ISSN 2409-2665
Journal of Logistics, Informatics and Service Science
Vol. 11 (2024) No. 6, pp. 266-278
DOI:10.33168/JLISS.2024.0615

Detecting Vulnerabilities in Senayan Library Management System Using Machine Learning

I Komang Sughosa Anantawijaya, Tuga Mauritsius

Information Systems Management Department, Binus Graduate Program, Master of Information Systems Management, Bina Nusantara University

i.anantawijaya@binus.ac.id, tmauritsus@binus.edu

Abstract. Vulnerabilities in the open-source Senayan Library Management System (SLiMS) can impact thousands of libraries globally. However, manual code reviews to uncover undisclosed weaknesses are time-consuming. This study proposes an automated machine learning based static analysis approach to detect vulnerabilities in SLiMS PHP code. A dataset of 22,877 samples with cross-site scripting, SQL injection and other weakness types is used to train classification models. Feature engineering extracted discriminative attributes indicating flaws while vectorization structured the code samples. A Multinomial Naïve Bayes model achieved 97% accuracy in detecting security issues. Three critical vulnerabilities were identified in SLiMS v9.6.1 which were responsibly disclosed and fixed. The findings demonstrate feasibility of applying machine learning to boost open-source security. Further enhancements through custom datasets and deep neural networks can aid real-time vulnerability discovery.

Keywords: Vulnerability Detection, Feature Engineering, Machine Learning, SLiMS, CVE

1. Introduction

Information Systems and Informatics Engineering are two important aspects in maintaining the operational sustainability of an organization or company. Various software products used in various agencies, both open source and closed source, are an integral part of the process. However, when there are reports of vulnerabilities in closed source software products, the provider company will immediately close the security gap and fix the vulnerabilities found. Unlike open-source software, the original code of the product is accessible to anyone. Although it may seem alarming, open source software provides flexibility, transparency, and better security assurance for its users. The use of technology in libraries has seen a significant increase in recent years. Many libraries have started to rely on computer systems to manage collections, circulation, and other operations. Senayan Library Management System (SLiMS) is one example of a system that is widely used by libraries around the world. However, weaknesses or vulnerabilities in this system are a major concern, especially with the increasing dependence on technology. Senayan Library Management System is an open-source software specifically designed to manage and automate library operations. Developed by a team from Senayan Development Community (SDC). Senayan Library Management System provides various features and functions that can help libraries in collection management, lending services, indexing, and general administration (Aini et al., 2022). SLiMS has several advantages that are open source which has complete functionality and high flexibility to be used online or interface. The disadvantage possessed by SLiMS is maintenance that has not been optimized, such as solving problems with the code on the features in SLiMS (Mae et al., 2022). The use of Senayan Library Management System has been widely used by various educational institutions such as private, public schools, and government library offices or private library offices (Aini et al., 2022; Cahyono & Heriyanto, 2013; Mae et al., 2022; Nasrullah et al., 2022). Senayan Library Management System has a critical weakness in 2022, which is fatal and the weakness has vulnerability id of CVE-2022-38292 (*NVD - CVE-2022-38292*, n.d.) where this weakness can make one of the codes in a feature can access a website that allows attackers to attack internal networks that can cause various impacts such as an attack like Server side Request forgery to Remote Code Execution which depends on the code and the configuration enabled. With the vulnerability occur in SLiMS, regardless of companies or organization use standards towards securing software and technology such that they use with or without any security measures to anticipate and mitigate any occurred vulnerability, it does not rule out the possibility of an attacker exploiting the flaw that unreported and unfixed vulnerability that is available in latest SLiMS version. The exploitation that attack the vulnerability code that is unfixed and unreported are Zero day attack (Patidar et al., 2019) which is one of the scariest experience that company or organization can experience including SLiMS. One of the most possible solution to anticipate a Zero day attack is to continuously perform source code review and white box penetration testing to anticipate and to fix vulnerabilities before releasing the open source program or product to every users. The disadvantage of finding and discovering a vulnerability in SLiMS or in any open source program is that source code review is taking a lot of time to read and analyze the code. To solve the problem that source code review for finding unreported and unfixed vulnerabilities will take time, creating a machine learning to detect the vulnerabilities PHP code in SLiMS then test potential vulnerability in the code using white box penetration testing in the code based on the machine learning detection result, it will solve the current problem on source code review that will take a lot of time knowing that machine learning can take pattern or vectorized/encoded dataset that will have similarities to the code that has vulnerabilities that will be detected for vulnerabilities. In order to realize the solution to the problem that source code review for white box penetration testing will take a lot of time, this study will focus on creating a supervised machine learning model to detect vulnerabilities in SLiMS Source code for vulnerabilities. Every vulnerabilities found by the supervised machine learning and reported to the Senayan Development Community (SDC) and National Vulnerability Database(NVD), will be explained using Common Vulnerabilities Scoring System. The specific version that will be targeted in this study is SLIMS 9 Bulian version 9.6.1. The research contribution lies in the novel

automated vulnerability detection using machine learning, enabling proactive fixing of potential security issues within SLiMS. This addition enhances the research's significance by addressing the critical need for efficient vulnerability detection and mitigation in open-source software systems.This study research goals is to find vulnerabilities in SLiMS using machine learning with vulnerable PHP code dataset to classify potential vulnerability that is not reported and fixed in SLiMS

2. Literature Review

2.1. Source code analysis

Source code analysis is the process of programmatically extracting information from source code to summarize, suggest improvements to, or modify the code. It involves tasks that take source code as input, process it, and/or produce source code as output. Source code is a representation of the code quality of a program that is useful for code analysis, testing, and summarization (Molland et al., 2022; Sharma et al., 2021; Ulle, 2022). There are two types of source code analysis, namely dynamic and static. Dynamic analysis is usually done using a software in the form of a Debugger where any running code can be stopped using breakpoints to see the data being processed from user input for inspection. While static analysis only focuses on reading and mapping code in a state where the code is not running or in progress.

2.2. Software vulnerabilities

Vulnerabilities in software are weaknesses or gaps in the system that can be exploited by unauthorized parties. Such vulnerabilities can allow outside attacks that can result in data leakage, manipulation, or takeover of system control. Vulnerabilities that have not been reported and have not been discovered are called zero-day vulnerabilities where these vulnerabilities can be detrimental to their use and can have a fatal impact (Azzedin et al., 2022). Understanding, finding, and reporting vulnerabilities in software is very important in an effort to improve system security (Barr, 2022). Basically, software vulnerabilities are security risks that occur as a result of bugs or defects identified during the operational phase of the program life cycle (Bhatt et al., 2022). These vulnerabilities are potentially exploitable software weaknesses caused by inadequate testing, poor coding techniques, and compatibility or incompatibility issues stemming from the insertion of new code in upgrades.

2.3. Machine learning

Machine learning is a technique that allows computers to learn and improve themselves without having to be explicitly programmed. Machine learning is concerned with developing computer programs that are able to access data and use it to learn on their own (Gupta et al., 2022; Loor et al., 2023). Research conducted by (Gupta et al., 2022) proves that machine learning can provide new solutions and innovations in tracking crime rates in each region. The Research conducted was using two classification algorithms and was examined in evaluating the system based on precision and accuracy measures. Research shows that Maharashtra, Delhi, and Gujarat have the highest crime rates, indicating the need for increased police attention. This research utilizes feature selection strategies such as manual selection and Chi-square to improve classifier accuracy. This approach using machine learning intends to reduce predictable crimes and increase the sense of security in the society by addressing the rising crime rates in various states in India and improving the security of its people.

3. Methodology

3.1. Research Framework

This study will use similarly to CRISP-DM methodology which will be modified in several ways. The

CRISP-DM itself is a method that has been used in many diverse studies related to machine learning and data analysis. The following is the proposed method with CRISP-DM stages with various additions to support this research.

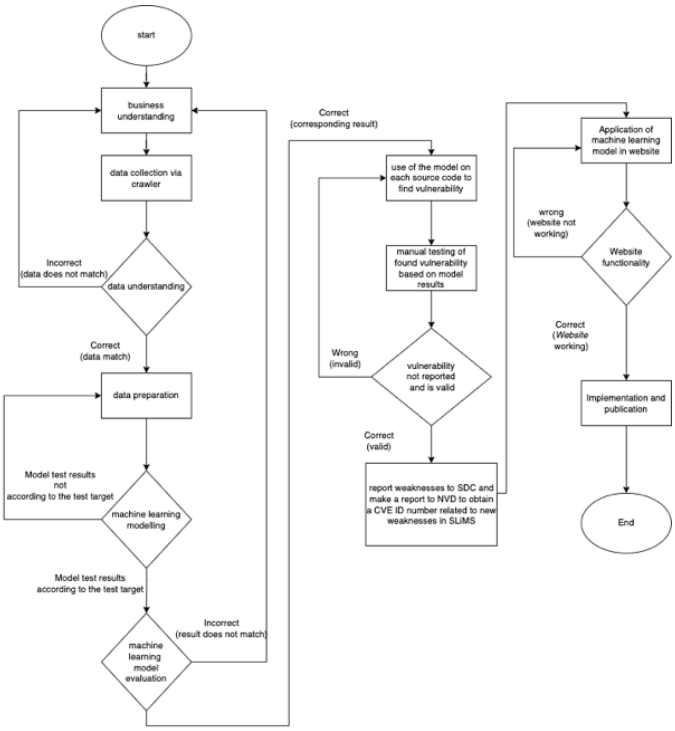


Fig. 1: Research Framework.

3.2. Data Collection via Crawler

After understanding the business and its objectives, this stage will focus on collecting data on the samate.nist.gov or famously known as Software Assurance Reference Dataset (SARD) by National Institute of Standards and Technology (NIST) using Python to collect PHP programming language source code that has vulnerabilities such as Cross Site Scripting, SQL Injection, and Others knowing that most vulnerabilities in previous version of SLiMS are mostly Cross Site Scripting and SQL Injection. The result of the crawler from the data that will be collected is test case id, vulnerable code, and vulnerable type as listed.

3.3. Data Understanding

When all data collection has been collected according to its category by the crawler, namely Cross Site Scripting, SQL Injection, and Others, this stage will focus on data cleaning such as removing code comments, tokenization using module named phply, data exploration, and data verification for the mapping process in the form of graphs or known as Text Mining.

3.4. Data preparation and Machine learning modeling

In this phase, the data that has been collected, understood, and mapped will be prepared such as finding feature extraction from feature engineering and vectorization. After data is processed and prepared is done, the data will be ready to be processed into the model into the supervised machine learning model. The model that will be used is Multinomial Naïve Bayes algorithms in Python using sklearn modules.

We believe by using Multinomial Naïve bayes algorithm with the data type provided can classify and perform better to classify vulnerable code.

3.5. Machine Learning Model Evaluation

The results of machine learning modeling with Multinomial Naïve Bayes will be evaluated based on the accuracy, recall, f1-precision, and confusion matrix. Once evaluation and reverifying the model result is finished, the model will be tested against unlabeled source code with certain vulnerabilities to classify whether the algorithm can detect vulnerabilities in the code or not. After the test and evaluation, the model then can be used in the next phase which is using it for detecting vulnerabilities in SLiMS source code.

3.6. Using of Models for Source Code Security Testing

The results of machine learning modeling and evaluation will be used to classify the vulnerability that exist in each SLiMS source code in its latest version to find weaknesses that exist and have not been reported to the Senayan Developer Community. After the weakness classification is finished, manual security testing based on the model results will be carried out on each source code that has been classified by the model with the aim of obtaining weaknesses that have not been reported and do not yet have a CVE (Common Vulnerabilities and Exposures) number. After the source code security testing process has been carried out and weaknesses that have not been reported and do not have a CVE number are found, each weakness will be reported to the Senayan Developer Community by github and the weaknesses will also be reported to MITRE to process the weaknesses found in order to get a CVE number where the data that will be provided in reporting the weaknesses found in the source code is the source code that has the weakness, the type of weakness, how to exploit the source code, and the SLiMS version used.

3.7. Application of machine learning model in website

The machine learning model that has been created and tested will be exported into a Python programming language module that can be loaded on Python and Flask with a module called pickle to run the machine learning model that has been created. After the module is loaded and tried to run, at this stage the creation of a Flask-based website will be carried out which will have pages in the form of an introduction to Senayan Library Management Systems, graphs of data processing results, data collected, total vulnerability found with the machine learning models, and a feature for public use who want to try the model to identify potential vulnerabilities in the PHP code.

3.8. Website functionality

After the website is created, at this stage, the website will be tested for functionality whether this website can run properly or not to minimize errors that occur when this website is implemented.

3.9. Implementation and publication

In this final stage, the website and machine learning model that has been created will be implemented on the website and virtual private server and published on a domain that will be registered under the name slims-pwned.xyz for public access.

4. Results and Discussions

4.1. Data Cleaning and Data preparation

The results from the custom scraper resulted in 56418 rows but in this study, we decided to use 22877 rows to prevent overfitting and underfitting.

Table 1. Dataset used

Labels	count	Is_null
Potential XSS	7629	0
Potential SQLI	7623	0
Potential Other Vulnerabilities	7625	0

The Dataset consist of combination of PHP and HTML, a PHP code parser named phply was used to tokenize to remove any HTML elements and any Unicode string such as backtick and other types of of symbol that does not parse and correlate to the code element. The elements that was removed from the dataset using phply was WHITESPACE, OPEN_TAG, CLOSE_TAG, INLINE_HTML, COMMENT, RBRACE, LBRACE, SEMI, BACKTICK, DOUBLE_COLON, COLON, AT, and NS_SEPARATOR. After it was removed and tokenized, the dataset token was then joined by adding spaces for bracket and symbols where afterwards data is lemmatize with NLTK wordnet.

Table 2. Data Cleaning

Raw Code	Cleaned Code
<html> <head> <title>Remote File Inclusion</title> </head> <body> Remote File Inclusion This is a link for the scanners: here Content: <?php if (isset($_GET['q'])) { include ($_GET['q'] . '.inc'); } else echo "Default content..."; ?> </body> </html>	if (isset ($_get['q'])) include ($_get['q'] . '.inc') else echo " default content... "

4.2. Feature Engineering and Machine Learning Modelling

After the processing of the dataset, feature engineering to understand the dataset was then performed. The selected feature engineering was intended to extract the total amount or pattern of each vulnerabilities in the dataset. In total there are 10 features that has been extracted from feature engineering

Table 3. Extracted Feature From Feature Engineering

Features	**Description**
concat_sql_count	Count for string concatenation in joined token
concat_no_sql	Count for string concatenation in joined token that has no keyword of “sql”
html_count	Count for keyword of html element in joined token
concat_format_one_sql	Count for string concatenation with single quote by formatting in joined token that has keyword of “sql”
concat_format_two_sql	Count for string concatenation with double quotes by formatting in joined token that has keyword of “sql”
concat_format_one_no_sql	Count for string concatenation with single quote by formatting in joined token that has no keyword of “sql”
concat_format_two_no_sql	Count for string concatenation with double quotes by formatting in joined token that has no keyword of “sql”
func_call	Count For function call in the code that has parameter value inside
var_concat	Count for variable concatenation with “.” symbol
var_concat_two	Count for variable concatenation with “.= ” symbols

Once the featured has been gathered the feature are then plotted to a graph below:

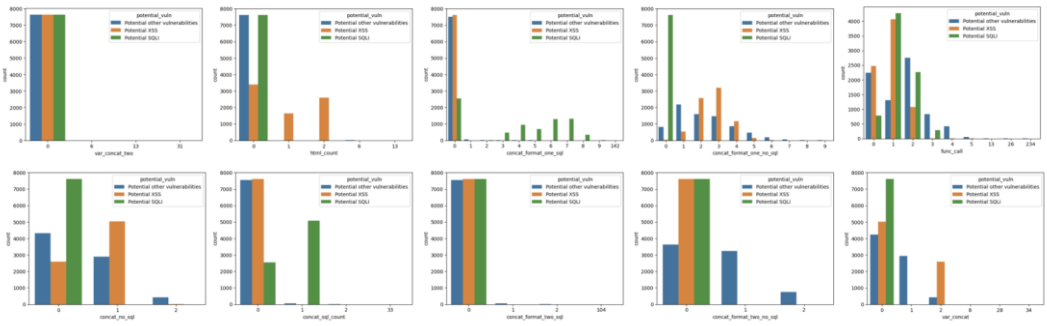


Fig. 2:Feature engineering Result.

Each of the extracted feature from feature engineering represent the most common vulnerabilities occur in the dataset such as ‘html_count’ which mostly are in the ‘potential XSS’ labels. The feature engineering which was performed can be considere a novelty since there is no previous work making this feature engineering for the vulnerability detection in the PHP code. On the other side, the joined token data then are passed to CountVectorizer to vectorize the joined token based on Bag-of-Words(BoW) methods with the parameter set as follows:

Table 4. CountVectorizer Parameters

max_df	min_df	lowercase	n_gram range	token_pattern	stopwords	binary
0.95	0.35	False	(3, 10)	'[a-zA-Z0-9$&+,:;=?@#|<>.^*()%!-]+'	stopwords.words('english')	True

These parameters allow for the creation of a structured representation of text data in the code. This parameter captures word presence as binary values, filtering out common English stop words, while considering n-grams ranging from 3 to 10 words since variable declaring has 3 elements from variable, symbols, and function calling in every code. The max_df and min_df parameters control word frequency thresholds since there are a lot of important symbols in the code that need to be reduced. With these parameters, it makes and enables BoW to classify the vulnerabilities in the code with the features extracted from the vocabulary. From the feature engineering result, it is impossible to use both the vectorized cleaned code with the extracted feature, therefore this model will use two machine learning with the proposed algorithm Multinomial Naïve Bayes where one model will use the extracted feature only and the other one will use count vectorizer

4.3. Machine learning evaluation and result

The modelling for the extracted feature classification resulted in 77% of Accuracy as follows:

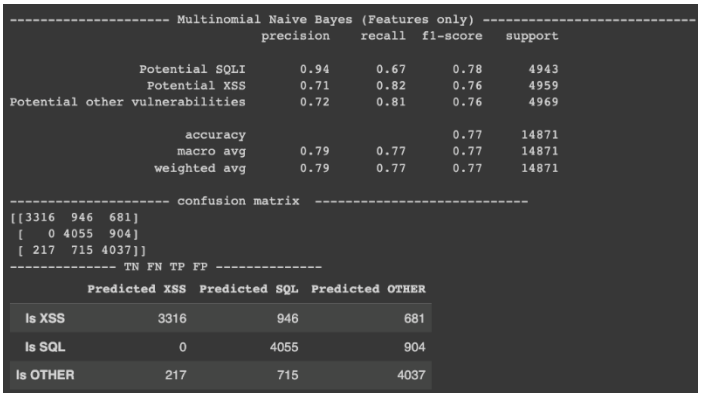
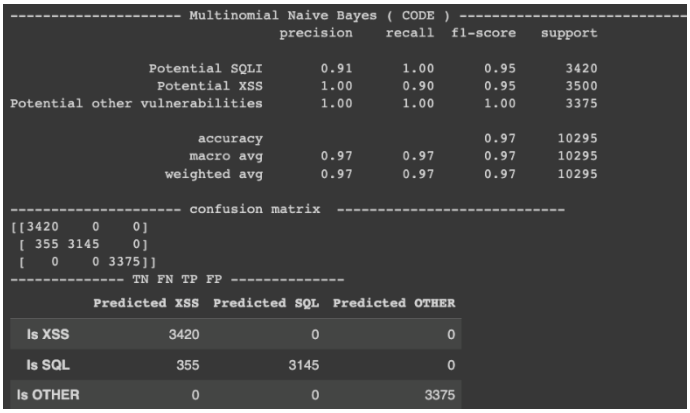


Fig. 3: Extracted Feature dataset for model Accuracy metric result.

While the modelling for the code only resulted in 97% of Accuracy as follows:



---------- Multinomial Naive Bayes (CODE) ----------

	precision	recall	f1-score	support
Potential SQLI	0.91	1.00	0.95	3420
Potential XSS	1.00	0.90	0.95	3500
Potential other vulnerabilities	1.00	1.00	1.00	3375
accuracy			0.97	10295
macro avg	0.97	0.97	0.97	10295
weighted avg	0.97	0.97	0.97	10295

---------- confusion matrix ----------

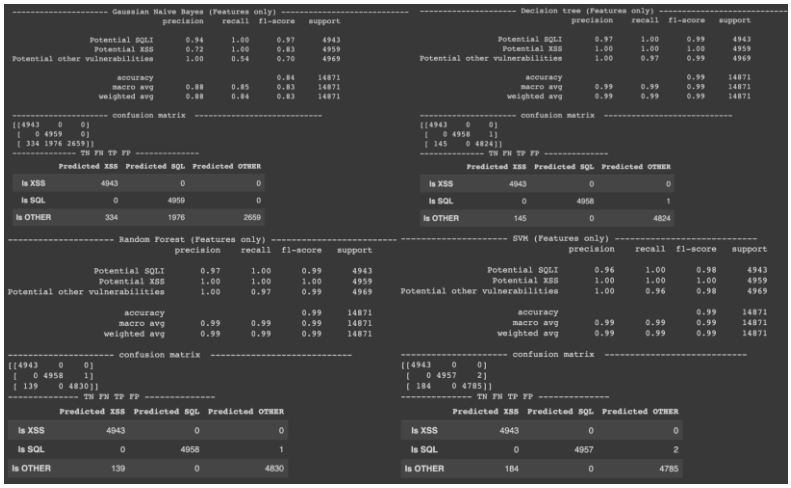
```
[[3420    0    0]
 [ 355 3145    0]
 [   0    0 3375]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	3420	0	0
Is SQL	355	3145	0
Is OTHER	0	0	3375

Fig. 4: Vectorized dataset for model Accuracy metric result.

The result of the two algorithm can be considered better, below are the comparison with another algorithm with SVM, Gaussian Naïve bayes, Decision Tree and Random Forest:



---------- Gaussian Naive Bayes (Features only) ----------

	precision	recall	f1-score	support
Potential SQLI	0.94	1.00	0.97	4943
Potential XSS	0.72	1.00	0.83	4959
Potential other vulnerabilities	1.00	0.54	0.70	4969
accuracy			0.84	14871
macro avg	0.88	0.85	0.83	14871
weighted avg	0.88	0.84	0.83	14871

---------- confusion matrix ----------

```
[[4943    0    0]
 [   0 4959    0]
 [ 334 1976 2659]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	4943	0	0
Is SQL	0	4959	0
Is OTHER	334	1976	2659

---------- Decision tree (Features only) ----------

	precision	recall	f1-score	support
Potential SQLI	0.97	1.00	0.99	4943
Potential XSS	1.00	1.00	1.00	4959
Potential other vulnerabilities	1.00	0.97	0.99	4969
accuracy			0.99	14871
macro avg	0.99	0.99	0.99	14871
weighted avg	0.99	0.99	0.99	14871

---------- confusion matrix ----------

```
[[4943    0    0]
 [   0 4958    1]
 [ 145    0 4824]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	4943	0	0
Is SQL	0	4958	1
Is OTHER	145	0	4824

---------- Random Forest (Features only) ----------

	precision	recall	f1-score	support
Potential SQLI	0.97	1.00	0.99	4943
Potential XSS	1.00	1.00	1.00	4959
Potential other vulnerabilities	1.00	0.97	0.99	4969
accuracy			0.99	14871
macro avg	0.99	0.99	0.99	14871
weighted avg	0.99	0.99	0.99	14871

---------- confusion matrix ----------

```
[[4943    0    0]
 [   0 4958    1]
 [ 139    0 4830]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	4943	0	0
Is SQL	0	4958	1
Is OTHER	139	0	4830

---------- SVM (Features only) ----------

	precision	recall	f1-score	support
Potential SQLI	0.96	1.00	0.98	4943
Potential XSS	1.00	1.00	1.00	4959
Potential other vulnerabilities	1.00	0.96	0.98	4969
accuracy			0.99	14871
macro avg	0.99	0.99	0.99	14871
weighted avg	0.99	0.99	0.99	14871

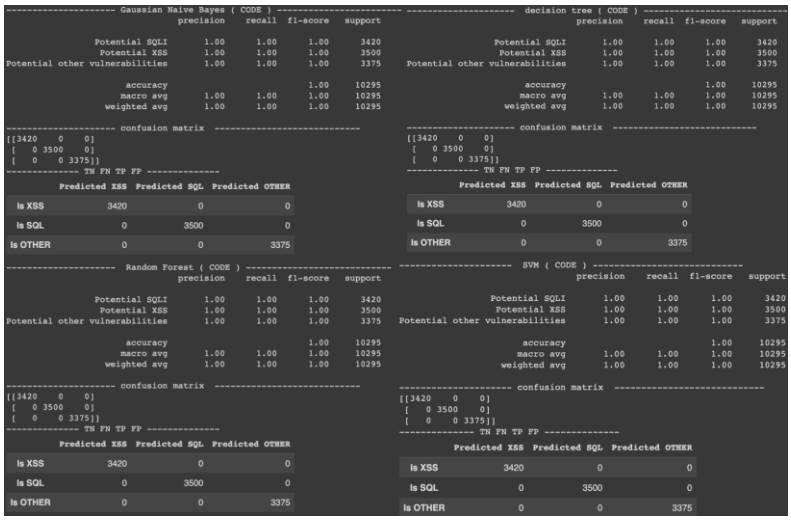
---------- confusion matrix ----------

```
[[4943    0    0]
 [   0 4957    2]
 [ 184    0 4785]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	4943	0	0
Is SQL	0	4957	2
Is OTHER	184	0	4785

Fig. 5: Extracted feature dataset for model Accuracy metric result comparison.



---------- Gaussian Naive Bayes (CODE) ----------

	precision	recall	f1-score	support
Potential SQLI	1.00	1.00	1.00	3420
Potential XSS	1.00	1.00	1.00	3500
Potential other vulnerabilities	1.00	1.00	1.00	3375
accuracy			1.00	10295
macro avg	1.00	1.00	1.00	10295
weighted avg	1.00	1.00	1.00	10295

---------- confusion matrix ----------

```
[[3420    0    0]
 [   0 3500    0]
 [   0    0 3375]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	3420	0	0
Is SQL	0	3500	0
Is OTHER	0	0	3375

---------- decision tree (CODE) ----------

	precision	recall	f1-score	support
Potential SQLI	1.00	1.00	1.00	3420
Potential XSS	1.00	1.00	1.00	3500
Potential other vulnerabilities	1.00	1.00	1.00	3375
accuracy			1.00	10295
macro avg	1.00	1.00	1.00	10295
weighted avg	1.00	1.00	1.00	10295

---------- confusion matrix ----------

```
[[3420    0    0]
 [   0 3500    0]
 [   0    0 3375]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	3420	0	0
Is SQL	0	3500	0
Is OTHER	0	0	3375

---------- Random Forest (CODE) ----------

	precision	recall	f1-score	support
Potential SQLI	1.00	1.00	1.00	3420
Potential XSS	1.00	1.00	1.00	3500
Potential other vulnerabilities	1.00	1.00	1.00	3375
accuracy			1.00	10295
macro avg	1.00	1.00	1.00	10295
weighted avg	1.00	1.00	1.00	10295

---------- confusion matrix ----------

```
[[3420    0    0]
 [   0 3500    0]
 [   0    0 3375]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	3420	0	0
Is SQL	0	3500	0
Is OTHER	0	0	3375

---------- SVM (CODE) ----------

	precision	recall	f1-score	support
Potential SQLI	1.00	1.00	1.00	3420
Potential XSS	1.00	1.00	1.00	3500
Potential other vulnerabilities	1.00	1.00	1.00	3375
accuracy			1.00	10295
macro avg	1.00	1.00	1.00	10295
weighted avg	1.00	1.00	1.00	10295

---------- confusion matrix ----------

```
[[3420    0    0]
 [   0 3500    0]
 [   0    0 3375]]
```

---------- TN FN TP FP ----------

	Predicted XSS	Predicted SQL	Predicted OTHER
Is XSS	3420	0	0
Is SQL	0	3500	0
Is OTHER	0	0	3375

Fig. 6: Vectorized dataset for model Accuracy metric result comparison.

Based on SVM, Gaussian Naïve bayes, Decision Tree and Random Forest algorithm with datased

of vectorized code and feature engineering, it shows that the model accuracy and confusin matrix was not balance, therefore the Multinomial Naïve Bayes algorithm was then used to detect every PHP code in SLiMS and it has successfully identified three vulnerable codes that has been reported.

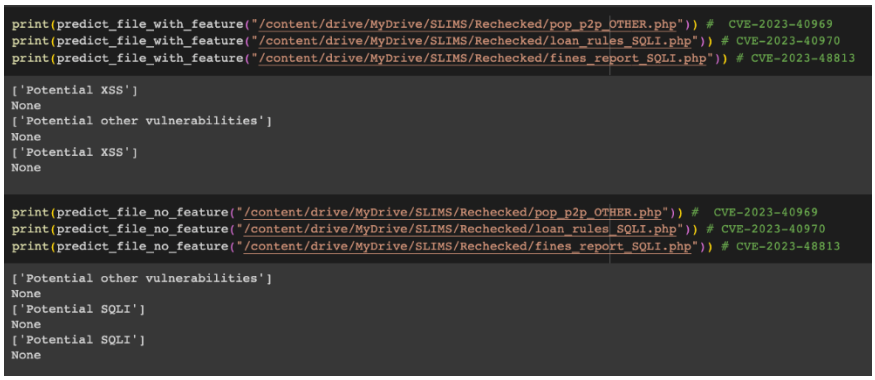


Fig. 7: Identified Vulnerable code with machine learning.

4.3.1 CVE-2023-40969 Analysis

The CVE-2023-40969 vulnerability was at the code of "admin/modules/bibliography/pop_p2p.php". The model which used vectorized code classified this code as "potential other vulnerabilities" where the model which use feature only classified this code as "potential XSS". If we take a look at the code, the main issue of the code can be spotted from line 35 to 37.

Fig. 8:Vulnerable line of code pop_p2p.php.

The input of parameter uri is passed to an unsafe string concatenation where then the string is used at the function XML parser named modsXMLsenayan. The impact of this vulnerability is the attacker can create a malicious XML to print out a malicious javascript to steal admin cookie where then the attacker must host the XML. To launch the attack, all attacker needs to do is having a full path of the vulnerable code in a link with a malicious link on the uri parameter which is similar to reflected XSS attack and vulnerability. The two models that detect this vulnerability are correct and relates to the code and this Vulnerability has been reported to Senayan Developers Community and National Vulnerability Database(NVD). The CVSS metric from NVD were shown as below:

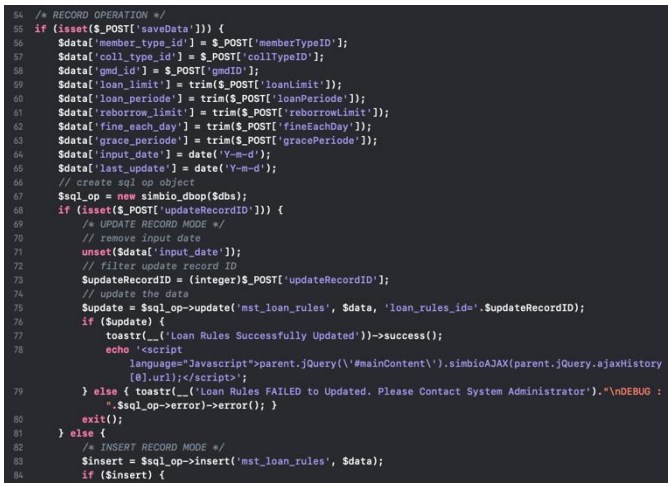
Table 5. CVSS Score and scale of CVE-2023-40969

CVSS Metrics	**Severity**
CVSS:3.1/AV:N/AC:L/ PR:N/UI:R/S:C/C:L/I:L/ A:N	6.1 (Medium)

4.3.2 CVE-2023-40970 Analysis

The CVE-2023-40979 vulnerability was at the code of "admin/modules/circulation/loan_rules.php". The model which used vectorized code classified this code as "potential SQLI" where the model which used features only classified this code as "Potential other vulnerabilities". If we take a look at the code,

the main issue of the code is difficult to spot on from line 55 to 84 because there is a database querying function being called at line 75 and 83:



```
54  /* RECORD OPERATION */
55  if (isset($_POST['saveData'])) {
56      $data['member_type_id'] = $_POST['memberTypeID'];
57      $data['coll_type_id'] = $_POST['collTypeID'];
58      $data['gmd_id'] = $_POST['gmdID'];
59      $data['loan_limit'] = trim($_POST['loanLimit']);
60      $data['loan_periode'] = trim($_POST['loanPeriode']);
61      $data['reborrow_limit'] = trim($_POST['reborrowLimit']);
62      $data['fine_each_day'] = trim($_POST['fineEachDay']);
63      $data['grace_periode'] = trim($_POST['gracePeriode']);
64      $data['input_date'] = date('Y-m-d');
65      $data['last_update'] = date('Y-m-d');
66      // create sql op object
67      $sql_op = new simbio_dbop($dbs);
68      if (isset($_POST['updateRecordID'])) {
69          /* UPDATE RECORD MODE */
70          // remove input date
71          unset($data['input_date']);
72          // filter update record ID
73          $updateRecordID = (integer)$_POST['updateRecordID'];
74          // update the data
75          $update = $sql_op->update('mst_loan_rules', $data, 'loan_rules_id='.$updateRecordID);
76          if ($update) {
77              toastr(__('Loan Rules Successfully Updated'))->success();
78              echo '<script
                    language="Javascript">parent.jQuery(\'#mainContent\').simbioAJAX(parent.jQuery.ajaxHistory
                    [0].url);</script>';
79          } else { toastr(__('Loan Rules FAILED to Updated. Please Contact System Administrator')."\nDEBUG :
                ".$sql_op->error)->error(); }
80          exit();
81      } else {
82          /* INSERT RECORD MODE */
83          $insert = $sql_op->insert('mst_loan_rules', $data);
84          if ($insert) {
```

Fig. 9:Vulnerable line of code loan_rules.php.

The vulnerability lies behind the unsafe parameter parsing which is gmdID, memberTypeID, and collTypeID where then it passed to a SQL query execution function simbio_dbop with variable sql_op which make the code is vulnerable to SQL Injection attack. To exploit this vulnerability, the attacker must have admin access or a user that has an access to the circulation module and make a custom request to the loan rules feature with a malicious SQL query to the unsanitized parameter. The model which use the vectorized code is correct on identifying the vulnerability in the code and this Vulnerability has been reported to Senayan Developers Community and National Vulnerability Database(NVD). The CVSS metric from NVD were shown as below:

Table 6. CVSS Score and scale of CVE-2023-409770

CVSS Metrics	**Severity**
CVSS:3.1/AV:N/AC:L/ PR:L/UI:N/S:U/C:H/I:H /A:H	8.8 (High)

4.3.3 CVE-2023-48813 Analysis

The CVE-2023-48813 vulnerability was at the code of "admin/modules/reporting/customs/fines_report.php". The model which used vectorized code classified this code as "potential SQLI" where the model which used features only classified this code as "Potential XSS". If we take a look at the code, the main issue of the code is from line 120 to 130 because there is an unsanitized parameter which concat with SQL Query.

```
<?php
} else {
    ob_start();
    $fines_data = array();
    // year
    $selected_year = date('Y');
    if (isset($_GET['year']) AND !empty($_GET['year'])) {
        $selected_year = (integer)$_GET['year'];
    }
    // month
    $selected_month = date('m');
    if (isset($_GET['month']) AND !empty($_GET['month'])) {
        $selected_month = $_GET['month'];
    }

    // query fines data to database
    // echo "SELECT SUBSTRING(`fines_date`, -2) AS `mdate`, SUM(debet) AS `dtotal`, `fines_id` FROM `fines` WHERE `fines_date` LIKE '$selected_year-$selected_month%' GROUP BY `fin
    $_fines_q = $dbs->query("SELECT SUBSTRING(`fines_date`, -2) AS `mdate`, SUM(debet) AS `dtotal` FROM `fines` WHERE `fines_date` LIKE '$selected_year-$selected_month%' GROUP BY
    while ($_fines_d = $_fines_q->fetch_row()) {
        $date = (integer)preg_replace('@^0+@i', '',$_fines_d[0]);
        $fines_data[$date] = '<div class="data"><a href="'.AWB.'modules/reporting/customs/member_fines_list.php?reportView=true&singleDate=true&finesDate='.$selected_year.'-'.$sel
    }

    // generate calendar
    $output = simbio_date::generateCalendar($selected_year, $selected_month, $fines_data);

    // print out
    echo '<div class="mb-2">'.__('Fines count report for').' <strong>'.$months[$selected_month].', '.$selected_year.'</strong> <a class="s-btn btn btn-default printReport" onclick
    echo $output;

    $content = ob_get_clean();
    // include the page template
    require SB.'/admin/'.$sysconf['admin_template']['dir'].'/notemplate_page_tpl.php';
}
```

Fig. 10:Vulnerable line of code loan_rules.php.

The vulnerability lies behind the unsafe parameter parsing which is month where then it passed to a SQL query execution function at line 126 which make the code is vulnerable to SQL Injection attack. To exploit this vulnerability, the attacker must have admin access or a user that has an access to the reporting module and make a custom request to the fines_report feature with a malicious SQL query to the unsanitized parameter. The model which use the vectorized code is correct on identifying the vulnerability in the code and this Vulnerability has been reported to Senayan Developers Community and National Vulnerability Database(NVD). The CVSS metric from NVD were shown as below:

Table 7. CVSS Score and scale of CVE-2023-48813

CVSS Metrics	**Severity**
CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H	8.8 (High)

5. Conclusion

An open-source software product that has unreported and unfixed vulnerabilities can lead to many problems for many users that are currently using it. By using supervised machine learning model for static code analysis towards source code security, can help identify a vulnerability in the code faster than performing manual analysis. However, automatic vulnerability identification with checking for false positivity is very crucial in order to make patches or publishing the source code or the product. This research has identified three vulnerabilities that were tested and reported that make SLiMS securer than before. However, this research limitation is determining invulnerable code since the label of the data is mostly about the vulnerabilities. Moreover, the code dataset that were scraped showed many incorrect and irrelevant from the labels such as SQL Injection vulnerabilities which also has XSS vulnerabilities which in future work, a dataset classification or generating new data is needed. Combining the two model with different data between vectorized code and extracted feature from the code is a challenge which might bring more better and synchronized output knowing the current model is not combined yet showed different result of vulnerability detection. The vectorized code uses bag-of-words which has many feature and vocabulary that is too much to process for any algorithm beside Multinomial Naïve Bayes. Therefore, by using deep learning and word embedding combined with the extracted feature and vectorized code might bring the best result in the future work.

Reference

Aini, Q., Rukmana, E. N., & Rohman, A. S. (2022). Penerapan Aplikasi Senayan Library Management System (SLIMS) dalam Pengelolaan Bahan Pustaka di Perpustakaan Sekolah. *Journal2.Um.Ac.Id*. http://journal2.um.ac.id/index.php/bibliotika/article/view/26239

Azzedin, F., Suwad, H., CONTINUA, M. R.-M. &, & 2022, undefined. (2022). An Asset-Based Approach to Mitigate Zero-Day Ransomware Attacks. *Cdn.Techscience.Cn*. https://doi.org/10.32604/cmc.2022.028646

Barr, J. R. (2022). *On FICO-Like Vulnerability Rating of Source Code*. https://doi.org/10.13140/RG.2.2.19998.41285

Bhatt, N., Kaur, J., Anand, A., Alhazmi, O., Alhazmi, O. H., & Author, C. (2022). Selecting Best Software Vulnerability Scanner Using Intuitionistic Fuzzy Set TOPSIS. *Researchgate.Net*. https://doi.org/10.32604/cmc.2022.026554

Cahyono, J. E. (Jefri), & Heriyanto, H. (Heriyanto). (2013). Analisis Pemanfaatan Senayan Library Management System (Slims) Di Kantor Perpustakaan Dan Arsip Daerah Kota Salatiga. *Jurnal Ilmu Perpustakaan*, *2*(3), 139–152. https://www.neliti.com/publications/101353/

Gupta, V. K., Shukla, S. K., Rawat, R. S., & Kumar Gupta, V. (2022). Crime tracking system and people's safety in India using machine learning approaches. *Researchgate.Net*, *2*(1). https://www.researchgate.net/profile/Vishan-Gupta-2/publication/358287007_Crime_Tracking_System_and_People's_Safety_in_India_using_Machine_Learning_Approaches/links/61fe2491870587329e917964/Crime-Tracking-System-and-Peoples-Safety-in-India-using-Machine-Learning-Approaches.pdf

Loor, C., Morocho, K., Politécnica, M. H.-R., & 2023, undefined. (2023). Using Data Mining Techniques for the Detection of SQL Injection Attacks on Database Systems. *Revistapolitecnica.Epn.Edu.Ec*. https://doi.org/10.33333/rp.vol51n2.02

Mae, S., Fernandez, B., Mae, S., & Pagayonan, S. (2022). Library Management Information System for Public High Schools: An Impact Assessment. *Ijres.Org*, *10*(5), 381–384. https://www.ijres.org/papers/Volume-10/Issue-5/Ser-15/1005381384.pdf

Molland, T., Nesbakken, A., og, J. L.-N. I. for forskning, & 2022, undefined. (2022). Automatic Detection and Fixing of Java XXE Vulnerabilities Using Static Source Code Analysis and Instance Tracking. *Ojs.Bibsys.No*. https://ojs.bibsys.no/index.php/NIK/article/view/1018

Nasrullah, N., Tawakkal, & Nursalsabila. (2022). Analisis Penggunaan Senayan Library Management System (Slims) di Perpustakaan Madrasah Aliyah Negeri 1 Majene Provinsi Sulawesi Barat. *Literatify : Trends in Library Developments*, *3*(2), 99–111. https://doi.org/10.24252/LITERATIFY.V3I2.31894

NVD - CVE-2022-38292. (n.d.). Retrieved October 3, 2023, from https://nvd.nist.gov/vuln/detail/CVE-2022-38292

Patidar, P., Analytical, H. K.-I. J. of R. and, & 2019, undefined. (2019). Zero-day attack detection using machine learning techniques. *Ijrar.Org*. https://www.ijrar.org/papers/IJRAR19J1648.pdf

Sharma, T., Kechagia, M., Georgiou, S., … R. T. preprint arXiv, & 2021, undefined. (2021). A survey on machine learning techniques for source code analysis. *Arxiv.Org*. https://arxiv.org/abs/2110.09610

Ulle, N. (2022). *Source Code Analysis and Type Inference for R*. https://search.proquest.com/openview/d025912f423eab2a6a754aea59cd794a/1?pq-origsite=gscholar&cbl=18750&diss=y