Multitek Indonesia: Jurnal Ilmiah
Volume: 17 No. 1, Juli 2023, Hal. 68 - 78
ISSN: 1907-6223 (print)
ISSN: 2579-3497 (Online)
http://journal.umpo.ac.id/index.php/multitek

DETECTION OF SQL INJECTION VULNERABILITY IN CODEIGNITER FRAMEWORK USING STATIC ANALYSIS

Muhammad Fahmi Al Azhar[1)], **Ruki Harwahyu**[2)]

[1), 2)] Department of Electrical Engineering, Faculty of Engineering, Universitas Indonesia, Depok, Indonesia
Corresponding Email : ruki.h@ui.ac.id

Abstrak

Serangan SQL Injection masih menjadi salah satu jenis serangan yang sering terjadi pada aplikasi berbasis web. Penyebab dan cara mencegah SQL Injection telah banyak dijelaskan di berbagai sumber. Sayangnya sampai saat ini kerentanan SQL Injection masih sering ditemukan pada berbagai aplikasi. Framework aplikasi berbasis web yang sudah memiliki fungsi untuk melindungi dari serangan seringkali tidak digunakan dengan maksimal. Hal ini tidak terlepas dari peran programmer yang seringkali melupakan aturan penulisan kode program untuk mencegah serangan SQL Injection. Penelitian ini kami lakukan untuk mendeteksi kerentanan SQL Injection pada source code menggunakan studi kasus framework PHP CodeIgniter. Kami membandingkan penelitian ini dengan tools static analysis seperti RIPS, Synopsys Coverity, dan Sonarqube. Tool yang kami kembangkan dapat mendeteksi adanya kerentanan SQL Injection yang tidak dapat dideteksi oleh kedua tools tersebut dengan akurasi sebesar 88.8%. Hasil penelitian kami dapat memberikan sugesti terhadap programmer sehingga mereka dapat memperbaiki kode yang mereka tulis.

Kata kunci: *Static Analysis*, *SQL Injection*, Php, Codeigniter

Abstract

SQL Injection attacks are still one type of attack that often occurs in web-based applications. The causes and ways to prevent SQL Injection have been widely explained in various sources. Unfortunately, until now, SQL Injection vulnerabilities are still often found in multiple applications. Web-based application frameworks that already have functions to protect against attacks are often not used optimally. This is inseparable from the role of programmers, who often forget the rules for writing program code to prevent SQL Injection attacks. We conducted this research to detect SQL Injection vulnerabilities in source code using a case study of the PHP CodeIgniter framework. We compared this research with static analysis tools like RIPS, Synopsys Coverity, and Sonarqube. The tool we have developed can detect SQL Injection vulnerabilities that cannot be detected by the two tools with an accuracy of 88.8%. The results of our research can provide suggestions for programmers so that they can improve the code they write.

Keyword: Static Analysis, SQL Injection, Php, Codeigniter.

INTRODUCTION

Web-based applications increasingly play a very important role in various fields [1]. Based on data from the Open Web Application Security Project (OWASP) for 2021, SQL Injection is still one of the top 10 attacks in web-based applications. In 2021, Injection occupied the number 3 position as a source of security risks in web applications [2].

SQL Injection is an exploitation technique by modifying SQL commands in the input form contained in the application, thus allowing the attacker to send SQL syntax to the database. SQL

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

Injection can also occur via parameters in the URL of a specific web application page. SQL Injection occurs due to the lack of validation performed from the input form in the application. SQL Injection has harmful effects such as unauthorized access, loss of confidentiality, and data integrity [3].

SQL Injection vulnerabilities are also inseparable from how programmers/developers write source code. Programmers often use open-source third-party libraries to speed up the application development process. Even though it could be that the library has a vulnerability [4]. By using specific libraries and frameworks, applications can be completed more quickly, but this also increases the vulnerability risk of the applications they make [5].

SQL Injection can occur in any application that uses a relational database, including PHP-based applications. Based on data from W3Tech [6], 77.4% of server-side applications are made using PHP. The size of this percentage is directly proportional to the number of vulnerabilities found in PHP-based applications. Creating applications using PHP frameworks to simplify and speed up application development [7]. Within the PHP framework, a helper function is used to perform queries to the database. The framework also has security features to protect applications from SQL Injection attacks.

SQL Injection consists of several types [3], in general, SQL Injection can be prevented in several ways (1). Sanitation is the process of converting data input into the application into safe data. (2). Validation is the process of ensuring that the input data is data that matches the pattern required by the application. (3). Using the binding function when querying the database. In addition to these three methods, [8] also mentions a configuration on the server that must be considered to minimize SQL Injection vulnerabilities. As long as the input in the database query is secure, SQL Injection attacks should not occur.

The essential thing that programmers can do is implement secure coding [8] to secure the applications they make. Unfortunately, programmers often neglect to implement this, even though the framework often used already has security features. The experience of programmers and the close deadlines for making applications are often the reasons for the emergence of SQL Injection security holes [9].

Static application security testing (SAST) is an option for testing application security. SAST conducts tests based on the application's source code to look for vulnerabilities and defects in the source code. The advantage of SAST is that programmers can immediately make corrections to the detected source code [10]. Unfortunately, SAST tools often produce false positives or false negatives [5].

Sonarqube [11] is one of the industry's most frequently used static analysis tools. Sonarqube is available in both community and paid versions. Sonarqube can scan and calculate using metrics such as code coverage, number of lines of code, and compliance with coding rules. We have tried to test this research object using Sonarqube but could not find any SQL Injection vulnerabilities in the source code.

RIPS [4] is a static source code analyzer to find vulnerabilities in PHP scripts. This tool is free and well-known for its fast scanning speed. Unfortunately, RIPS is out of date and no longer being developed. When we tested the application, it didn't find any vulnerabilities related to SQL Injection.

Synopsys Coverity [12] is a commercial tool capable of performing static application source code analysis. Synopsys provides a comprehensive report but has not been able to find SQL Injection vulnerabilities in the CodeIgniter framework [13].

Several related studies from our research were SQLIFIX [9], using training data to improve PHP and Java application source code. However, this research does not mention how to detect SQL Injection in PHP applications that use frameworks.

Research [14] makes OOPIXY static analysis tools to detect vulnerabilities in PHP programs. OOPIXY is a development of previous research, namely PIXY [15]. The advantage of OOPIXY is that it is an OOP feature that can detect interprocedural data flows in application source code. OOPIXY can

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

detect vulnerabilities in PHP version 5 and above, but there is no discussion about how to detect them for specific frameworks.

In this research, we develop a new method for detecting SQL Injection vulnerabilities in source code using a static analysis approach. The main contribution of this research is to detect coding errors that can cause SQL Injection in the CodeIgniter version 3 framework. This research differs from previous research, where previous research only focused on PHP without a specific framework. Current application development is always created using a framework. Based on a survey conducted by Jetbrains in 2021 [16], 93% of respondents said they used a certain framework to build applications.

In addition, even though a framework is used, if it is not appropriately implemented, the application will remain vulnerable to attacks. With this research, it is hoped that it is easier for developers to check whether their source code is safe and follow the rules for using the CodeIgniter framework.

RESEARCH METHODS

This research takes a case study on a PHP-based application created using the CodeIgniter 3 framework at the Badan Pusat Statistik (BPS). The steps taken in this study were (1) testing the application source code using SAST tools, (2) manually checking the source code, (3) conducting a literature study on the rules for using CodeIgniter, and (4) developing SQL detection tools. Injection, and (5) perform testing and evaluationn.

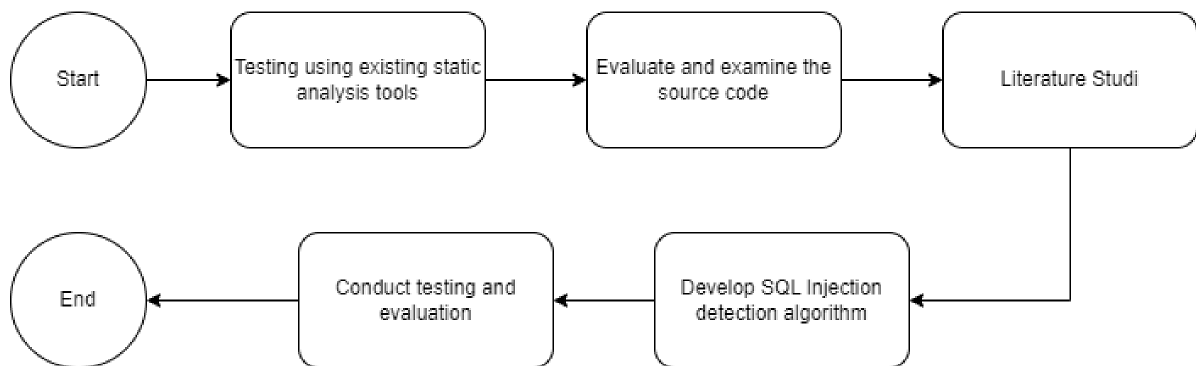

Figure 1. Research Methods

This study uses several SAST tools: Synopsis Coverity, Sonarqube, and RIPS. Synopsys Coverity is a commercial tool that tests the security and defects of an application's source code. Sonarqube is a tool used to detect bugs in source code. Sonarqube also has a feature to detect vulnerabilities in source code. Sonarqube is available in both Community Edition and commercial versions. RIPS is a static source analyzer to find vulnerabilities in PHP scripts.

In addition to using these three tools, we manually check how the application's source was written to find SQL Injection vulnerabilities in the source code. We also ensure that the source code has been written according to the rules for writing code from the CodeIgniter framework.

Testing Using Existing Tools

The first step we took was to examine the application source code used as the object of this

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

research. The examination is carried out using tools and manually. The tools used in this research are Synopsys Coverity, a commercial SAST tool, and RIPS, an open-source tool for performing static analysis on PHP. We also use Sonarqube, a tool for reviewing and finding bugs in source code. We did not find any SQL Injection vulnerabilities in the source code from testing these three tools.

Manual Testing

Furthermore, we manually check the source code to find out which parts of the source code have coding errors and can be the cause of SQL Injection but are not detected by these two tools. We also seek best practices for using certain helper functions in CodeIgniter.

The following table shows coding errors that may occur in the code. We put a checkmark to indicate whether the tools used detected the coding error. Apart from being based on the application's source code, we have also added a list of mistakes that can occur when writing the application's source code.

STUDY OF LITERATURE

A literature study is used to determine the appropriate methodology so that it can solve the problems that exist in this study.

SQL Injection

SQL injection is an attack technique on web applications that exploits the weaknesses of the database system used by the application. In an SQL injection attack, the attacker inserts malicious SQL code into the input entered by a user on a vulnerable web page. As a result, attackers can access or manipulate databases unauthorized. SQL injection is still one of the most common attacks against web applications. These attacks can cause significant organizational losses, such as data theft, deletion, and even illegal access to database-connected systems.

Some best practices to prevent SQL injection include avoiding unverified input, validating input correctly, and using parameterized queries to prevent unintentionally manipulating SQL. In addition, firewalls and input filtering can also help prevent SQL injection attacks.

This study [8] explains in more detail that there is a coding technique approach to prevent SQL Injection, namely:

1. Input and URL validation

   The main cause of security holes in applications is the lack of input and URL validation. When creating an application, the programmer must assume that all input by the user is evil, so validation must be carried out. For example, if a user must fill in a telephone number, the application must validate that the form can only be inputted with numbers.

2. Data Sanitization

   Sanitization is a method for modifying the input data entered by the user to ensure that the data is valid. Sanitization is the most appropriate way to prevent SQL Injection. Data used as input in SQL queries must be clean of dangerous characters, for example, single quotes and white spaces.

   One example of data sanitization is when logging in. The login form usually consists of two fields, namely username and password. The username field should not contain apostrophes because these characters are dangerous characters that can cause errors in SQL queries. Therefore it is necessary to check whether the username field does not contain harmful characters and only accepts characters that are allowed. To remove these dangerous characters, regular expressions can be used. Regular expressions are powerful tools for parsing strings and matching patterns.

   In addition to input, the sanitization of cookies also needs attention. The attacker can modify the cookies stored in the user's browser.

3. Prepared Statement (or PDO) for Query Execution

   Prepared statements are features in the PHP programming language to perform queries more efficiently. PHP is one of the most widely used

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

programming languages, which causes a lot of insecure code. PHP Data Objects (PDO) is a database abstraction layer that programmers can use to work with databases more securely. By using the abstraction layer, programmers don't have to worry about the type of database used. Programmers can focus on the API available by PDO to perform queries into the database.

4. Query and session tokenization

   Query tokenization is a technique for converting input into several tokens. For example, all characters before the quotation marks, dashes, and spaces will be added as a token. By using this method, it will be known whether the query has been injected or not.

CodeIgniter Framework

A PHP framework is a software development or structure written using the PHP programming language. Framework provides a specific set of functions, classes, and rules to facilitate the development of complex and large-scale web applications.

Frameworks make it easier to develop web applications by providing an organized and standardized structure for code generation, thus speeding up the development process and minimizing errors in software development. The framework also offers unique features such as database management, session management, form validation, and a template system.

The advantages of using a PHP framework are increasing efficiency in software development, facilitating ongoing maintenance and development, increasing application security, and reducing errors in software development. However, using the framework also has drawbacks, such as dependence on specific rules, higher complexity, and lack of flexibility in some software development cases.

CodeIgniter is a popular PHP framework because of its small size and easy learning. In addition, CodeIgniter uses the MVC pattern, equipped with helper support, making it easier for developers to create certain features. Unfortunately, developers usually don't adhere to the framework's coding rules. As a result, the code written can cause bugs or security holes.

Figure 2 is a code snippet often found in CodeIgniter controller files. There's nothing wrong with the code at first glance, as it runs without error. However, if we pay more attention, the source code contains vulnerabilities, including SQL Injection. Line 1 is a tainted variable where the user can modify the input. The developer uses the global variable GET function to retrieve the id attribute that the user inputs. Instead of using $_GET, we should use the helper in CodeIgniter, namely $this->input->get()

In line 9, a simple query retrieves records from the database. There's nothing wrong with writing this query, but the $id variable in the query should use parameterized queries/bindings to minimize SQL Injection attacks. By using CodeIgniter, then writing the query should be $query = $this->db->query("SELECT * FROM projects WHERE id=? ORDER BY id DESC", [$id]);

The following error from the source code is that there is no sanitization or validation process for the $id variable, which is user input. The $id variable usually only accepts integers, so at least intval() function can be used, which will take the integer value of a variable. Furthermore, the use form_validation feature in CodeIgniter 3 by using only numeric rules on the $id variable also can prevent any unwanted character.

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

```php
Example.php
1   <?php
2   defined('BASEPATH') or exit('No direct script access allowed');
3
4   class Example extends CI_Controller
5   {
6       public function index()
7       {
8           $id = $_GET['id'];
9           $query = $this->db->query("SELECT * FROM projects WHERE id=$id ORDER BY id DESC");
10          $data = $query->row_array();
11
12          $this->load->view('projects/index', [
13              'data' => $data,
14          ]);
15      }
16  }
17
```

Figure 2. Example of CodeIgniter 3 source which is vulnerable to SQL Injection

Abstract Syntax Tree (AST)

Abstract Syntax Tree (AST) is a data structure representing a computer program's syntax structure. AST is used by compilers and interpreters in processing program code because it makes it easier to analyze, optimize, and transform source code.

The AST consists of nodes representing elements in the source code, such as variables, operations, functions, and conditional statements. Each node in the AST has children (subtrees) that represent the part of the source code associated with that node. In addition, in the AST, each node has a type that indicates the type of element represented by that node, such as "binary expression" for mathematical operations, "function declaration" for function declarations, and so on.

AST is used by compilers and interpreters in the parsing and analysis of source code, where the source code is converted into AST, which is easier for the program to understand and process. AST is also used in the optimization and transformation of the source code, where the source code is converted into a form that is more optimal or according to particular needs.

An understanding of AST can help developers understand and analyze source code, fix syntax errors, perform code optimization, and perform code transformations for specific purposes.

This study uses the Nikic PHP-AST library to facilitate access and analysis of PHP source code. This library supports all features of the PHP programming language, including new features introduced in PHP 7.x, such as type hinting, nullable types, return type declarations, and more.

Tainted Source & Sink

Any input that someone outside the application can modify is a risk that can cause SQL Injection. This input can be a GET parameter from a URL or a form in the application. This input will be stored in a variable on the source code side. SQL Injection can occur if the database server executes the malicious input variable. For each tainted source, at least one of the following treatments should be given:

1. Sanitation, is a process to change the input data to be safe.
2. Validation is the process of ensuring that the input provided by the user matches the pattern required by the application.

A taint sink is any function that can execute a taint source. If not handled properly, taint sinks can be exploited by attackers to carry out SQL Injection attacks.

Regular Expression

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

Regular Expression (Regex) is a pattern or sequence of characters used to match and manipulate text in a programming language. For example, Regex is usually used to search and replace strings (string matching and replacement) in the text that meets a specific pattern.

Regex uses certain characters that have special meanings such as ^, $, *, +, ?, |, (,), [,], {, } as metacharacters. These metacharacters create more complex patterns to match text to the desired character sequence.

This study uses regular expressions to help match specific patterns in source code to find SQL Injection vulnerabilities in that source.

Detection Methods

Based on the literature study and examination carried out in the previous step, we created a detection method which can be seen in Table 1. Our approach is to use the function approach in a file. Checking is done line by line in a function to conclude whether the function is vulnerable. We avoid the per-line approach because it minimizes errors in the checking results.

Table 1. SQL Injection vulnerability detection stages

No	Stages	Description
1.	Detect taint sourcse and variables	By using AST, for each detected function, a variable will be listed in that function. Next will be determined whether it is a taint source or not.
2.	Detect the sanitization of the variable.	Each function will be checked to determine whether it contains sanitation or not.
3.	Detect the presence of form validation.	Each function will be checked to determine whether it contains validation or not.
4.	Detect the presence of dangerous sink	Each function will be checked to determine whether it contains a dangerous sink. This sink can be an intermediary for SQL Injection errors in the web view.
5.	Detect the presence of query and concatenated string	Each function will check whether it contains a query with a concatenated string. Concatenated strings are very dangerous because they can be infiltrated by malicious code that can be executed by the database, thus enabling SQL Injection attacks to be successful. To get the concatenated string, we use regex.

Based on the five steps above, we will conclude that if a function contains a query and contains a concatenated string, a warning must be given. The essence of this check is the existence of a query and a concatenated string in a function. Tainted variables, sanitizers, and validations are only used for informational purposes only with the assumption that the query helper in CodeIgniter has been able to protect against SQL Injection as long as it is used correctly. Table 2 shows the algorithm we use to detect whether a CodeIgniter 3 project contains SQL Injection.

Table 2. SQL Injection Detection Algorithm on the CodeIgniter framework

	Input: path to CodeIgniter project, and application name **Output:** list of files, functions detected as vulnerable to SQL Injection, along with their recommendations
1	**folder project** ← Read PHP files in the application and model folders

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

2	**foreach(file in project folder)**
3	**variable initialization,** to save files and lines
4	**detect all function in a file,** using PHP parser
5	**foreach(fungsi in file)**
6	**check for tainted variable**
7	**check for sanitizer**
8	**check for validation**
9	**check for dangerous sink**
10	**check for query and concatenated string**
11	**draw conclusions**
12	**Save result to database**
13	**endforeach**
14	**endforeach**

The programming language we used to implement this algorithm is PHP using the CodeIgniter framework version 4. We utilize the Spark Command feature in CodeIgniter to make executing the source code to be examined easier. Each stage in Table 1 will be stored in the database for analysis and conclusion. The database design used to store test results is shown in Figure 3 below

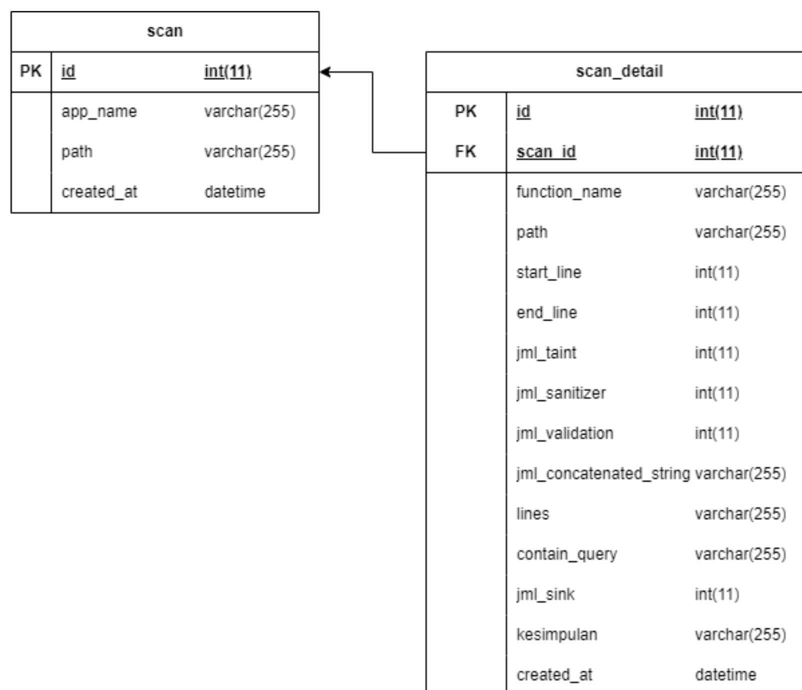

Figure 3. Database design

Saving the test results in a database can facilitate the process of analyzing and developing these tools in the future.

Evaluation Method

The evaluation method used in this research compares the test results from existing tools with the method we created. Next, we manually verify whether the detection results are correct or not.

RESULTS AND DISCUSSION

The test results from this study can be seen in Table 3. We tested 4 real applications at the Badan Pusat

ISSN: 1907-6223 (Print)
ISSN:2579-3497 (Online)

Statistik and disguised the names of these applications for security reasons.

RIPS, Synopsys Coverity, and RIPS could not find SQL Injection vulnerabilities in the CodeIgniter 3 source code. However, the method we proposed was able to find SQL Injection vulnerabilities in 138 functions out of 1182 functions, of which 6 functions turned out to be false positives. This false positive can occur because.

Tabel 3. SQL Injection detection test results

No	Application	Number of SQLI Vulnerable Functions Detected		
		Sonarqube/RIPS	Our Method	False Positive
1.	Application 1	-	69	-
2.	Application 2	-	6	1
3.	Application 3	-	49	2
4.	Application 4	-	14	3

Figure 2 shows one of the functions detected as vulnerable to SQL Injection. The problem with this function is the use of concatenated strings and no binding so that injecting the $id variable in the function is possible.

```
55    public function get_albumfoto($id){
56        return $this->db->query('SELECT * FROM albumfoto WHERE
          id_album ='.$id )->result();
57    }
```

Figure 4. The detected function is vulnerable to SQL Injection attacks

The accuracy obtained from this study is 88.8%, with a very low false positive rate. However, even though our method produces low false positives, we admit that the method we use can only detect up to the function level. To detect up to the line level is very difficult because we have to understand the context and flow of the running program. This research produces tools that are easy to apply to other frameworks by making some adjustments.

CONCLUSION

This study concludes that using AST and Regular Expressions can detect coding errors that can cause SQL Injection in the CodeIgniter version 3 framework. Our method shows very good results for detecting SQL Injection vulnerabilities. By using the tools from this research, the development team can easily test SQL injection vulnerabilities from the source code they write. This research can be further developed to detect the line level by paying attention to the flow of the program. In addition, the results of this research can be integrated into the CI/CD process when developing applications to find vulnerabilities and coding errors early on.

ACKNOWLEDGMENTS

This research was sponsored by the Kementerian Komunikasi dan Informatika Republik Indonesia at program Beasiswa S2 Dalam dan Luar Negeri Tahun 2021. The researchers would like to thank the Kementerian Komunikasi dan Informasi for supporting this research.

REFERENCES

[1] M. Liu, K. Li, and T. Chen, “Security testing of web applications: A search-based approach for detecting SQL injection vulnerabilities,” in *GECCO 2019 Companion - Proceedings of the*

2019 Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, Inc, Jul. 2019, pp. 417–418. doi: 10.1145/3319619.3322026.

[2] N. Larson, "OWASP Top Ten 2021: Where we've been and where we are," 2022.

[3] P. Vats and A. Saha, "An Overview of SQL Injection Attacks," *SSRN Electronic Journal*, May 2019, doi: 10.2139/ssrn.3479001.

[4] A. Ibrahim, M. El-Ramly, and A. Badr, "Beware of the Vulnerability! How Vulnerable are GitHub's Most Popular PHP Applications?," in *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, 2019, pp. 1–7. doi: 10.1109/AICCSA47632.2019.9035265.

[5] I. Medeiros and N. Neves, "Effect of Coding Styles in Detection of Web Application Vulnerabilities," in *2020 16th European Dependable Computing Conference (EDCC)*, 2020, pp. 111–118. doi: 10.1109/EDCC51268.2020.00027.

[6] W3Techs, "Usage Statistics and Market Share of PHP for Websites, May 2023," 2023. https://w3techs.com/technologies/details/pl-php

[7] S. Tenzin, "PHP Framework for Web Application Development," *IARJSET International Advanced Research Journal in Science*, vol. 9, no. 2, 2022, doi: 10.17148/IARJSET.2022.9218.

[8] B. Gautam, J. Tripathi, S. Singh, and M. T. Student, "A Secure Coding Approach For Prevention of SQL Injection Attacks," 2018. [Online]. Available: http://www.ripublication.com

[9] M. L. Siddiq, Md. R. R. Jahin, M. R. Ul Islam, R. Shahriyar, and A. Iqbal, "SQLIFIX: Learning Based Approach to Fix SQL Injection Vulnerabilities in Source Code," in *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021, pp. 354–364. doi: 10.1109/SANER50967.2021.00040.

[10] Synopsys, "Managing Web Application Security With Coverity," 2021.

[11] Sonar, "Code Quality Tool & Secure Analysis with SonarQube," 2023. https://www.sonarsource.com/products/sonarqube/

[12] Synopsys, "Coverity SAST Software | Synopsys," 2023. https://www.synopsys.com/software-integrity/security-testing/static-analysis-sast.html

[13] CodeIgniter, "CodeIgniter User Guide," 2023. https://www.codeigniter.com/userguide3/ (accessed May 31, 2023).

[14] M. Nashaat, K. Ali, and J. Miller, "Detecting Security Vulnerabilities in Object-Oriented PHP Programs," in *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2017, pp. 159–164. doi: 10.1109/SCAM.2017.20.

[15] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: a static analysis tool for detecting Web application vulnerabilities," in *2006 IEEE Symposium on Security and Privacy (S&P'06)*, 2006, pp. 6 pp. – 263. doi: 10.1109/SP.2006.29.

[16] JetBrains, "PHP Programming - The State of Developer Ecosystem in 2021 Infographic," 2023. https://www.jetbrains.com/lp/devecosystem-2021/php/