

Sendo assim, embora as ferramentas Horusec e SonarQube Community possuam funcionalidades extras que também são interessantes para equipes de desenvolvimento, estas funcionalidades podem ser complementadas com outras ferramentas, ao se optar pelo uso do Bearer como ferramenta SAST para análises de segurança.

6.1 Trabalhos Futuros

Para expandir o trabalho realizado neste estudo, é recomendado realizar outras análises utilizando as mesmas ferramentas, porém, utilizando outros projetos alvos a serem examinados, tanto de linguagens diferentes como da mesma linguagem, mas de frameworks diferentes.

Além disso, outra possibilidade de expansão deste estudo é realizar a comparação com ferramentas de análise dinâmica, bem como comparar qual a diferença na capacidade de detecção de vulnerabilidades entre ferramentas de análise estática e de análise dinâmica.

7. REFERÊNCIAS

[1] **The Evolution of Web Applications**. Bluent Tech, 2016. Disponível em: <https://www.bluent.net/blog/evolution-of-web-applications/>. Acesso em: 27/03/2024.

[2] **Threats and vulnerabilities in web applications 2020–2021**. Positive Technologies, 2022. Disponível em: <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020-2021>. Acesso em: 27/10/2023.

[3] **Entenda a importância dos testes de software e como aplicá-los**. Blog Vericode, 2023. Disponível em: <https://blog.vericode.com.br/testes-de-software/>. Acesso em: 09/04/2024.

[4] MORENO, Daniel. **Introdução ao Pentest**. 1 ed. São Paulo: Novatec Editora Ltda., 2015.

[5] PRICE, John. **How Long Does Penetration Testing Take?**. Subrosa Cyber, 2023. Disponível em: <https://subrosacyber.com/blog/how-long-does-penetration-testing-take>. Acesso em: 28/10/2023.

[6] **O que é teste de penetração?**. IBM. Disponível em: <https://www.ibm.com/br-pt/topics/penetration-testing>. Acesso em: 22/10/2023.

[7] WEIDMAN, Georgia. **Testes de Invasão:** Uma introdução prática ao hacking. 4. ed. São Paulo: Novatec Editora Ltda., 2016. 30 p.

[8] **Pentest**: o que é, quais os tipos e por que realizar. EcoTrust, 2022. Disponível em: <https://blog.ecotrust.io/pentest/>. Acesso em: 31/03/2024.

[9] **About the OWASP Foundation**. OWASP Foundation, 2024. Disponível em: <https://owasp.org/about/>. Acesso em: 31/03/2024.

[10] **OWASP Top Ten**. OWASP Foundation, 2023. Disponível em: <https://owasp.org/www-project-top-ten/>. Acesso em: 31/03/2024.

[11] **Introduction**. OWASP Top 10 Team, 2021. Disponível em: <https://owasp.org/Top10/>. Acesso em: 31/03/2024.

[12] **Top 10 OWASP Vulnerabilities for 2023**. SiteLock, 2023. Disponível em: <https://www.sitelock.com/blog/top-10-owasp-vulnerabilities/>. Acesso em: 31/03/2024.

[13] MORENO, Daniel. **Pentest em Aplicações Web**. 1. ed. São Paulo: Novatec Editora Ltda., 2017.

[14] **SAST vs. DAST.** GitLab, 2024. Disponível em: <https://about.gitlab.com/topics/devsecops/sast-vs-dast/>. Acesso em: 31/03/2024.

[15] **O que é CI/CD?**. Red Hat, 2023. Disponível em: <https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd>. Acesso em: 10/04/2024.

[16] **OWASP Top 10 for 2021**: A Summary. Kiuwan, 2021. Disponível em: <https://www.kiuwan.com/blog/owasp-top-10-for-2021/>. Acesso em: 04/05/2024.

[17] **A04:2021 – Design Inseguro**. OWASP Top 10 Team, 2021. Disponível em: <https://owasp.org/Top10/pt_BR/A04_2021-Insecure_Design/>. Acesso em: 04/05/2024.

[18] **A10:2021 – Server-Side Request Forgery (SSRF)**. OWASP Top 10 Team, 2021. Disponível em: <https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/>. Acesso em: 04/05/2024.

[19] Bearer. Disponível em: <https://github.com/bearer/bearer?tab=readme-ov-file>. Acesso em: 03/05/2024.

[20] Horusec. Disponível em: <https://github.com/ZupIT/horusec?tab=readme-ov-file#about> Acesso em: 03/05/2024.

[21] **About Us**: learn about Sonar's values, history and leadership. SonarSource, 2008-2024. Disponível em: <https://www.sonarsource.com/company/about/>. Acesso em: 08/04/2024.

[22] CHIBUZOR, MacBobby. **Inspect your code with Docker and SonarQube**. LogRocket, 2022. Disponível em: <https://blog.logrocket.com/inspect-code-docker-sonarqube/>. Acesso em: 03/04/2024.

[23] **Instale utilizando Docker-Compose**. Zup IT, 2022. Disponível em: <https://docs.horusec.io/docs/pt-br/web/installation/install-with-docker-compose/#imagens>. Acesso em: 04/04/2024.

[24] **Introduction to the server installation**. SonarSource, 2008-2024. Disponível em:

<https://docs.sonarsource.com/sonarqube/latest/setup-and-upgrade/install-the-server/introduction/#instance-components>. Acesso em: 03/05/2024.