3rd International Conference on Information Science and Technology Innovation (ICoSTEC)
July 27, 2024, Yogyakarta, Indonesia

Analysis of Security Hotspots in Diploma 3 Information Technology Program's Final Project at Del Institute of Technology

Hernawati Susanti Samosir[1*], Muhammad Anis Al Hilmi[2*], Yulanda Pasaribu[1], Salomo Gemayel Josep Sinambela[1], Vivaldi Adventus Simangunsong[1], Benyamin Sibarani[1], Yen Rylin Hutasoit[1]

Information Technology, Del Institute Of Technology
Sitoluama, Laguboti
Politeknik Negeri Indramayu,
Indramayu, West Java
[1]hernawati@del.ac.id
[2]alhilmi1@gmail.com
[1]pasaribuyulanda94@gmail.com
[1]salomogemayel@gmail.com
[1]vivadvent@gmail.com
[1]benyaminsibarani2406@gmail.com
[1]yenrylin01@gmail.com
*corresponding author

***Abstract*—**
This research investigates security hotspots in Del Institute of Technology students' final project particularly regarding the implementation of the Model-View-Controller (MVC) principles using PHP Laravel. A security hotspot was proposed to find patterns in program code sections that could be hotspots or possible vulnerabilities.

Some of security hotspot examples are misplacement of logic in the view instead of the controller, improper handling of file uploads in the controller, and various other errors. The research generated through this methodology offers insights into commonly overlooked vulnerable points in software development practices. Additionally, the study includes an analysis of 16 students' final projects, where data is collected, and controller and blade files are separated. Subsequently, a plugin accessible at *https://marketplace.visualstudio.com/items?itemName=MuhAnisAlHilmi.laravel-php-codesniffer* is executed. This plugin, a result of previous research, is useful for assisting in secure coding, detecting security indicators, and preventing vulnerabilities.

Eight security hotspots are created to help detect vulnerabilities in the code. Each line of code is then examined to determine its compatibility with each previously established security hotspot. Furthermore, we used a threshold of around 80% in this research based on IBM standards. The results will be evaluated in terms of accuracy and F1 score, allowing for the identification of which security hotspots are most frequently encountered in student final projects. This research is expected to contribute to the improvement of programming standards and security practices in software engineering, providing a better understanding for educators and developers.

Keywords— Security Hotspots, MVC Principles, PHP Laravel, Vulnerabilities, Secure Coding Practices

I. INTRODUCTION

A security hotspot highlights a security-sensitive piece of code that the developer needs to review. Upon review, you will either find there is no threat, or you need to apply a fix to secure the code [1]. Web applications containing high-severity vulnerabilities was 66 percent in 2020 and 62 percent in 2021, significantly more than in 2019 [2].

At Del Institute of Technology, there are still many students who do not understand the use of hotspot security. Starting from research that had been previously developed by Hilmi et al. [3]. From that research it was stated that there was a plugin that was proven very accurate in detecting security hotspots in the program code. Therefore, the author collected program code (PHP Laravel) from the first final projects of D3TI study program's students and then analysed the types of security hotspots that were often carried out by these students. The hope is that using this plugin can help developers find out how secure the web application they are building is and will later become a learning tool for creating program code that applies hotspot security to the application they want to build.

II. RELATED WORK

A.Security Hotspot Rules

In this section, we elaborate on the systematic approach applied in developing detection rules for security vulnerabilities, along with the evaluation metrics utilized to assess the effectiveness of these rules. The detection of security vulnerabilities is governed by eight key rules, as outlined in the set of security vulnerability rules presented in Table 1.

Table 1. Detection rules

No	Rule	Pattern
1	Disallow logic in blade file	detect @if, @elseif, @else
2	Disallow unencrypted id in blade	detect string->id)
3	Detect read-only string in blade	detect validate([detect rules() return [] detect Validator::make() detect controllerName::create()
4	Detect raw SQL syntax in controller and blade	detect readonly string
5	Detect raw SQL syntax in controller and blade	detect DB::raw, selectRaw(", whereRaw(", havingRaw(", orderByRaw(", groupByRaw("
6	Detect unescaped string {!!..!!} in blade	detect {!! … !!}
7	Detect upload file spot in controller	detect T_VARIABLE->file(or T_VARIABLE->image->
8	Disallow weak hash in controller and blade	detect MD5/SHA1

The detection rules, as seen in Table 1, serve as the foundation for identifying security vulnerabilities within the system. These rules encompass various criteria and conditions indicative of potential vulnerabilities, providing a comprehensive framework for proactive security assessment.

C. Metric Evaluation

After detecting security vulnerabilities using the specified rules, the next step involves applying evaluation metrics to measure the system's performance. Evaluation metrics provide a quantitative assessment of the accuracy, sensitivity, precision, and overall effectiveness of the detection system. Four main metrics are used for this purpose:

True Negatives (TN): Occurs when a security hotspot is not detected according to the rule set, and upon manual inspection, it is confirmed that no security hotspot occurred. **False Positives (FP):** Occurs when a security hotspot is detected according to the rule set, but upon manual inspection, it is confirmed that no security hotspot occurred. **False Negatives (FN):** Occurs when a security hotspot is not detected according to the rule set, but upon manual inspection, it is confirmed that a security hotspot did occur. **True Positives (TP):** Occurs when a security hotspot is detected according to the rule set, and upon manual inspection, it is confirmed that a security hotspot did occur.

The collected data, representing the counts of TN, FP, FN, and TP, is used to calculate the following evaluation metrics:

Accuracy: Overall precision of the detection system, calculated as (TN + TP) / (TN + FP + FN + TP).

True Positive Rate (TPR): Also known as Sensitivity or Recall, measures the proportion of actual security hotspots correctly identified, calculated as TP / (TP + FN).

False Positive Rate (FPR): Evaluates the level of false alarms, calculated as FP / (FP + TN).

Precision: Precision metric measures the accuracy of positive security hotspot predictions, calculated as TP / (TP + FP).

Recall: Similar to TPR, emphasizes the system's ability to identify actual positives, calculated as TP / (TP + FN).

F1 Score: Providing a balanced assessment of system performance, calculated as 2 * (Precision * Recall) / (Precision + Recall).

Through the application of these evaluation metrics, researchers can acquire in-depth insights into the strengths and weaknesses of the security vulnerability detection system, thereby facilitating refinement and continuous improvement.

Extensive research has focused on investigating security weaknesses in web applications, including those that depend on PHP-based frameworks. Articles [4] and [5] support the early identification of weaknesses, with [4] specifically highlighting the use of machine learning methods to predict vulnerable elements. Khan's groundbreaking publication in 2022 [6] offers a comprehensive examination of secure software engineering. It outlines essential elements such as measurements, tools, standards, and research areas that are vital for creating robust software applications. Zhao [5] presents a new methodology that combines dynamic and static analysis to improve the efficiency of discovering security vulnerabilities in PHP online applications. These studies emphasize the crucial significance of discovering and reducing security vulnerabilities in software development, specifically in PHP Laravel projects.

Expanding on previous research, particularly the study titled "A Static IDE Plugin to Detect Security Hotspots for Laravel Framework-Based Web Applications" by Hilmi et al. [3], this current study utilizes a widely recognized plugin known for its precision in identifying security weaknesses in program code. This research highlights the effectiveness of the plugin in helping developers identify vulnerabilities and follow secure coding practices, hence strengthening the integrity and resilience of PHP Laravel projects.

This study further investigates security vulnerabilities in the final projects of Del Institute of Technology students, specifically examining the implementation of Model-View-Controller (MVC) principles using PHP Laravel. The research aims to reveal typical vulnerabilities and faults in software development methods by examining 16 student projects and utilizing the previously described plugin. This will bring attention to disregarded aspects of software development. The study aims to assess the frequency and significance of identified security hotspots for project integrity by implementing eight security hotspots and using a threshold of around 80% according to IBM standards. The results of this inquiry are expected to enhance programming standards and strengthen security protocols in software engineering, therefore promoting a more comprehensive knowledge among educators and developers.

Furthermore, the research is consistent with the wider discussion on secure software engineering techniques. Paper [4] and [5] promote the implementation of proactive strategies to detect and address vulnerabilities, emphasizing the crucial importance of ongoing enhancement in software security. Khan's extensive framework [6] offers a clear plan for incorporating security issues into the software development lifecycle, highlighting the importance of a comprehensive approach to safe software engineering. Similarly, Zhao's novel framework [5] emphasizes the significance of utilizing both dynamic and static analysis to improve the efficiency of vulnerability detection techniques in PHP online applications.

To summarize, the compilation of previous studies emphasizes the pressing requirement to tackle security weaknesses in web applications, namely those created utilizing PHP Laravel frameworks. Developers can strengthen the security of software systems by using advanced tools and procedures, like the IDE plugin mentioned in Hilmi's research [3]. These tools help developers find and fix security vulnerabilities, making the software more resistant to possible attackers. The area of secure software engineering is constantly evolving through collaborative efforts and continuous research activities. This progress leads to improved security practices and strong software ecosystems.

III. RESEARCH METHOD

Testing involved gathering 16 Laravel-based projects of students' first year project of Diploma 3 Information Technology at Del Institute of Technology. We are focusing solely on files of controller and blade files, which contain the logic and display components of the program. First of all, we use tokenization for every blade and controller file. From all controller and blade files, we then take 20 files from each that have the most warnings. Subsequently, blank lines of code were eliminated in order to create a dataset for every line of code, classified as either containing or without a security hotspot.

Eight security hotspots are defined to facilitate vulnerability detection, with each line of code scrutinized for compliance with these identified hotspots. The detected results were cross-referenced with manually assigned labels. Confusion matrix calculations were performed, followed by data collection procedures.

Several stages in this research consist of 5 stages that will be described detail below

a. Development of detection rules

We use several sources for detecting rules. Some of them are OWASP Laravel Cheat Sheet, official documentation, and other studies of potential vulnerabilities. We got recommendation: for writing secure program code, we are suggested to focus on appearance, not logic (which should be in the controller file).

b. Detection method with code tokenization

```
T_VARIABLE => $foo
T_WHITESPACE => ·
T_EQUAL => =
T_WHITESPACE => ·
T_CONSTANT_ENCAPSED_STRING => 'bar'
T_SEMICOLON => ;
T_WHITESPACE => \n
```

c. Implementation of detection tools and extension

After compiled the rules then we implemented into program code for the detection process, termed “sniff”. Sniff basic structure consist of a register function and a process function [7].

```php
namespace StandardName\Sniffs\MyCategory;

use PHP_CodeSniffer\Sniffs\Sniff;
use PHP_CodeSniffer\Files\File;

class ExamineTokensSniff implements Sniff {
    public function register() {
        return [T_FUNCTION];
    }

    public function process(File $phpcsFile, $currentTokenIndex)
        $tokens = $phpcsFile->getTokens();
        var_dump($tokens[$currentTokenIndex]);
    }
}
```

Figure 1. Sniff basic structure

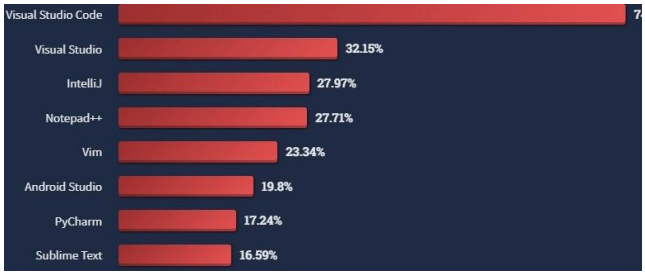


Figure 2. Popular IDE in 2022

The fundamental detection engine, which is built on PHPCS, then has to be linked with an IDE as an extension or plugin; VSCode is the most preferred option based on a 2022 StackOverflow survey [8].

d. Dataset collection and processing

For testing, we use 16 students' final projects were collected in Gitlab repository by students at Del Institute of Technology of Diploma 3 Information Technology Program. After that, the gathered data is sorted, as seen in Figure 3.

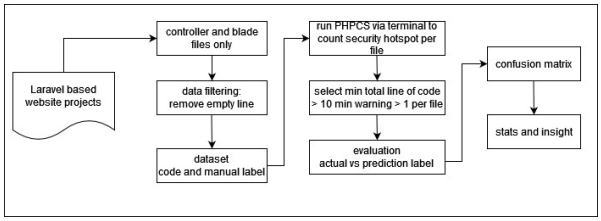


Figure 3. Summary of the development and evaluation stages

We only use controller and blade file because the core part of the program is in these 2 file types, namely logic, and display. Subsequently, blank lines of code were eliminated in order to create a dataset for every line of code, classified as either containing or without a security hotspot.

Then, the created rules are scanned using PHPCS using the CMD/Windows Powershell console. The hardware PHPCS is installed on is a A Zenbook AMD Ryzen 7 5700U with Radeon Graphics.

Eventually finished, the detection outputs are compared with the real labels that we manually created. Following that, data is gathered in order to get insight from the detection results, and calculations are performed using the confusion matrix.

e. Functional test of detection tool

VSCode integration is required for this security hotspot identification tool to help developers throughout the SDLC phase. Should the developer input something that corresponds with the eight established hotspot security detection rules, a warning tone will subsequently sound.

IV. RESULT AND DISCUSSION

In this section, we outline the systematic approach taken in the development of detection rules for security hotspots, along with the evaluation metrics applied to assess the effectiveness of such rules. Security hotspot detection is governed by eight detection rules, as explained before.

From research that have been done before, we conclude there are 8 detection/sniff rules were obtained, shown in Table 1 before.

The eight rules highlight the security hotspot. The rules checked in our projects than we got the result. There are 16 laravel-based website projects have been collected. For the 16 data projects, a data cleaning process was carried out, focusing on detecting controller and blade-type files. We collected a total of 4605 lines of code. From the detection results, TP = 346; FP = 55; TN = 5208; and FN = 0, the values are shown in Table 2.

Table 2. Evaluation Metrics

No	Component	Total number
1	Accuracy	0,99
2	Precision	0,86
3	Recall	1
4	F1 score	0,92
5	TPR	1
6	FPR	0,01

When running, the extension takes 0 for the shortest number of tokens, 40 tokens, and 125 ms for the most extended tokens (25558 tokens). The implementation of detection tool in VSCode in shown in figure 4

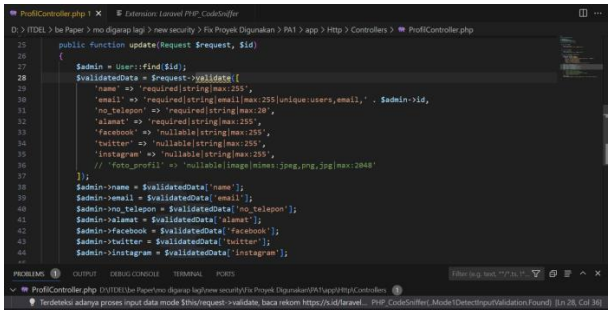
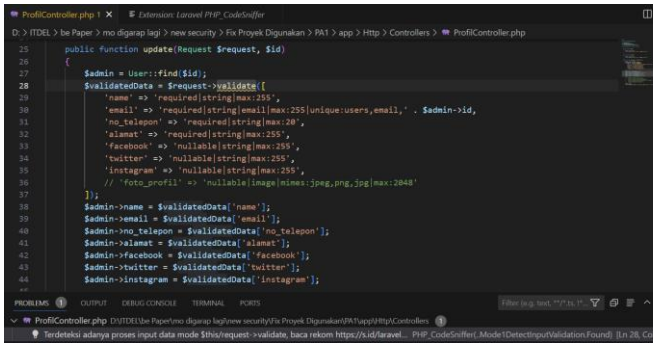
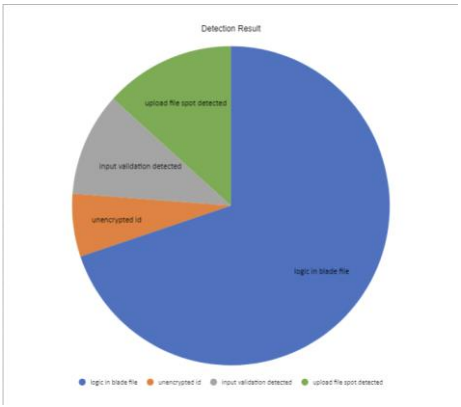


Figure 4. Output example of the extension in analysing code



There are four detection result that we found while we run the plugin, they are the logic placed in the blade file, unencrypted ID, spot validation of user input and upload file spot detected which is shown in Figure 2.

Figure 2. detection result by type



V. CONCLUSIONS :

The Laravel Code Sniffer plugin provides developers warnings directly on the code editor. Developers can fully focus on developing applications without having to spend additional time detecting security hotspots. This Laravel Code Sniffer plugin is proven to be able to detect security hotspots in Laravel-based PHP programs. We did a comparison between the plug-ins and our own analysis. The results of the plug in are not much different from our analysis. When compared with the results of our analysis, the accuracy level of this plug-in reaches 99% with an F1 score of 92,64%. This plug in also works very fast. From the data we collected, the time required is between 0 - 548 ms. This plugin has proven to be very accurate in detecting security hotspots, and can increase the security of a Laravel web application.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of Del Institute of Technology in the development of this work.

REFERENCES

[1] SonarCloud, “Security hotspots,” SonarCloud . Accessed: Feb. 02, 2024. [Online]. Available: https://docs.sonarsource.com/sonarcloud/digging-deeper/security-hotspots/

[2] Positive Technologies, “Threats and vulnerabilities in web applications 2020–2021,” Jun. 2022, Accessed: Feb. 02, 2024. [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020-2021

[3] M. Anis Al Hilmi, Raswa, R. Robiyanto, D. Oranova Siahaan, A. Puspaningrum, and H. Susanti Samosir, “A Static IDE Plugin to Detect Security Hotspot for Laravel Framework Based Web Application,” in *2023 IEEE International Conference on Data and Software Engineering (ICoDSE)*, IEEE, Sep. 2023, pp. 1–6. doi: 10.1109/ICoDSE59534.2023.10291941.

[4] I. Abunadi and M. Alenezi, “An empirical investigation of security vulnerabilities within web applications,” 2016. Accessed: Feb. 02, 2024. [Online]. Available: https://malenezi.github.io/malenezi/pdfs/jucs_22_04_0537_0551.pdf

[5] J. Zhao and R. Gong, “A New Framework of Security Vulnerabilities Detection in PHP Web Application,” in *2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, IEEE, Jul. 2015, pp. 271–276. doi: 10.1109/IMIS.2015.42.

[6] R. A. Khan, S. U. Khan, and M. Ilyas, “Exploring Security Procedures in Secure Software Engineering: A Systematic Mapping Study,” in *The International Conference on Evaluation and Assessment in Software Engineering 2022*, New York, NY, USA: ACM, Jun. 2022, pp. 433–439. doi: 10.1145/3530019.3531336.

[7] Payton Swick, “Creating Sniffs for a PHPCS Standard.” Accessed: Jul. 13, 2023. [Online]. Available: https://payton.codes/2017/12/15/creating-sniffs-for-a-phpcs-standard/

[8] Stackoverflow Insight, “Stackoverflow 2022 Developer Survey.” Accessed: Jul. 13, 2023. [Online]. Available: https://survey.stackoverflow.co/2022/#section-most-popular-technologies-integrated-development-environment