Análise estática de código para detectar vulnerabilidades web

Marcos Paulo Arruda Junior, Charles Boulhosa Rodamilans

[1]Universidade Presbiteriana Mackenzie (UPM)
Rua da Consolação, 930 Consolação, São Paulo - SP, 01302-907 – Brazil

juninho1_2010@hotmail.com; charles.rodamilans@mackenzie.br

Abstract. *Static code analysis is a very important step during the software development cycle, in order to detect vulnerabilities at an early stage. Developers use automated tools that analyze the entire code to find possible problems. However, these tools are not free from problems, as some can detect a considerable amount of inaccurate data, such as false positives and false negatives, and can be harmful to organizations. In this article, a practical approach was used to evaluate four different static analysis tools for the PHP language, where two of these tools are for commercial use and the other two were used for free versions. For the Java language, two tools for commercial use were used. This approach considers an analysis methodology where the best tool is the one that reports the most vulnerabilities with the least number of false positives. To compare the tools, controlled Benchmarks were used that had specific vulnerabilities, XSS and SQLi.The results showed that all tools for commercial use had a superior performance of up to 66% approximately, compared to open source tools, however, not all vulnerabilities were detected by the tools.*

Resumo. *Análise estática de código é uma etapa muito importante durante o ciclo de desenvolvimento de um software, para realizar a detecção de vulnerabilidades no seu estágio inicial. Os programadores fazem o uso de ferramentas automatizadas que analisam o código inteiro com a finalidade de encontrar possíveis problemas. Entretanto, essas ferramentas não estão livres de problemas, pois algumas podem detectar uma quantidade considerável de dados imprecisos, como falsos positivos e falsos negativos, podendo trazer prejuízo para as organizações. Neste artigo, foi utilizada uma abordagem prática para avaliar quatro ferramentas de análise estática diferentes para a linguagem PHP, onde duas dessas ferramentas são de uso comercial e as outras duas foram utilizadas as versões gratuitas. Para a linguagem Java, foram utilizadas duas ferramentas de uso comercial. Essa abordagem considera uma metodologia de análise onde a melhor ferramenta é aquela que reporta a maior quantidade de vulnerabilidades com o menor número de falsos positivos. Para a avaliação dessas ferramentas. Para comparar as ferramentas, foram utilizados Benchmarks controlados que possuíam vulnerabilidades específicas, XSS e SQLi. Os resultados mostraram que todas as ferramentas de uso comercial tiveram um desempenho superior de até 66% aproximadamente, comparado às ferramentas de código aberto, porém nem todas as vulnerabilidades foram detectadas pelas ferramentas.*

1. Introdução

A informação é um produto muito valioso para as empresas e é de interesse das organizações mantê-la segura, pois o vazamento de informações sigilosas, seja de usuários ou até mesmo sobre a empresa, pode ser muito danoso. De acordo com um relatório feito pelo Instituto Ponemon junto com a IBM, evidencia que apenas em 2020 o custo total de uma violação de dados global teve uma média de 3.86 milhões de

dólares [1]. Devido a esses dados, a demanda para o desenvolvimento de aplicações seguras tem crescido exponencialmente. Para manter as informações e o *software* seguros, uma etapa importante é a detecção de vulnerabilidades.

Vulnerabilidade é definida "como uma propriedade dos requisitos de segurança do sistema, design, implementação ou operação que pode ser acidentalmente acionada ou explorada intencionalmente e resultar em uma falha de segurança" (traduzido de [2]). Para a detecção de vulnerabilidades no escopo de um código, a análise estática de código é uma etapa muito importante.

Análise estática de código fornece uma maneira escalável para rever toda a segurança do código fonte de um software, podendo ser utilizada no estágio inicial do ciclo de desenvolvimento de uma aplicação, não havendo a necessidade de que o código esteja executando [3].

Atualmente, esse tipo de análise pode ser feita de forma manual, ou fazendo o uso de ferramentas automatizadas, que são especializadas na detecção de vulnerabilidades e problemas presentes no código fonte do software. É estimado que as ferramentas para análise estática de código tenham a capacidade de detectar metade das vulnerabilidades presentes em um programa [3].

A análise estática é realizada durante a etapa de desenvolvimento do projeto, quando os programadores estão codificando o programa. Muitas vezes, os programadores confiam no compilador para detectar os problemas do software, porém o compilador não detecta tudo e as vulnerabilidades de segurança permanecem no código [4].

Entretanto, apesar da ferramenta escanear o código fonte inteiro, ela possui suas limitações, reportando quantidades elevadas de Falsos Positivos (FP) e não detectando algumas vulnerabilidades importantes que estão presentes nos códigos. [3] Por isso, a escolha de uma ferramenta que retorne uma grande quantidade de FP pode fazer com que o custo do projeto seja elevado, pois haverá retrabalho dos desenvolvedores com a finalidade de encontrar uma vulnerabilidade que não existe no código.

Dentre as vulnerabilidades que estão entre as 10 mais exploradas da web, segundo a *Open Web Application Security Project (OWASP)*[7], temos:

- XSS é uma falha de segurança que afeta aplicações web. Essa falha permite que o hacker injete códigos HTML ou JavaScript maliciosos nas páginas que são mostradas na tela do usuário. Uma execução bem sucedida desse ataque pode mudar completamente o comportamento esperado da aplicação, permitindo com que dados privados do usuário sejam expostos sem seu consentimento (traduzido de [10]).
- SQLi é uma falha de segurança onde o atacante injeta comandos SQL na base de dados através de formulários da página ou pela URL. Esse tipo de vulnerabilidade ocorre quando o servidor não valida todos os campos que são enviados pelo usuário [11].

O presente trabalho tem por objetivo geral avaliar diferentes ferramentas de análise estática de código, com a finalidade de verificar a que possui um melhor desempenho para detectar *Cross-site scripting* (XSS) e *SQL Injection* (SQLi). O objetivo específico é verificar a eficiência das ferramentas, através da utilização de Falsos Negativos (FN).

Para a avaliação das ferramentas, foram utilizados *Benchmarks*, que são diversos códigos, com ou sem vulnerabilidades, que são separados pelas linguagens e tipo de vulnerabilidade. Neste projeto foi utilizado o *benchmark* julia, que é disponibilizado pelo *National Institute of Standards and Technology* (NIST) [9].

Após a análise das ferramentas, utilizando as métricas, foi possível observar que a ferramenta Checkmarx obteve um desempenho aproximadamente 17% superior à ferramenta Appscan, com um *informedness* de 0,73 na análise de XSS da linguagem Java. Já na análise de SQL injection da linguagem Java, a Checkmarx também obteve um desempenho com aproximadamente 5% de superioridade.

Para a vulnerabilidade XSS na linguagem PHP foi observado que a ferramenta Checkmarx teve um desempenho aproximadamente 35% superior à ferramenta Appscan, que ficou em segundo lugar. Porém para a análise de SQLi da linguagem PHP foi observado que a ferramenta AppScan obteve um melhor desempenho com uma superioridade de 61% frente à ferramenta SonarCloud, que ficou em segundo lugar.

Este artigo está organizado conforme as seções descritas a seguir: na seção 1 temos a Introdução; na seção 2, os Trabalhos relacionados; seção 3, a metodologia da pesquisa; seção 4 os resultados; seção 5 a conclusão.

2. Trabalho Relacionados

Esta seção apresenta um breve resumo dos artigos e pesquisas mais importantes, que serviram como base para o conhecimento teórico sobre o tema abordado.

KAUR, A. e outros [2] propuseram o uso de ferramentas para análise estática de código com o código fonte aberto, com a finalidade de comparar os resultados das análises em duas linguagens diferentes, JAVA e C/C++, observando as vulnerabilidades reportadas. Foram usadas as *Common Weakness Enumeration* (CWE) como padrão para classificação das vulnerabilidades. A CWE foi criada para auxiliar o departamento nacional de cibersegurança dos Estados Unidos, estão presentes em sua lista diversos erros como: vulnerabilidades, *bugs*, erros de implementação, código, arquitetura, entre outros. As vulnerabilidades analisadas neste artigo foram as seguintes: (a) *Buffer Overflow;* (b) *Null Pointer Dereference;* (c) *Memory Leak;* (d) *Integer Overflow.* Foi utilizado o benchmark JULIET para a avaliação das ferramentas.

NUNES, P. e outros [3] propuseram neste artigo, o uso de ferramentas de Benchmark para comparar a capacidade dos analisadores em encontrar vulnerabilidades, foram consideradas quatro aplicações reais, em diferentes cenários com diferentes objetivos. O experimento com o benchmark avaliou 134 plugins do WordPress, e cinco ferramentas de código aberto para análise estática em PHP: RIPS, Pixy, phpSAFE, WAP, WeVerca.

GUPTA, M. e outros [5] propuseram uma implementação técnica baseada em mineração de dados, com a finalidade de minimizar o número de falsos positivos que são reportados. A ferramenta foi utilizada para analisar as seguintes linguagens: PHP e MySQL. Sendo avaliada em seis aplicações diferentes. Os autores concluíram que a ferramenta cumpriu seu papel, reportando uma porcentagem pequena (valor aproximado não informado no artigo) de falsos positivos nos ataques realizados.

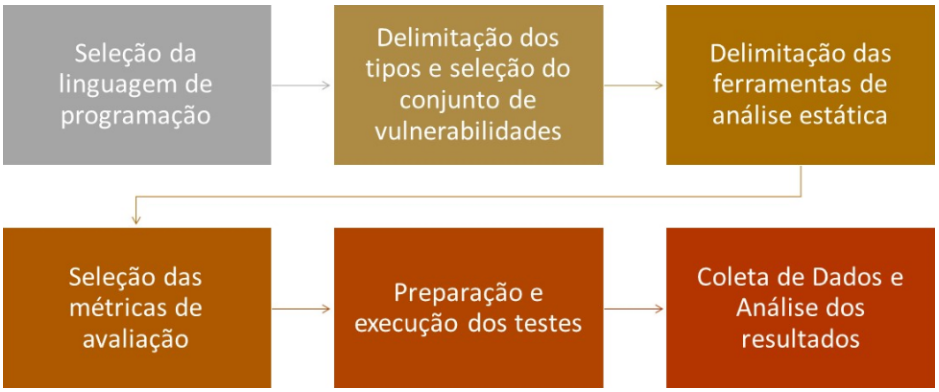
MARCILIO, D. e outros [6] reportaram neste artigo um método sobre como os desenvolvedores utilizam a ferramenta SonarQube, coletando o depoimento de 18 desenvolvedores de diferentes organizações sobre o uso dessa ferramenta. 66%dos participantes que tiveram o depoimento coletado concordaram que essa ferramenta é importante para garantir a qualidade do software.

GOSEVA-POPSTOJANOVA, K. e outros [2] propuseram a análise de diferentes ferramentas para análise estática de código, com o objetivo de verificar os pontos fortes e fracos de cada uma. Foram analisados dois programas de código fonte aberto, um implementado em C e o outro em Java. O benchmark test suite Juliet também foi utilizado para avaliação das ferramentas. Apenas ferramentas de uso comercial foram avaliadas neste experimento. Essas ferramentas, foram testadas em diferentes cenários definidos pelos autores. Os resultados mostraram que a escolha da melhor ferramenta varia de acordo com o cenário que ela atuará, tendo seus pontos fortes e fracos para cada tipo de cenário.

3. Metodologia

No que tange à metodologia que será empregada neste artigo, será utilizada a metodologia que foi adaptada do artigo [3] - a metodologia original envolve diversos cenários que estão fora do escopo deste artigo. A Figura 1 mostra o fluxo que foi seguido durante o desenvolvimento deste artigo:

Figura 1: Fluxograma da metodologia utilizada



Fonte: Autoria Própria

Para a realização dos testes, foram utilizados diversos códigos controlados que possuem a vulnerabilidade desejada (seção 3.2), com isso, foi possível verificar a assertividade das ferramentas na detecção de vulnerabilidades pela análise estática.

Tendo os resultados, foram utilizadas duas métricas com a finalidade de classificar as ferramentas em um cenário que a ferramenta que reportasse um menor número de FP, seria a melhor classificada.

3.1 Seleção da linguagem de programação

As linguagens de programação analisadas foram PHP e JAVA. Segundo o ranking feito pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE), essas linguagens estão entre as 10 mais utilizadas para programação web em 2020 [13].

3.2 Delimitação dos tipos e seleção do conjunto de vulnerabilidades

As vulnerabilidades escolhidas foram *SQL Injection (*SQLi) e *Cross-Site scripting (*XSS), de acordo com um documento das 10 vulnerabilidades mais exploradas desenvolvido pela *Open Web Application Security Project* (OWASP)[7] (Tabela 1), elas estão localizadas em primeiro e sétimo lugar, respectivamente.

OWASP é uma entidade sem fins lucrativos, que atua com foco na colaboração para o fortalecimento da segurança de aplicações *web* ao redor do mundo. É uma entidade de código aberto, onde a comunidade de segurança pode ajudar com abundância de informações, desenvolvimento, testes, implementação e suporte para aplicações. Esse documento foi desenvolvido em 2017, tendo sua última versão atualizada em outubro de 2020 [8].

Tabela 1 - OWASP Top 10.

A1 : 2017- Injeção
A2 : 2017- Quebra de Autenticação
A3 : 2017- Exposição de Dados Sensíveis
A4 : 2017- Entidades Externas de XML (XXE)
A5 : 2017- Quebra de Controle de Acessos
A6 : 2017- Configurações de Segurança Incorretas
A7 : 2017- Cross-Site Scripting (XSS)
A8 : 2017- Desserialização Insegura
A9 : 2017- Utilização de Componentes Vulneráveis
A10 : 2017- Registro e Monitorização Insuficiente

Fonte: OWASP [12]

Nos *Benchmarks* são disponibilizados diversos códigos, com ou sem vulnerabilidades, que podem ser usados para avaliação das ferramentas. Esses códigos são separados por linguagens e pelo tipo de vulnerabilidade presente. Para a linguagem de programação PHP, as ferramentas foram avaliadas através de testes do *Benchmark* Julia, que é disponibilizado pelo *National Institute of Standards and Technology* (NIST) [9]. E para a linguagem JAVA, foi utilizado outro *Benchmark* fornecido pelo NIST [9]. Tendo o *benchmark* como base, as linguagens e vulnerabilidades ficaram separadas de acordo com a tabela 2 abaixo.

Tabela 2 - Lista das linguagens e vulnerabilidades analisadas.

Linguagem	Vulnerabilidade	Número de códigos vulneráveis	Número de códigos seguros
PHP	XSS	4351	500
PHP	SQLi	912	500
JAVA	XSS	976	500
JAVA	SQLi	960	500

Fonte: Autoria própria.

3.3 Delimitação das ferramentas de análise estática

Ferramentas para análise estática de código são frequentemente utilizadas por desenvolvedores, com a finalidade de encontrar bugs ou falhas de segurança. Essas ferramentas possuem a capacidade de percorrer todas as linhas de código e verificar onde a falha está presente.

Neste artigo, foram utilizados diferentes analisadores:

A. SonarCloud - Ferramenta de uso livre para projetos públicos no github, caso haja necessidade de privacidade no projeto é oferecido um plano pago. Essa ferramenta realiza análises das linguagens C#, JavaScript, TypeScript, PHP e Python [14].
B. SonarQube - Essa ferramenta possui quatro tipos de planos, tendo o *Community* para uso livre e os planos Developer, *Enterprise e Data Center* para uso comercial. Não possuindo períodos de testes para fins educacionais, apenas comerciais. Essa ferramenta faz a análise das linguagens C#, Python, PHP, JavaScript, TypeScript, C e C++ [15].
C. AppScan - É uma ferramenta de uso comercial, tendo 30 dias disponíveis para realizar testes. Suporta diversas ferramentas, dentre elas podemos citar as seguintes: Java, ABAP, Python, PHP, C, C + +, Swift, dentre outras [16].

D. Checkmarx - Ferramenta de uso comercial, havendo um período de teste por 15 dias. Suporta diversas ferramentas, dentre as quais podemos citar: Java, Kotlin, Go, C + +, PHP, Python, Cobol, dentre outras [17].

Para a linguagem PHP foram utilizadas as ferramentas SonarCloud, SonarQube, AppSacan e Checkmarx. Para a linguagem JAVA, foram utilizadas as ferramentas AppScan e Checkmarx. Houve uma tentativa de uso da ferramenta integrada ao Github para a análise da linguagem Java, porém não foi possível ao enfrentar diversos erros na construção da análise.

3.4 Seleção das métricas de avaliação

As métricas utilizadas foram baseadas no artigo [3], sendo elas:

A. *Informedness:* É uma métrica que prioriza a ferramenta que reporta um maior número de vulnerabilidades e ao mesmo tempo uma quantidade pequena de falso positivo. Essa será a métrica principal. Tendo sua fórmula definida por:

   VP/(VP+FN) + VN/(FP+VN) - 1

B. *Recall:* Prioriza a proporção de vulnerabilidades verdadeiras que são corretamente identificadas pelas ferramentas. Essa métrica será utilizada apenas para desempate, caso necessário. Tendo sua fórmula definida por:

   VP/(VP+FN)

3.5 Preparação e execução dos testes

Após a separação dos testes por linguagem e tipo de vulnerabilidade, os códigos foram colocados nas ferramentas, que possuem sua execução de forma simples e automática. Após a execução de cada varredura (*scan*), as ferramentas trazem uma lista com o tipo de vulnerabilidade encontrada e o local do código que ela está localizada, além de um relatório geral sobre a qualidade do código.

3.6 Coleta de dados e análise dos resultados

Após a coleta dos dados, os dados foram analisados e apresentados na seção 4.

4. Resultados

Esta seção apresenta os resultados obtidos através de todos os experimentos realizados durante a pesquisa.

Na tabela 3 é possível observar os dados coletados e os resultados de cada ferramenta, para a linguagem PHP com a vulnerabilidade XSS. Esses resultados mostram que a ferramenta Checkmarx obteve um desempenho 35% superior à ferramenta AppScan, pois além de reportar 815 vulnerabilidades, também reportou apenas 113 de FPs.

A tabela 4 apresenta os resultados da análise realizada na linguagem PHP, com a vulnerabilidade SQLi presente. A ferramenta AppScan obteve um desempenho de aproximadamente 61% superior à ferramenta Checkmarx, que teve a segunda melhor classificação, tendo reportado todas as vulnerabilidades presentes sem reportar nenhum FP.

Tabela 3 - Resultados XSS da linguagem PHP - 4351 códigos vulneráveis e 500 códigos não vulneráveis.

Ferramenta	VP	FP	FN	VN	*Recall*	*Informedness*
Checkmarx	815	113	3536	387	0,19	0,65
AppScan	928	244	3423	256	0,21	0,3
SonarCloud	110	0	4241	500	0,03	0,03
SonarQube	0	0	0	500	0	0

Fonte: Autoria própria

Tabela 4 - Resultados SQLi da linguagem PHP - 912 códigos vulneráveis e 500 códigos não vulneráveis.

Ferramenta	VP	FP	FN	VN	*Recall*	*Informedness*
AppScan	912	0	0	500	1	1
Checkmarx	120	91	792	409	0,13	0,39
SonarCloud	264	150	648	350	0,29	0,34
SonarQube	0	0	912	500	0	0

Fonte: Autoria própria.

Na tabela 5 é possível observar os resultados de cada ferramenta, para a linguagem JAVA com a vulnerabilidade XSS. A ferramenta Checkmarx obteve um desempenho 13% superior à ferramenta AppScan, pois reportou 576 vulnerabilidades verdadeiras, detectando apenas 76 casos de FPs.

A tabela 6 mostra os resultados da linguagem JAVA com a vulnerabilidade SQLi. A ferramenta Checkmarx obteve um melhor desempenho, tendo aproximadamente 5% de superioridade à ferramenta AppScan, reportando 542 vulnerabilidades e apenas 89 FPs.

Tabela 5 - Resultados XSS da linguagem JAVA - 976 códigos vulneráveis e 500 códigos não vulneráveis.

Ferramenta	VP	FP	FN	VN	*Recall*	*Informedness*
Checkmarx	576	76	400	424	0,59	0,73
AppScan	100	45	876	455	0,1	0,6

Fonte: Autoria própria.

Tabela 6 - Resultados SQLi da linguagem JAVA - 960 códigos vulneráveis e 500 códigos não vulneráveis.

Ferramenta	VP	FP	FN	VN	*Recall*	*Informedness*
Checkmarx	542	89	418	411	0,56	0,68
AppScan	189	62	771	438	0,2	0,63

Fonte: Autoria própria.

Após a realização de todos os testes, foi possível observar que todas as ferramentas de uso comercial foram melhores quando comparado às ferramentas de código aberto. O SonarQube, na versão community, não foi capaz de encontrar nenhuma vulnerabilidade nos testes realizados. Já o Sonar Cloud, que é um serviço baseado no SonarQube, teve um desempenho melhor conseguindo detectar algumas vulnerabilidades existentes.

5. Conclusão

Neste artigo, foram utilizadas ferramentas de análise estática de código para verificar sua eficiência na detecção de vulnerabilidades web. Foram utilizados *benchmarks* desenvolvidos para as linguagens PHP e JAVA.

Após a análise das ferramentas, utilizando as métricas, foi possível observar que a ferramenta Checkmarx obteve um desempenho aproximadamente 13% superior à ferramenta Appscan, com um *informedness* de 0,73 na análise de XSS da linguagem Java. Já na análise de SQL injection da linguagem Java, a Checkmarx também obteve um desempenho com aproximadamente 5% de superioridade.

Para a vulnerabilidade XSS na linguagem PHP foi observado que a ferramenta Checkmarx teve um desempenho aproximadamente 35% superior à ferramenta Appscan, que ficou em segundo lugar. Porém para a análise de SQLi da linguagem PHP foi observado que a ferramenta AppScan obteve um melhor desempenho com uma superioridade de 61% frente à ferramenta SonarCloud, que ficou em segundo lugar.

Esses resultados apontam que as ferramentas de uso comercial obtém um melhor desempenho quando testadas contra ferramentas de código aberto, portanto é importante

que uma empresa faça investimentos em ferramentas comerciais no momento de fazer a análise estática de seus códigos.

As ferramentas analisadas demonstraram que a análise estática é propensa a erros. Como apresentado neste trabalho, com exceção à ferramenta AppScan na análise de SQLi para a linguagem PHP, nenhuma outra ferramenta foi capaz de detectar 100% das vulnerabilidades presentes nos códigos, para ambas as linguagens, por isso também é de grande importância realizar o uso de diferentes ferramentas no momento do teste .

Como trabalhos futuros pode-se expandir o número de ferramentas de detecção e vulnerabilidades testadas. As ferramentas utilizadas podem ser testadas em aplicações de ambientes reais para verificar a eficácia de detecção em situações reais com códigos mais complexos.

Referências

[1] IBM Security Cost of a Data Breach Report 2020. Disponível em: <https://www.ibm.com/security/digital-assets/cost-data-breach-report/#/pt>. Acesso 20-05-2021.

[1] GOSEVA-POPSTOJANOVA, K and PERHINSCHI (2018), “A. On the capability of static code analysis to detect security vulnerabilities, Information and Software Technology”.

[2] KAUR, A; NAYYAR, R. A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code, Volume 171, 2020, ISSN 1877-0509.

[3] NUNES, P; MEDEIROS, I; FONSECA, J.C; NEVES, N; CORREIRA, M and VIEIRA, M, "Benchmarking Static Analysis Tools for Web Security," in IEEE Transactions on Reliability, vol. 67, no. 3, pp. 1159-1175, Sept. 2018, doi: 10.1109/TR.2018.2839339.

[4] Teixeira, Emanuel Pedro Loureiro. Ferramenta de análise de código para detecção de vulnerabilidades. Diss. Dissertação (Mestrado) — Engenharia Informática, Departamento de Informática, Universidade de Lisboa, 2007.

[5] GUPTA, M. K.; GOVIL, M. C. and SINGH, G. "An approach to minimize false positive in SQLI vulnerabilities detection techniques through data mining," 2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014), Ajmer, 2014, pp. 407-410, doi: 10.1109/ICSPCT.2014.6884962.

[6] MARCILIO, D; BONIFACIO, R; MONTEIRO, E; CANEDO, E; LUZ, W and PINTO, G, "Are Static Analysis Violations Really Fixed? A Closer Look at Realistic Usage of SonarQube," 2019 IEEE/ACM 27th International Conference on Program

[7] Disponível em: <https://owasp.org/>. Acesso 10-05-2021.

[8] H. Sϕhoel, M. G. Jaatun and C. Boyd. "OWASP Top 10 - Do Startups Care?," 2018 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), 2018, pp. 1-8, doi: 10.1109/CyberSecPODS.2018.8560666.

[9] Stivalet, Bertrand C. PHP Vulnerability Test Suite. National Institute of Standards and Technology, 2015. Disponível em:<https://samate.nist.gov/SARD/testsuite.php>. Acesso 10-03-2021.

[10] A. Shrivastava, S. Choudhary and A. Kumar. "XSS vulnerability assessment and prevention in web application," 2016 2nd International Conference on Next Generation Computing Technologies (NGCT), 2016, pp. 850-853, doi: 10.1109/NGCT.2016.7877529.

[11] C. Ping, W. Jinshuang, Y. Lanjuan and P. Lin. "SQL Injection Teaching Based on SQLi-labs," 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE), 2020, pp. 191-195, doi: 10.1109/ICISCAE51034.2020.9236904.

[12] OWASP Top 10 - 2017. Disponível em: <https://owasp.org/www-pdf-archive/OWSP_Top_10-2017-pt_pt.pdf> Acesso 18-05-2021

[13] Interactive: The Top Programming Languages, Disponível em: <https://spectrum.ieee.org/ns/IEEE_TPL_2020/index/2020/1/0/0/0/1/50/1/50/1/50/1/30/1/30/1/20/1/20/1/5/1/50/1/100/1/50/> Acesso 22-05-2021.

[14] SonarCloud Security Coverage. Disponível em: <https://sonarcloud.io/code-security> Acesso 18-05-2021.

[15] Disponível em: <https://www.sonarqube.org/features/multi-languages/> Acesso 18-05-2021.

[16] Static analysis language support. Disponível em: <https://help.hcltechsw.com/appscan/ASoC/src_language_support.html#src_language_support__table_ylp_rn5_jw> Acesso 18-05-2021.

[17] Checkmarx Static Application Security Testing (CxSAST). Disponível em: <https://www.checkmarx.com/products/static-application-security-testing/> Acesso 18-05-2021.