APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 1 (2025) 539-559
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

Secure Coding Review System based on Static Analysis Approach

Chin Yuann Lin[1], Isredza Rahmi A Hamid[1]*

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: rahmi@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2025.06.01.030

Article Info

Received: 20 July 2024
Accepted: 19 June 2025
Available online: 30 June 2025

Keywords

Secure Coding, Review System, Static Analysis, Machine Learning Approach, Object Oriented Analysis and Design

Abstract

The Secure Coding Review System allow users to examine source code for accessing the security vulnerabilities. The current secure coding review tools focuses on using advanced malware injection. Yet, the current systems do not provide detailed feedback with suggestions, user friendly interface and does not have community platform which makes it lack human touch expertise. The system focuses on performing static analysis on source code which concentrates on security vulnerabilities like input validation, authentication, authorization, and user management. The Secure Coding Review system will generate detailed reports highlighting identified issues along with feedback, recommendation and suggestion. Programming languages used to develop the system are HTML, CSS and JavaScript for frontend, PHP for backend and Python Programming language to train the Random Forest Model. The methodology will follow an Object-Oriented Analysis and Design framework. The Secure Coding Review System is able to review uploaded C source code on accessing security vulnerability and provide a detailed report, suggestions and feedback to the users and a community platform is provided. Various test cases and user acceptance testing were conducted successfully and had satisfied all criteria. Machine Learning testing and validation had also been carried out with satisfactory result with an accuracy above 80 percent. The system offers the advantage of identifying security vulnerabilities using the Random Forest Algorithm providing a detailed report and supplying a community platform for programmers to engage with one another. Possible future improvements might involve broadening the datasets on targeted vulnerabilities to increase predictability, integrating more security vulnerability assessment classifications, including dynamic review, and hosting the system online.

1. Introduction

A secure code review is a system which examines the program's source code by identifying any security vulnerabilities in the program. As the invention of new technologies becomes more complex, the combination of human expertise and technology support is required in security code reviews as tools may be able to carry out the task of reviewing abundance of source codes yet human verification is needed to determine the real issues and calculate the risk [1]. Dynamic analysis, which evaluates code during execution, often struggles to pinpoint exact locations of security vulnerabilities, making fixes time-consuming [2]. This highlights the need for more

effective methods. Static analysis identifies vulnerabilities early in the SDLC, making it cost-effective and timesaving [19]. Therefore, by integrating static analysis tools early in program development enhances security and streamlines vulnerability remediation [16]. Besides that, the existing systems are mostly tool-based and Command Line Interface (CLI) which makes it less convenient for users to access. The absence of a community platform among existing systems also causes the lack of human touch of expertise.

To overcome the issues of the existing secure code reviewing tools, the Secure Coding Review System is to compensate the lack of tools or systems implementing static secure code analysis, provide a GUI which provides easier access to the user and provide more clear dan detailed feedback, recommendation and fixes and integrates a community platform, which serves to promote opportunities among developers to provide feedback and code verification.

This project is inspired by the increasing need for reliable and secure software nowadays to ensure the safety of existing software cyber threats are becoming more advanced. This issue makes it crucial to undergo vulnerabilities detection earlier in the software development lifecycle. Besides that, the absence of a community platform causes the existing tools to lack the human touch of expertise and feedback that a community of experienced developers can offer. The aim of developing the system is to create a static code analysis that checks source code automatically for security vulnerabilities but also provide a community platform to offer human verification to verify and strengthen code security. Open Worldwide Application Security Project (OWASP) Code Review Guide stated, only an expert can evaluate the real risk and understand the significance of a code bug and vulnerabilities [1]. The objectives of this project are:

- to design Secure Coding Review System based on object-oriented approach
- to develop Secure Coding Review System with added community features
- to implement alpha and beta testing on the Secure Coding Review System to the target user.

The Secure Coding Review System will use two approaches to review security vulnerabilities. First, by implementing a Random Forest algorithm to learn the concept of secure and insecure source codes to examine and classify the code for vulnerabilities. Then, the secure coding review system will be used to automatically identify security vulnerabilities on source code while providing detailed reports containing recommendation, feedback, and code fixes. These automated checks are known as static code analysis which examines a code without having to execute the code [2]. Second, a user-friendly community platform will be developed additionally with the secure coding review system where users can share their code and provide a collaborative environment where community members can provide verification, insights, feedback, and suggestions. The system developed using python programming language and Random Forest machine learning algorithm including datasets of secure and insecure coding of input validation, authentication and user management and authorization is targeted towards anyone who has a C source code. The system will consist of a user module including a code review module, report module and community module. Additionally, admin module to manage the users and community.

The system will focus on reviewing the security aspects of C programming source code. By combining automated static code analysis with the human verification from a community platform, the system aims to provide a better approach of security code review as it addresses both automated and manual security code review, giving developers a stronger defence against potential threats.

The rest of the paper is organized as follows: Section 2 discusses about a literature review on secure coding principles, concepts of existing systems and comparison of the existing systems with systems Section 3 describes about the methodology used on the secure coding review system which would discuss the Software Development Cycle used and the process of carrying out the system. Section 4 explains the system analysis and design of the system. Finally, the last section concludes the current work and highlights the future contribution.

2. Literature Review

This section explains about the literature review related to existing secure coding review system, Common Weakness Enumeration and comparison between related work and Secure Code Review System.

2.1 Secure Coding Review System

Secure coding review's main objective is to examine the program's source code by identifying any security vulnerabilities in the program. The current secure code review system emphasizes dynamic secure coding review which addresses the nature of a code by executing the code [2]. This would prolong the life cycle for each system development if only dynamic review were done at the end of the testing phase. Besides that, the existing systems are mostly tool-based and Command Line Interface (CLI) which makes it less convenient for users to access. The absence of a community platform among existing systems also causes the lack of human touch of expertise. Therefore, combination of human expertise and technology support is required in security code reviews as tools may be able to carry out the task of reviewing abundance of source codes yet human verification is needed to determine the real issues and calculate the risk. A company can ensure that application

developers are complying with secure development practices by using secure code review strategies to ensure that the application has been created to be "self-defending" [1].

2.2 Common Weakness Enumeration (CWE)

The Common Weakness Enumeration is a catalogue of software and hardware vulnerabilities created by the community. A "weakness" is a condition in a software, firmware, hardware, or service component that, under conditions, may contribute to the emergence of vulnerabilities. The CWE List and its classification taxonomy identify and describe CWE problems. Knowing the flaws that lead to vulnerabilities allows software developers, hardware designers, and security architects to eliminate them before deployment, which is easier and less expensive [3]. According to [4], Table 1 lists the related common weakness enumeration which are associated with the vulnerability of its datasets which will be utilized and classified in the system. The description is written according to the Top 25 Common Weakness Enumeration List [5].

Table 1 *Common Weakness Enumeration (CWE)*

Common Vulnerabilities Enumeration	Description
CWE-787	Out-of-bounds Write
CWE-20	Improper Input Validation
CWE-120	Buffer Copy Without Checking Size of Input
CWE-125	Out-of-bounds Read
CWE-416	Use After Free
CWE-476	NULL Pointer Dereference
CWE-190	Integer Overflow or Wraparound
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')

The system's security strategy priorities input validation, memory management, and concurrency control, which corresponds to Common Weakness Enumeration (CWE) categories. It reduces the danger of injection attacks and buffer overflows by thoroughly validating inputs (CWE-20, CWE-120, CWE-119), guaranteeing that only expected input values within set ranges are permitted avoiding malicious data from residing in the database and causing the system to malfunction [17]. Effective memory management (CWE-787, CWE-125, CWE-416, CWE-476, CWE-190) reduces vulnerabilities such as out-of-bound accesses and memory leaks by implementing adequate allocation, deallocation, and bounds checking techniques. Robust concurrency control (CWE-362) techniques, such as locks and semaphores, protect against race circumstances, ensuring reliable and secure operation in multi-threaded or distributed contexts. Addressing these CWE categories strengthens the system's security posture, lowering the likelihood of exploitation from common vulnerabilities and improving software dependability and resilience.

2.3 Study on Existing Secure Coding Review Systems

This section explains the studies that have been done related to the existing secure coding review system such as Flawfinder, CPPcheck and SPLINT.

2.3.1 Flawfinder

Flawfinder [6] is a Command Line Interface (CLI) program which reviews the C/C++ source code and reports the potential security vulnerabilities sorted by its risk level using static code analysis developed by David A. Wheeler using Python. The program needs to be downloaded and used Python to run the program on windows. The advantage of this program is that it can find and remove potential vulnerabilities in a short time before a program is publicly released. Flawfinder utilizes pattern matching to find security flaws by assigning security risk level to identified vulnerability according to the potential impact. After reviewing the source code, Flawfinder shows an analysis summary containing hits, lines analysed, hits level, minimum risk level and the possible vulnerabilities in the existing source code. Flawfinder aids to secure software in prevention of common programming errors that could be exploited by malicious actors if it were to be released before the scanning.

2.3.2 CPPcheck

CPPcheck [7] is a Graphical User Interface (GUI) program which reviews the C/C++ source code and identifies potential security vulnerabilities, bugs issues and coding style issues. It is an open-source project which uses a variety of checks and heuristics to statically review the code without having to compile it. It spots the codes for common mistakes such as memory leaks, null point dereferences when an expected valid pointer is defined as Null, undefined behaviour and security issues. The program is needed to be downloaded and can be viewed in windows with a simple graphic user interface. The advantage of this program is that it able to give a simple feedback and recommendation if the identified issues which aids developer in addressing to the potential problem efficiently. The output of CPPcheck after reviewing the source code. It shows an analysis log which states the possible vulnerabilities at its existing source code.

2.3.3 SPLINT

SPLINT [8] is a Command Line Interface (CLI) tool which reviews the C source code on any UNIX system with a standard C compiler and it is not convenient to use SPLINT in a non-UNIX platform. It is also an open-sourced project which identifies the potential security vulnerabilities and coding errors. The advantage of this program is it uses sophisticated checks and annotations to analyse the source code and shows the possible memory leaks, data misuse and other potential security flaws that might lead to runtime errors or integrity compromise issue. SPLINT can analyse complex code structures and provide feedback. After reviewing the C source code, it shows the analysis summary at the terminal of the compiler.

2.4 Comparison Between Related Work and Secure Coding Review System

Table 2 shows the comparison between existing secure coding review systems Flawfinder, CPPcheck and SPLINT with the Secure Coding Review System. The characteristics, functions, and features such as programming language, method used to analyse the security vulnerabilities, community platform, report generation, code improvements and user interface of the existing system and the system are compared.

Table 2 *Comparison Between Related Work and Secure Coding Review System*

Features	Flawfinder [6]	CPPCheck [7]	SPLINT [8]	Secure Coding Review System (Proposed System)
Programming Language	Python	C++	C	Python, HTML, CSS, PHP, JavaScript
Program Code Check	C/C++	C/C++	C	C
Method used to analyse the security vulnerabilities	Pattern Matching	Pattern Matching	Type Checking	ML (Random Forest Algorithm)
Community Platform	No	No	No	Yes
Generate Report	No	No	No	Yes
Code Improvements	Give feedback	Give feedback and recommendation	Give feedback	Provide suggestions
User Interface	CLI	GUI	CLI	GUI
Vulnerabilities Assessment	Common Security Vulnerabilities such as Buffer overflows.	Memory Leaks, Null Point Dereferencing and Uninitialized variables	Memory management, type safety, Null Point Dereferencing and Uninitialized variables	Input Validation, Memory Management, and Concurrency Control

The programming language used to develop the CPPcheck is C++, SPLINT uses C while Flawfinder and the system use Python. All tools check C programming language but Flawfinder and CPPcheck check C++ programming language. Flawfinder and CPPcheck used Pattern Matching to identify all flaws, while SPLINT uses type checking. Our system utilizes the Random Forest Algorithm to analyse and predict the outcome of the security vulnerability of the existing source code and provide recommendations. Moreover, they did not provide report generation. In contrast, the system will offer these features. All current systems provide coding feedback. Only CPPcheck gives recommendations. Our secure coding review system provides feedback, recommendations,

and improvements. In addition, the user interface for Flawfinder and SPLINT are CLI, while CPPCheck and the System can be views in GUI. Vulnerabilities assessment provided by Flawfinder is Buffer overflows, CPPcheck review Memory Leaks, Null Point Dereferencing and Uninitialized variables, SPLINT have similarity with CPPcheck with additionally type safety. The system will be accessing Input Validation, Memory Management, and Concurrency Control which are security vulnerabilities such as buffer overflow on scanf, dereference failure, array bounds violated, division by zero.

3. Methodology

This section explains the methodology used in the project which is the Object-oriented Analysis and Design (OOAD). This methodology is chosen because it uses the object-oriented paradigm and visual modelling across the development life cycles to analyse and design an application, system, or company to improve communication and product quality [10]. Object-oriented design (OOD) defines how these software objects cooperate to meet the requirements; object-oriented programming (OOP) implements the design objects in a programming language. Object-oriented analysis (OOA) finds and describes business objects or concepts in the problem domain [9]. To meet the open-closed principle, which states that modules should be open for expansion but closed for modification, the object-oriented paradigm places a strong emphasis on modularity and reusability [15]. Iterative methods are frequently used in OOAD to work on requirements, analysis, design, coding, and system implementation. Object-oriented Analysis and Design (OOAD) consists of six phases which are requirements, analysis, design, coding, system implementation and testing phases as shown in Fig. 1.

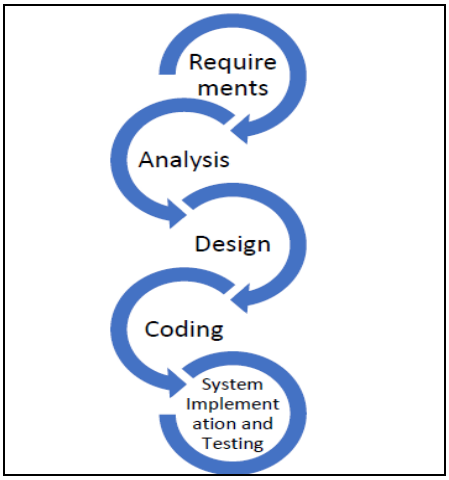


Fig. 1: *Object Oriented Analysis and Design Methodology [11]*

3.1 Requirement Phase

Requirements phase is to identify and determine problem statement, objective and scope and prepare a proposal to ensure the accuracy and completeness of the project Proposal of Title and idea to the supervisor is done at the beginning of the planning phase. Then, a Gantt Chart based on Object-oriented Analysis and Design methodology is prepared to make sure the smoothness of the project and it can be finished on schedule. Studies on related topics are also done to get more insight into the Secure Code Review System by researching into more studies on secure code review principles, static code analysis and common weaknesses enumerations in C such as OWASP TOP 10, Common Weakness enumeration and other further analysis for the next phase.

3.2 Analysis Phase

Analysis phase is the research phase to analysis the requirement of the Secure Code Review System. During the requirement analysis phase, studies, and review of three similar existing systems are done to gain more understanding of the system. Studies on Secure Code Review guidelines are also done to ensure the understanding of the concepts needed to be implemented in the secure coding review system. A comparison of characteristics between the existing systems and the system is made to make sure that the system contains the improvements needed. Then, the functional and non-functional requirements, hardware, and software requirement are defined. Lastly, the flowchart and context diagram of the Secure Coding Review System are designed. Table 3 and Table 4 show the hardware and software specifications respectively.

Table 3 *Hardware specification*

Hardware Requirements	
Model	Honor MagicBook Pro
Processor	AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
Operating System	Windows 10
Storage	512 GB SSD
Random Access Memory	16.0 GB

Table 4 *Software specification*

Software Requirements	
Visual Studio Code	Used to write codes and design interfaces
Flask	Run Machine Model Interface
XAMPP	Used to run the coding in localhost and connect to database
PhpMyAdmin	Used to run MySQL queries and create databases
Google Colab	Used to create machine learning model

3.3 Design Phase

Design Phase is visualizing the technical details and solution acquired from requirement and analysis phase. During the design phase, the system interface, machine learning model and database design is done. The system interface is illustrated in a wireframe using Figma containing all interfaces for the system. A flowchart, use-case and system architecture design using Draw.io is to show the machine learning model and system is done. The Entity Relationship Diagram using Draw.io, and data dictionary is used to design the database system.

3.4 Coding Phase

The coding phase is the development of PHP programming code to create the functionality of the backend of the system using Visual Studio Code. Python programming is also used to execute the Machine Learning model using Google Colab. Frontend programming such as HTML, CSS and JavaScript is used to create the system interface using Visual Studio Code and then XAMPP is used to run the coding in localhost and connect to database.

3.5 System Implementation and Testing Phase

The unit testing phase is done by creating test cases to define the inputs, expected outputs and conditions that should be done by the system. The machine learning system and validation is also done at this stage to get the accuracy of the model prediction and its confusion matrix. The bugs and problems are also identified and fixed in these phases. An acceptance criterion is defined for users which the criteria listed must be met to address every issue in each iteration and improve at the next iteration. User Acceptance Testing is done to collect feedback and evaluation from beta users using questionnaires then document it to be improved in the next iteration. Therefore, for every iteration a better system which fits the requirements is created and improved.

4. Analysis and Design

This section explains the analysis and design of the project which includes the functional requirements, non-functional requirements, Unified Modelling Language (UML) and System Design.

4.1 Functional Requirements

The functional requirements in Table 5 illustrate the functionalities of the user module which includes user registration, user login, secure code review, report generation and user community and the admin module which includes of admin registration, admin login and admin community.

Applied Information Technology and Computer Science Vol. 6 No. 1 (2025) p. 539-559

Table 5 *Functional Requirements*

Module	Functionalities
Registration: User, Admin	- Users register a new user account. - Admin details have been stored in database.
Login: User, Admin	- Users' login to their user account. - Admin login to their account.
Secure Code Review: User	- User able to upload existing C source code to be review for its security vulnerabilities and suggestion given.
Report Generation: User	- Detailed report on security vulnerabilities of the reviewed code containing feedback, recommendation and improvements suggestion generated.
Community: User, Admin	- Users can create a post, upvote, and comment on other posts. - Admin can view and delete post and users' information.

4.2 Non-functional Requirements

Non-functional requirements are the system operation requirements which must be achievable for the system. Table 6 is the Non-functional requirements table which consists of the modules and its respective functionalities. The non-functional requirements in Table 6 are usability, availability, and security. The usability of the system should be according to the scope stated and achieve all objectives, the availability is about the uptime of the system should be 24 hours daily and the security is about all sensitive information should be encrypted and stored safely and password should be hashed and salted before saving into database.

Table 6 *Non-functional Requirements*

Module	Functionalities
Usability	- System should be able to carry out all function based on the provided scope.
Availability	- System should be able to work with internet connectivity.
Security	- All sensitive information should be encrypted with AES 256 encryption Algorithm. - Password should be hashed and salted before saving into database.

4.3 Unified Modelling Language (UML)

Unified Modelling Language (UML) are model diagrams used to develop systems with visualizing the system into various diagrams such as use case diagram, class diagram, activity diagram and sequence diagram. The class diagram is used to illustrate the static relationships among the objects, and the activity diagram is used to show the system workflow. The use case diagram is to show the functional requirements of the system, and the sequence diagram is used to show how the use case scenario is connected. There are a total of five modules which are registration, login, secure code review, report generation and community.

4.3.1 Use Case Diagram

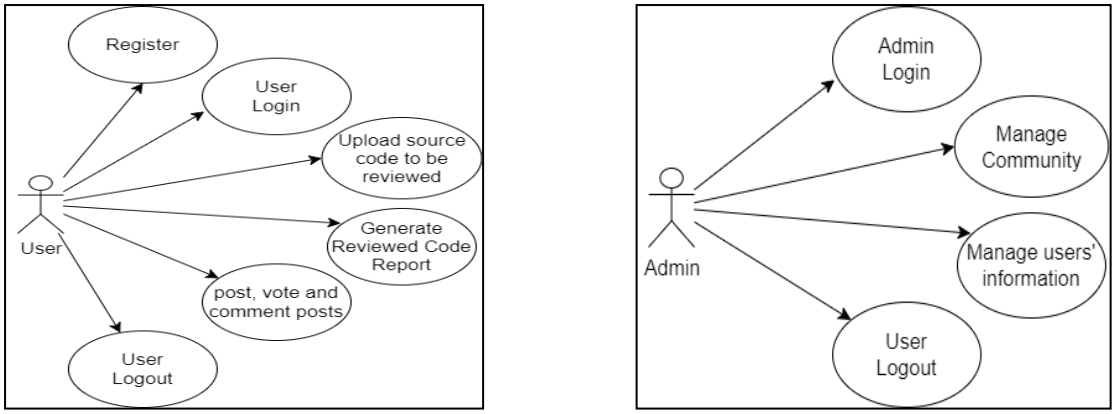
A use case diagram is used to illustrate the functional requirements of the system on the user and administrative. Fig. 2 (a) illustrates the use case diagram for the user while Fig. 2 (b) is the case diagram for admin. The user module includes register a new user account, login to their user account, upload source code to be review, generate reviewed code report and create a post, vote, and comment on other posts and logout. The administrator module includes login, logout and managing community and users' information.

4.3.2 Sequence Diagram

Sequence Diagram illustrates the control structure between objects and how the functions are conducted. The sequence diagram of the registration, login, code review, report summary and community interface are illustrated accordingly in Appendix A: Secure Coding Review System Sequence Diagram.

4.3.3 Activity Diagram

Activity Diagram illustrates the flow of the activity that is carried out in the system and shows its process. The activity diagram of the registration, login, code review, report summary and community interface are illustrated accordingly in Appendix B: Secure Coding Review System Activity Diagram.



(a) **(b)**

Fig. 2 *Secure Coding Review System's Use Case Diagram for(a) User; (b) Admin*

4.3.4 Class Diagram

The Class Diagram illustrates the structure of the system which includes the classes and their respective attributes, methods, and relationships between the objects. Fig. 3 illustrates the Secure Coding Review System's Class Diagram which contains the objects users, admin, post, comments, and code report.

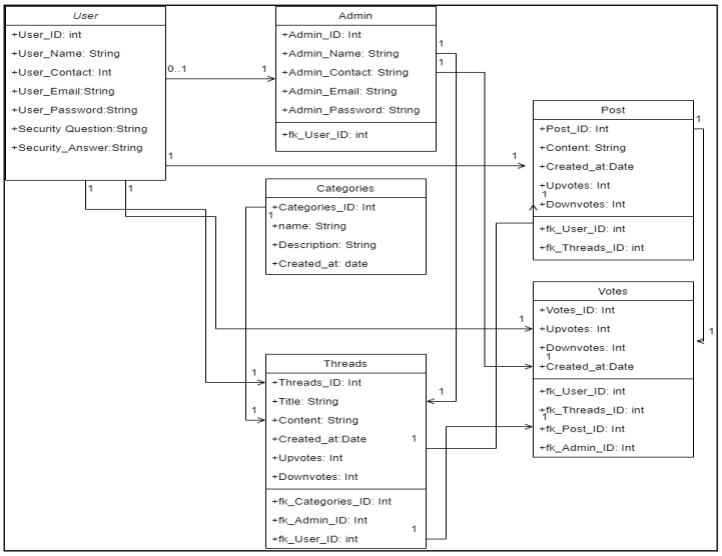


Fig. 3 *Secure Coding Review System's Class Diagram*

4.4 System Design

System Design is to aid in developing the system according to the requirements stated. This section discusses the System Architecture Design, Flowchart and Entity-Relationship Diagram (ERD).

4.4.1 System Architecture Design

System Architecture Design provides a visualization on how the system will be communicating with each other to accomplish the objectives of the system. The system requires the internet to request and response to run the system. Once it is successful, the user can access into the system as admin or user where they will be able to access to their respective module. For user, they can upload their code to be reviewed, generate and view report, view or comment post while admin can manage community and users. Fig. 4 illustrates the system architecture design of the secure coding review system. It starts from user requesting access to the Secure Code Revieing System through the browser. After successful access to the system interface, user can login. Admin can manage community and user where all data exchange will be made with the database. On the other hand, user can upload source code to be reviewed using Random Forest machine learning algorithm (blue square box in Fig. (4). User also van view the report generated. The user can also access to the Community platform to create, view, comment or vote post.

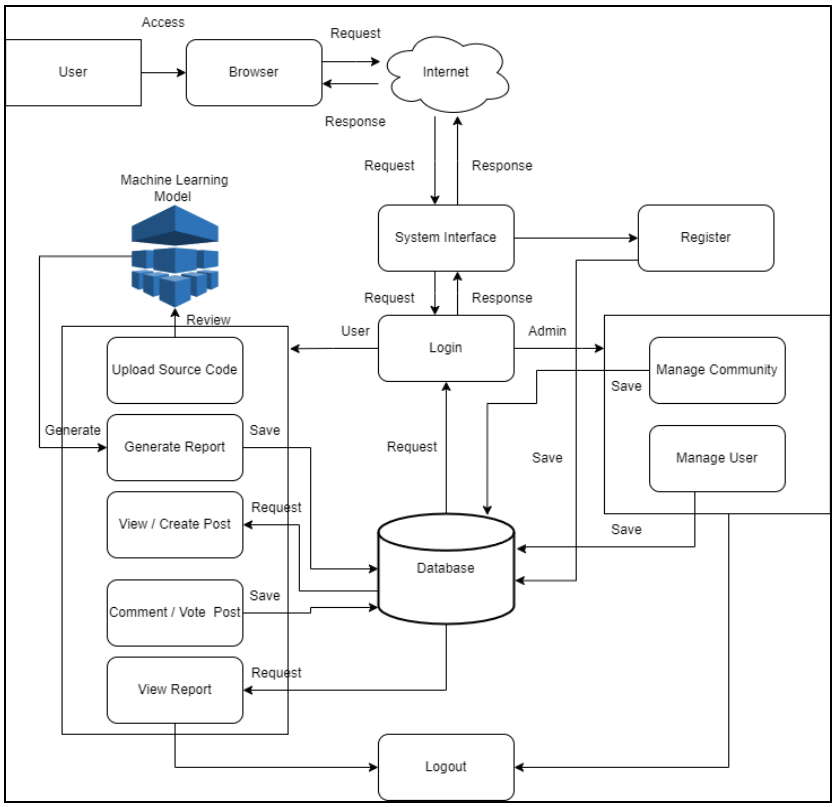


Fig. 4 *Secure Coding Review System's System Architecture Design*

4.4.2 Flowchart

Fig. 5 shows the flowchart of the Random Forest algorithm on whether there are vulnerabilities and categorizes the vulnerabilities according to array bounds violated, buffer overflow and dereference failure. Then, it will calculate the outcome and provide a detailed output on the vulnerabilities spotted by giving feedback, recommendation and suggestions.

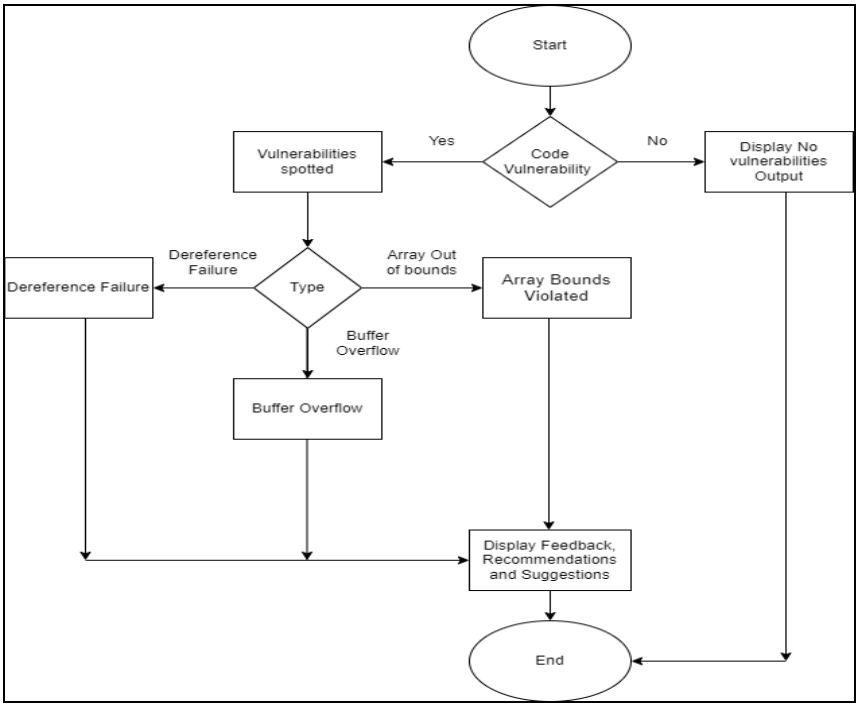


Fig. 5 *Flowchart for Random Forest Algorithm*

Fig. 6 shows a Flowchart to illustrate the sequence of the systems on how all components will work starting from registration and login. This makes it easier to develop the system accordingly and make it easy to understand how the system works. From the flowchart, the input of user, process, decision, and output of the user can be seen clearly according to its shapes respectively. The new user is required to register an account before logging into the system. Then, the user can login using their valid user and password. The user can choose if they want to review source code or go to the community. If user choose to review a code, they can upload source code and the system will generate an output on reviewed code where user can choose to generate a report or end. If user chooses to go community, user can choose to either search for a post or create a post. If user wants to create a post, there are required to create the post, save and upload it and they system will display it. If the user chooses to search post, system will display related post and user can select the post they want, the system will display the post they selected. Then user can choose to either vote or command the post before leaving the page.

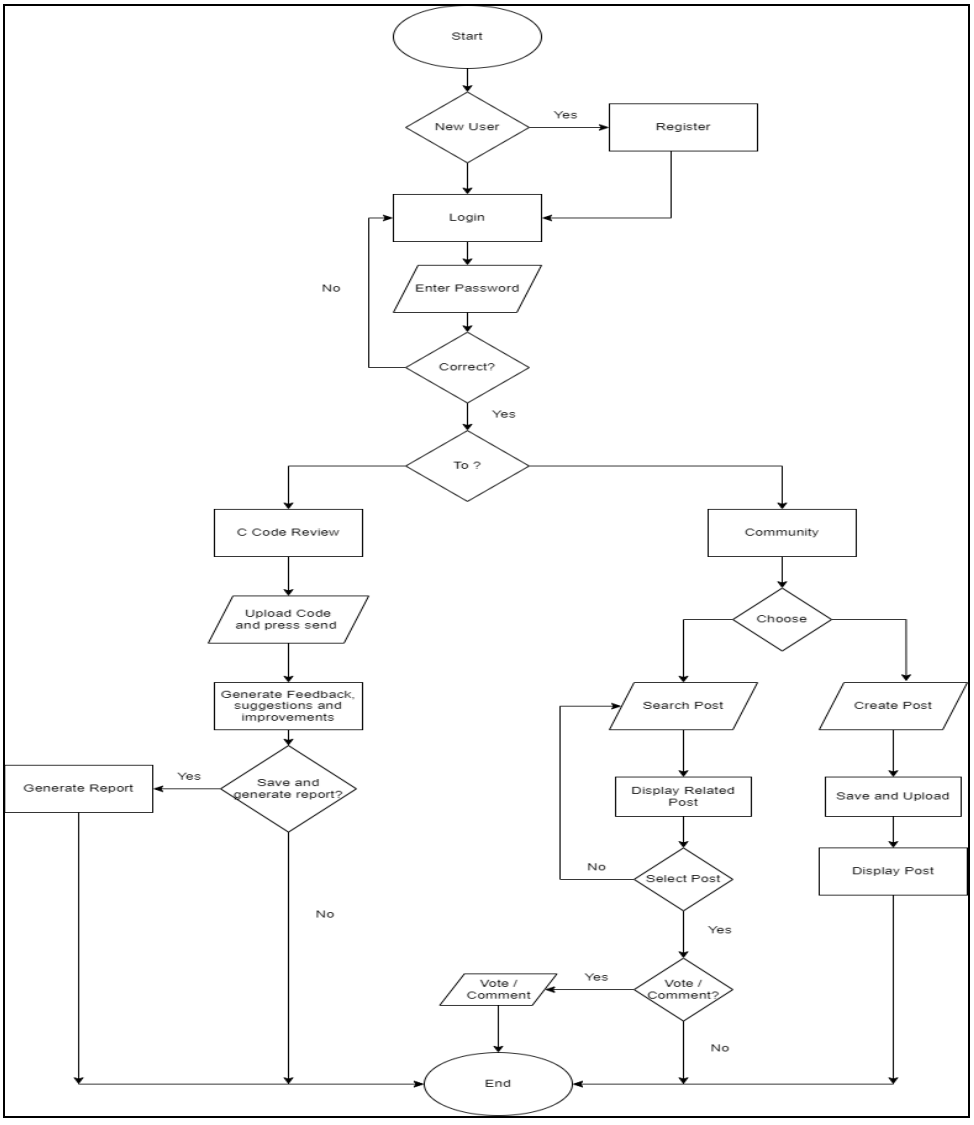


Fig. 6 *Flowchart for Secure Coding Review System*

4.4.3 Entity-Relationship Diagram (ERD)

The Entity-Relationship Diagram (ERD) illustrates the database design by describing the object and its attributes that are included in the database for information storing and exchange of information between the system and the database. There are six objects with its respective attributes as shown in Fig. 7 shows the Secure Coding Review System's Entity-Relationship Diagram which contains the objects users, admin, post, comments, and code report. Each object has their respective Primary Key which is declared as "Object_ID". Each object is also connected to each other with foreign keys to link the data among them.

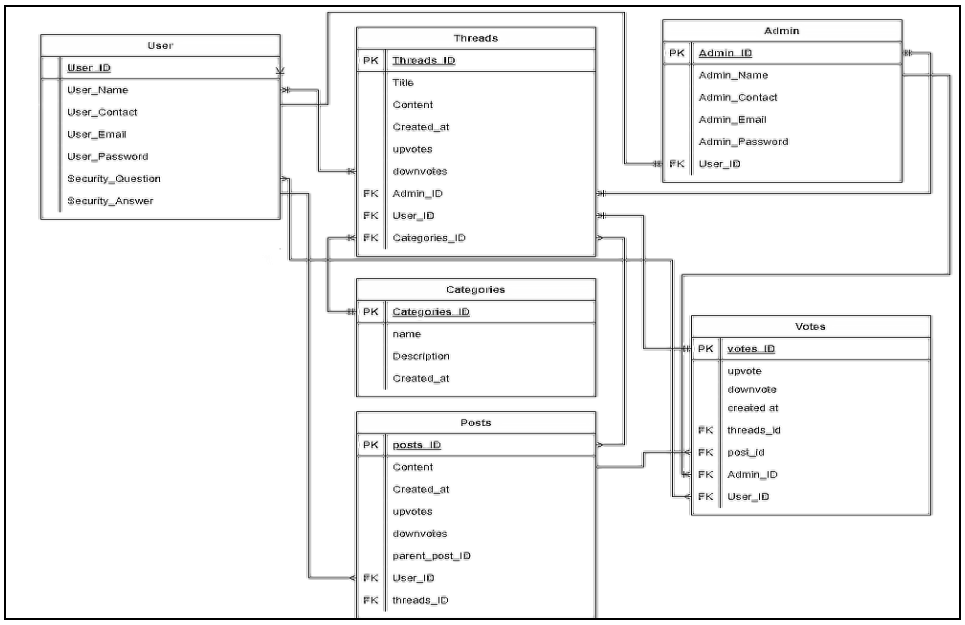


Fig. 7 *Secure Coding Review System's Entity-Relationship Diagram*

4.5 Implementation

Implementation is carried out to determine all functionality of each module is satisfied and fulfilled.

4.5.1 Registration Login Page

Fig. 8 and Fig. 9 illustrate the system's Registration and Login page. The system must be able to receive inputs from users, verify the details, verify captcha and allow the user to access the system after successful registration and login. If the verification is valid during login which matches the information user registered with, the patient can access the second authentication page.

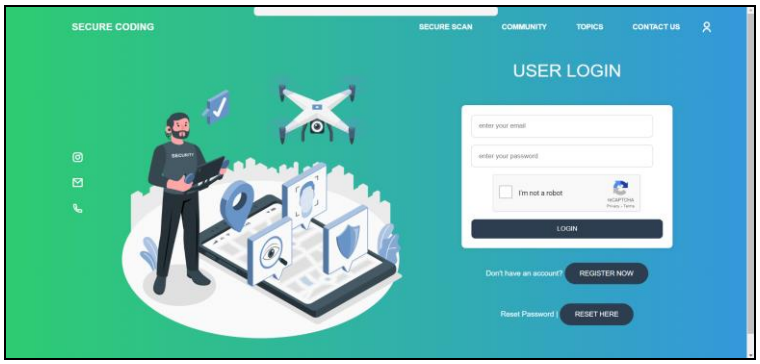


Fig. 8 *Login Page*

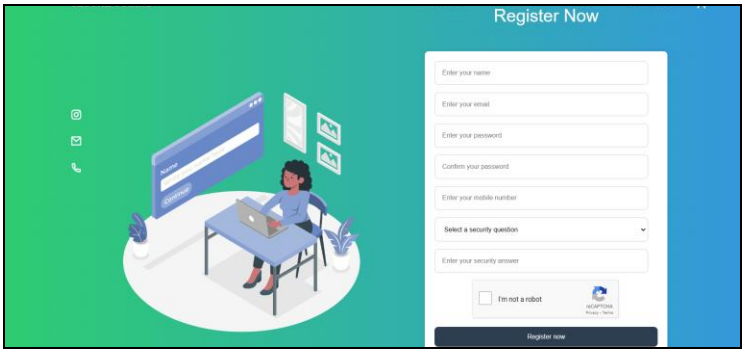


Fig. 9 *Registration Page*

4.5.2 OTP Verification Page

Fig. 10 shows the OTP Verification Page which gives the systems additional security as it implements a 2-factor authentication where user will receive an OTP at their email at each login as in Fig. 11. If the OTP is valid, the user will be able to gain access to the system.

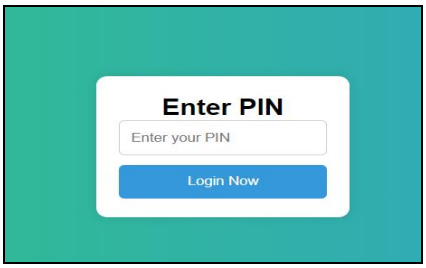


Fig. 10 *Enter Pin Page*

Fig. 11 *Pin Received in Email*

4.5.3 Community Module

Community module is developed for users to view threads, comments, upvotes and downvotes related to subjects related to programming. The higher the upvotes, the higher the questions will be listed on the threads list page of the community. For more than 50 downvotes would cause the post to be discarded by an admin. Fig. 12 illustrates the home page of the forum which shows the categories of discussion and latest threads to the users. Fig. 13 shows the interface of the create thread page where user can create their own thread by filling in the required inputs. Fig. 14 shows the list of threads under each category seen in the home page and Fig. 15 illustrates the thread of each user where user can reply, upvote or downvote threads.

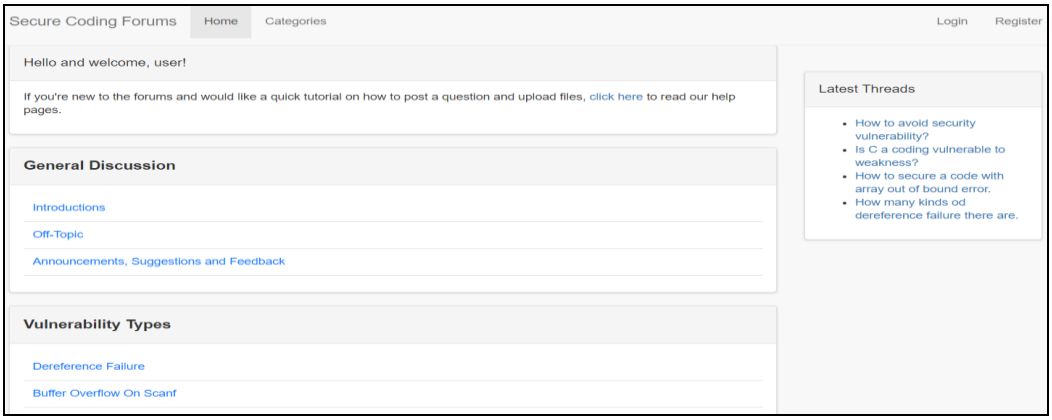


Fig. 12 *Community Home page*

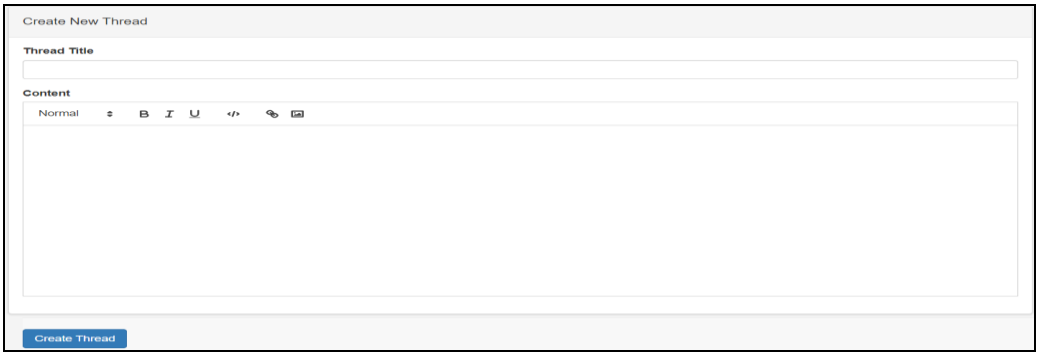


Fig. 13 *Post Thread page*

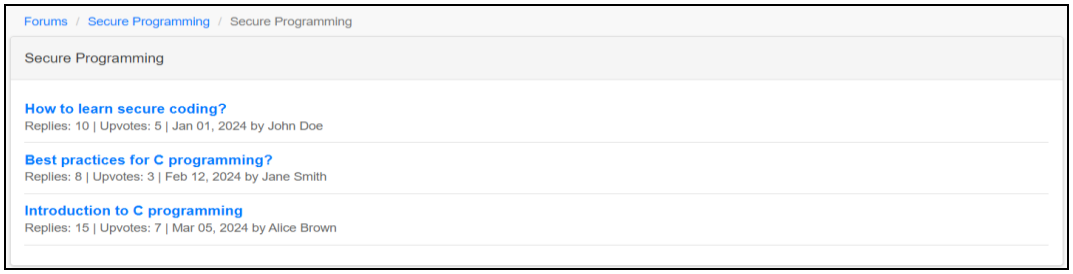


Fig. 14 *Threads under Category Secure Programming page*

Fig. 15 *Threads details page*

4.5.4 Secure Code Review Module

The Secure Code Review Module is built using HTML, CSS, JavaScript, and PHP. The Random Forest Algorithm model which are the vulnerability and error type prediction model is used to scan code. The dataset used to train the model are extracted from FORM_AI datasets [4]. Python is used to train the machine learning model as Its object-oriented programming methodology is straightforward but efficient, and its high-level data structures are efficient. Because of its interpreted nature, dynamic typing, and attractive syntax, Python is a great language for scripting and quick application development across a wide range of platforms [18]. A report page and a secure scan review page make up the module's two sections. It gathers information from the Abstract Syntax Tree (AST) [14], cleans and standardises data before processing it, then by using Clang [20] to remove comments and preprocess the code and incorporates machine learning to forecast code vulnerabilities. Additionally, the module has a chatbot interface for interactive code review sessions. It guides users through file uploads, error type analysis, and vulnerability detection, and the results are displayed in an extensive HTML report.

Fig. 16 illustrates the code on loading the machine learning models and its corresponding vectorizer and encoder, Fig. 17 shows the function of feature selection where the comment is removed, and the code is preprocessed. Fig. 18 functions check for each line if there are any vulnerabilities and continue to loop through the whole coding to predict vulnerabilities and its error types as in Fig. 19. Fig. 20 illustrates the uploaded file and pulling all the function at main to run all functions to get the prediction output.

```
# Load the saved models, encoders, and vectorizer
vulnerability_model_file_path = 'final_Code_Analysis_vulnerability_model_rf.joblib'
vulnerability_vectorizer_file_path = 'final_tfidf_vectorizer_rf_vulnerability.joblib'
error_type_model_file_path = 'final2_Code_Analysis_error_type_model.joblib'
error_type_encoder_path = 'final2_error_type_encoder.joblib'
function_name_encoder_path = 'final2_function_name_encoder.joblib'
tfidf_vectorizer_file_path = 'final2_tfidf_vectorizer_rf.joblib'

# Load models and encoders
try:
    vulnerability_clf = joblib.load(vulnerability_model_file_path)
    vulnerability_vectorizer = joblib.load(vulnerability_vectorizer_file_path)
    best_clf_error_type = joblib.load(error_type_model_file_path)
    error_type_encoder = joblib.load(error_type_encoder_path)
    function_name_encoder = joblib.load(function_name_encoder_path)
    tfidf_vectorizer = joblib.load(tfidf_vectorizer_file_path)
except Exception as e:
    print(f"Error loading models: {str(e)}")
```

Fig. 16 *Loading Models*

```
def remove_comments(code):
    pattern = r"(//.*?$|/\*.*?\*/)"
    code = re.sub(pattern, "", code, flags=re.DOTALL | re.MULTILINE)
    return code

def preprocess_code(code, vectorizer, window_size=3):
    lines = code.split('\n')
    processed_lines = []
    for i in range(len(lines)):
        start = max(0, i - window_size)
        end = min(len(lines), i + window_size + 1)
        context = " ".join(lines[start:end])
        processed_lines.append(context)

    code_tfidf = vectorizer.transform(processed_lines)
    return code_tfidf.toarray()
```

Fig. 17 *Function to remove comments from uploaded code*

```
def is_vulnerable_line(line, window_size=3):
    line_tfidf = vulnerability_vectorizer.transform([line])
    prediction = vulnerability_clf.predict(line_tfidf)
    return prediction[0] == 1
```

Fig. 18 *Function to check for vulnerability each line*

```
def predict_error_type_for_lines(code, error_type, window_size=3):
    predictions = []
    known_function_names = set(function_name_encoder.classes_)

    unknown_function_encoded = np.mean(function_name_encoder.transform(list(known_function_names)))

    code_tfidf = preprocess_code(code, tfidf_vectorizer, window_size)
    lines = code.split('\n')
```

Fig. 19 *Function to check for the error type if vulnerable.*

```
def main():
    # Upload C code file
    uploaded = files.upload()

    for filename in uploaded.keys():
        with open(filename, 'r') as file:
            code = file.read()

        # Remove comments from code
        code = remove_comments(code)

        all_predictions = []
        error_types = ["array bounds violated", "buffer overflow on scanf", "dereference failure: NULL pointer",
                       "dereference failure: accessed expired variable pointer", "dereference failure: array bounds violated",
                       "dereference failure: forgotten memory", "dereference failure: invalid pointer", "division by zero"]

        print("Checking code for vulnerabilities...")
        for error_type in error_types:
            predictions = predict_error_type_for_lines(code, error_type)
            all_predictions.append((error_type, predictions))

            # Display predictions
            if predictions:
                print(f"\nResults for {error_type}:")
                for function_name, line_number, error_type in predictions:
                    print(f"Function: {function_name}, Line: {line_number}, Predicted Error Type: {error_type}")
            else:
                print(f"No occurrences of {error_type} found.")

        # Generate a report
        generate_report(all_predictions)
```

Fig. 20 *Main Function*

A chatbot user interface for the secure code review is developed for a more interactive session with user such as Fig. 21. Users can choose 1 or 3 to learn more about general enquiries and more about secure coding information. As for selection 2, user can carry out their secure code review such as Fig. 22, which they will be prompted to upload a c file, and their c file will be displayed which will lead them to vulnerability detection following with Error Type detection and user can type into the number which the error is to learn more specific details on the error type such as in Fig. 23. Lastly, the user can type exit to end the conversion which will generate a Hypertext Markup Language (HTML) report immediately and redirect user to the report page.

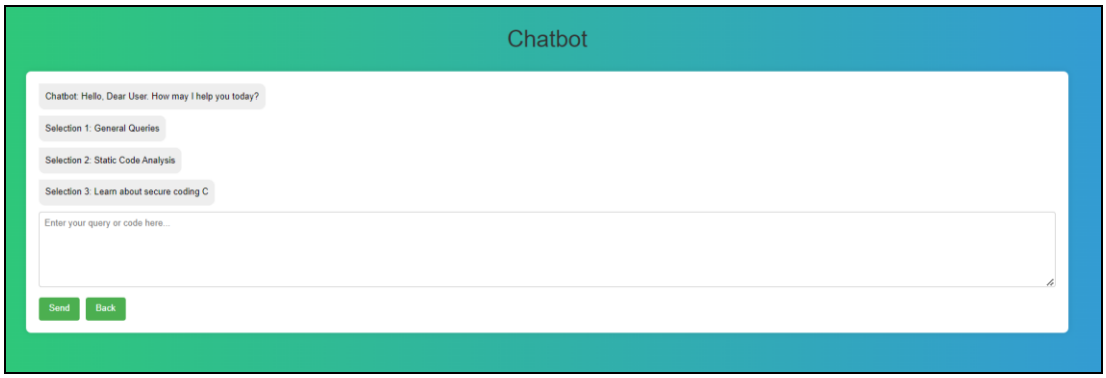


Fig. 21 *Main Secure Code Review Interface*

Fig. 22 *Secure Code Review Interface (Selection 2)*

Fig. 23 *Secure Code Review Prediction Output Interface*

4.5.5 Report Module

The report module consists of the report after the vulnerability scan. It consists of few parts such as the vulnerability type, occurrences, the line where the vulnerability is, the function where the vulnerability is, the related Common Weakness Enumeration and its website link, the explanation of the vulnerability, suggestion and the sample code for user to reference and improve their code. Users can download and print the report at the report module. Fig. 24 shows the scan report which summarizes the predicted vulnerabilities to see how many occurrences have occurred and lists out all the function and the line where the vulnerability is while Fig. 25 is the layout after pressing the print report button. Fig. 26 and 27 show the code on formatting the output of the report after every scanning which resulted in the report illustration shown.

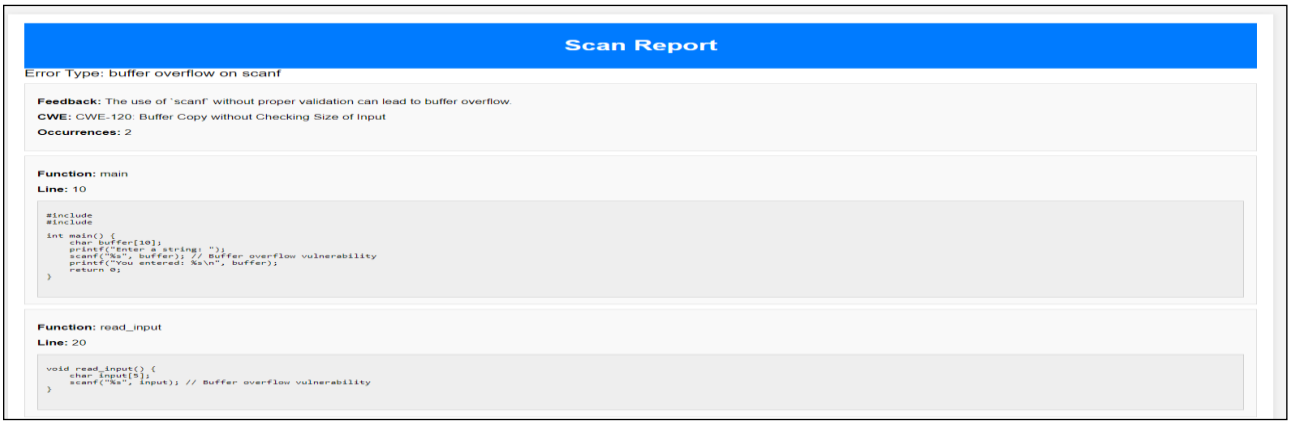


Fig. 24 *Scan Report Module*

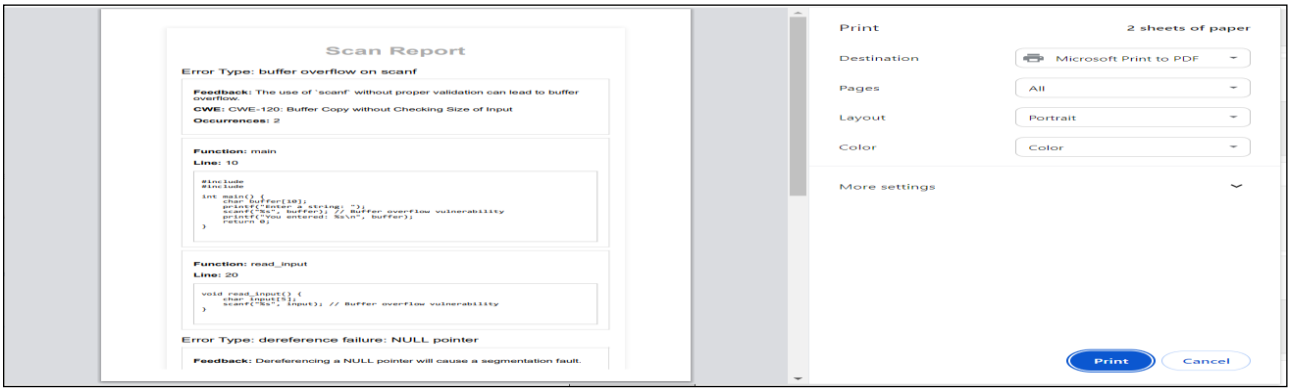


Fig. 25 *Print Report Module*

```
def generate_html_report(results, filename="report.html"):
    html_content = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Scan Report</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 0; padding: 0; background-color: #f4f4f4; }
        .container { width: 80%; margin: 20px auto; padding: 20px; background: #fff; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); }
        .header { background: #007bff; color: #fff; padding: 10px 0; text-align: center; }
        .section { margin-bottom: 20px; }
        .section-title { font-size: 20px; margin-bottom: 10px; }
        .box { padding: 15px; border: 1px solid #ddd; background: #f9f9f9; margin-bottom: 10px; }
        .code-box { background: #eee; padding: 10px; border: 1px solid #ccc; }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>Scan Report</h1>
        </div>
    """

    for error_type, predictions in results.items():
        html_content += f"""
        <div class="section">
            <div class="section-title">Error Type: {error_type}</div>
            <div class="box">
                <p><strong>Feedback:</strong> {predictions['feedback']}</p>
                <p><strong>CWE:</strong> {predictions['cwe']}</p>
                <p><strong>Occurrences:</strong> {len(predictions['occurrences'])}</p>
            </div>
```

Fig. 26 *Code for scan report (Format Output)*

```
            for function_name, line_number in predictions['occurrences']:
                html_content += f"""
                <div class="box">
                    <p><strong>Function:</strong> {function_name}</p>
                    <p><strong>Line:</strong> {line_number}</p>
                    <div class="code-box">
                        <pre>{predictions['example']}</pre>
                    </div>
                </div>
                """

            html_content += "</div>"

        html_content += """
        </div>
    </body>
    </html>
        """

        with open(filename, 'w') as f:
            f.write(html_content)
```

Fig. 27 *Code for scan report cont. (Format Output)*

4.6 Test Case

Test cases of the system modules such as user registration module, user login module, code review module, community module and report generation module are created to test the satisfaction of the beta users.

Table 7 *Test Case for User registration*

No	Test Cases	Description	Test Results	
Test_1A			Success	Fail
User Registration Module				
1.	Test_1A_001	A new user should be able to register a new account.	✓	
2.	Test_1A_002	The system should display error messages if there are any error inputs.	✓	
3.	Test_1A_003	The system should display a success message upon successful registration.	✓	
3.	Test_1A_004	The system displays error messages if RECAPTCHA not checked.	✓	

Table 8 *Test Case for User Login and Pin verification*

No	Test Cases	Description	Test Results	
Test_1B			Success	Fail
User Login Module				
1.	Test_1B_001	Registered users should be able to redirect to enter pin page successfully after inputting correct email and password.	✓	
2.	Test_1B_002	The system should send an OTP pin to the user's email after successful inputting correct email and password.	✓	
3.	Test_1B_003	The system should verify the validity of the OTP pin inputted by the user for successful login.	✓	
4.	Test_1B_004	The system should display error messages if there are any error inputs or the pin entered is invalid.	✓	
5.	Test_1B_005	The system displays error messages if RECAPTCHA not checked.	✓	
6.	Test_1B_006	The system should verify and start a session according to the specific user.	✓	

Table 9 *Test Case for review code module*

No	Test Cases	Description	Test Results	
Test_1C			Success	Fail
Review Code Module				
1.	Test_1C_001	The system should allow the user to upload a .C file and press the upload and scan button.	✓	
2.	Test_1C_002	The system should be able to run a review on the C coding for accessing vulnerabilities.	✓	
3.	Test_1C_003	The system should be able to provide feedback, recommendation and improvements after every successful review.	✓	
3.	Test_1C_003	The system should display error messages if there are any error inputs.	✓	
4.	Test_1C_004	The system should be able to generate a report after every successful review.	✓	
5.	Test_1C_005	The system should be able to scan for buffer overflow on scanf vulnerability type.	✓	
6.	Test_1C_006	The system should be able to scan for arrays bound violated vulnerability type.	✓	
7.	Test_1C_007	The system should be able to scan for dereference failure vulnerability type.	✓	

Table 10 *Test Case for community module*

No	Test Cases	Description	Test Results	
Test_1D			Success	Fail
Community Module				
1.	Test_1D_001	The system should allow users to post their queries into the community forum.	✓	
2.	Test_1D_002	The system should allow users to vote and comment on the post of other users.	✓	
3.	Test_1D_003	The system should allow users to search for related forums using a search bar.	✓	

Table 11 *Test Case for report module*

No	Test Cases	Description	Test Results	
Test_1E			Success	Fail
Report Module				
1.	Test_1E_001	The system should allow users to view their report on the review outcome of their C coding.	✓	
2.	Test_1E_002	The system should allow users to print their report on the review outcome of their C coding.	✓	

4.7 User Testing

User acceptance testing is carried out to determine if users are satisfied with the functionality and interface of the system. A set of questions are given in the form based on functionality and user satisfaction. A total of 10 beta users who have done the testing and filled out the forms are FSKTM's final year students majoring in Information Security who had experienced coding in C programming language. Table 12 shows the questions in the user acceptance form. The survey form consists of various acceptance requirements which the beta testers are required to test the Secure Code Review System and fill up the form by filling up yes or no to see if the requirements are met.

Table 12 *User Acceptance Form*

Question	Acceptance Requirement	Result	
		Yes	No
1	Process of registration and login is smooth.		
2	User able to upload a .C file for security vulnerabilities scanning.		
3	User able to scan .C file for security vulnerabilities.		
4	Report is generated after each scanning.		
5	Report generated is easy to understand.		
6	User able to create, update and delete their post at forum.		
7	User able to view, comment, upvote and downvote existing post.		
8	The overall system layout is user-friendly.		
9	Admin able to view and delete post and user accounts.		
10	Admin able to update categories		

Fig. 28 illustrates the user acceptance result from the 10 beta users who are FSKTM's final year students majoring in Information Security who had experienced coding in C programming language. From the results below, all 10 beta users chose yes for all the questions above which consists of the overall functionality and user experience of the system. With these results, the conclusion that can be made is that all objectives have been achieved and the beta users are satisfied with the functionality of the system and its experience.

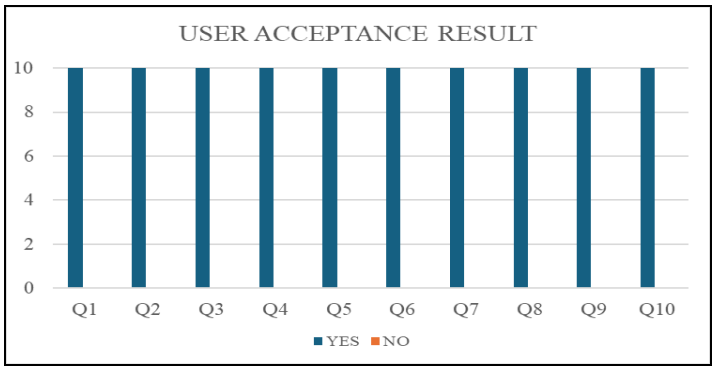


Fig. 28 *User Acceptance Result*

4.8 Model Testing and Validation

Model testing and validation is conducted to get information about the recall, accuracy, precision, and general dependability of the model. We will use a subset of the FormAI dataset [4] that was not used for training to test the model. This guarantees that the performance indicators of the model accurately represent its capacity to generalize to new, untested data. The dataset contains various features related to code segments, their vulnerability types, and associated error types.

Fig. 29 (a) and (b) illustrates the model testing and validation for the vulnerability model to test whether a C source code is vulnerable or not vulnerable. Fig. 29 (a) illustrates that the model has an accuracy of 91.8 percent which means that the model is able to accurately predict the vulnerability of the source code with minimal false negatives and false positives while Fig. 29 (b) illustrates the prediction output result for the vulnerability model which shows 9 out of 10 predictions are predicted correctly.

```
Vulnerability Detection Model Report (Test Set):
                          precision    recall  f1-score   support

NOT VULNERABLE up to bound k       0.91      0.93      0.92      1889
                  VULNERABLE       0.93      0.91      0.92      1951

                    accuracy                           0.92      3840
                   macro avg       0.92      0.92      0.92      3840
                weighted avg       0.92      0.92      0.92      3840

Accuracy (Test Set): 0.9182291666666667
Cross-Validation Scores (Training Set): [0.90507812 0.91757813 0.91074219]
Mean Cross-Validation Score (Training Set): 0.9111328125
Confusion Matrix (Test Set):
[[1759  130]
 [ 184 1767]]
TP: 1767, TN: 1759, FP: 130, FN: 184
```

(a)

```
Predictions for 5 Vulnerable and 5 Not Vulnerable Cases (Test Set):

Vulnerable Cases:
Predicted: VULNERABLE, Actual: VULNERABLE
Predicted: VULNERABLE, Actual: VULNERABLE
Predicted: VULNERABLE, Actual: VULNERABLE
Predicted: VULNERABLE, Actual: VULNERABLE
Predicted: VULNERABLE, Actual: VULNERABLE

Not Vulnerable Cases:
Predicted: NOT VULNERABLE up to bound k, Actual: NOT VULNERABLE up to bound k
Predicted: NOT VULNERABLE up to bound k, Actual: NOT VULNERABLE up to bound k
Predicted: NOT VULNERABLE up to bound k, Actual: NOT VULNERABLE up to bound k
Predicted: VULNERABLE, Actual: NOT VULNERABLE up to bound k
Predicted: NOT VULNERABLE up to bound k, Actual: NOT VULNERABLE up to bound k
```

(b)

Fig. 29 *Model Testing and Validation for vulnerability (a) Accuracy and Confusion Matrix; (b) Prediction Result*

Fig. 30 (a) and (b) illustrates the model testing and validation for the error type model to evaluate the accuracy of the error type model to accurately predict an error type. Fig. 30 (a) illustrates that the model has an accuracy of 86.5 percent which means that the model is able to accurately predict the error type of the source code which is a satisfactory result while Fig. 30 (b) illustrates the confusion matrix result for the error type model which shows that there are 5 False Positives which is a common disadvantage when doing static analysis.

```
Error Type Detection Model Report (Test Set):
                                    precision    recall  f1-score   support

                  arithmetic overflow       0.82      0.79      0.81       209
                array bounds violated       0.90      0.91      0.91       166
              buffer overflow on scanf       0.88      0.85      0.86       222
      dereference failure: NULL pointer       0.86      0.83      0.84       235
dereference failure: array bounds violated       0.85      0.90      0.87       215
   dereference failure: forgotten memory       0.85      0.89      0.87       206
    dereference failure: invalid pointer       0.90      0.88      0.89       208
                   division by zero       0.86      0.89      0.88       213

                            accuracy                           0.86      1674
                           macro avg       0.87      0.87      0.87      1674
                        weighted avg       0.87      0.86      0.86      1674
```

(a)

```
Accuracy (Test Set): 0.8649940262843488
Confusion Matrix (Test Set):
[[165   5   4   4   4   3   7  17]
 [  0 151   7   2   3   0   0   3]
 [ 16   3 189   4   0   0   2   8]
 [  1   0   6 194   6  27   0   1]
 [  0   4   4   7 193   2   5   0]
 [  0   0   0  10   6 184   5   1]
 [  4   2   1   3  15   0 183   0]
 [ 14   2   4   1   1   1   1 189]]
TP: 151, TN: 165, FP: 5, FN: 0
```

(b)

Fig. 30 *Model Testing and Validation for error type model (a) Accuracy; (b) Confusion Matrix*

Fig. 31 (a) and (b) illustrates the model testing and validation for the error type model to evaluate the capability of the model to predict the error type accurately. Each error type categories are tested five times against the test dataset. Based on the observation from these figures, it validates that there is minimal prediction error as it only predicts correctly the error type 34 times out of 40 times.

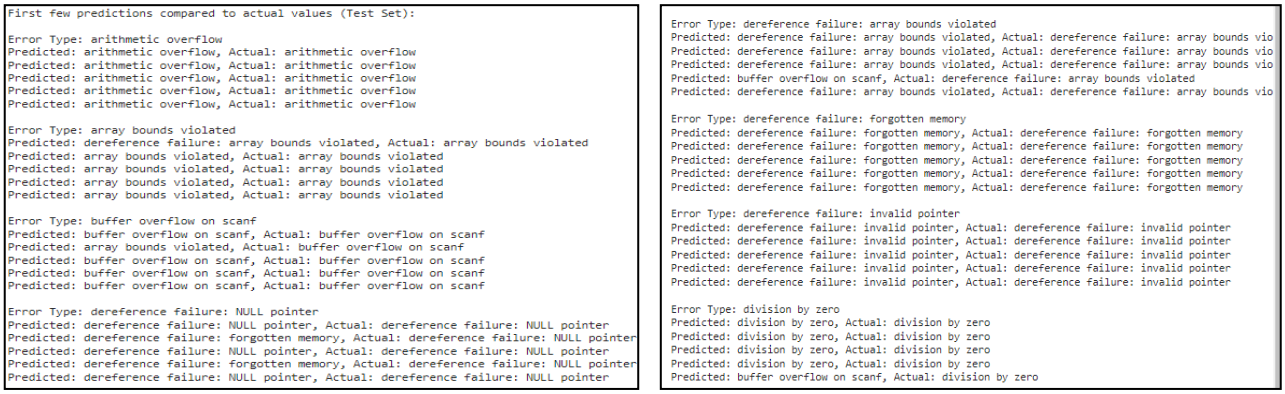


Fig. 31 *Model Testing and Validation for error type model (a) Prediction Result; (b) Prediction Result cont.*

5. Conclusion

Secure Coding Review System for accessing vulnerabilities is developed to examine the program's source code by identifying any security vulnerabilities in the program. The Secure Coding Review System is also developed to compensate the lack of tools or systems implementing static secure code analysis, provide a Graphical User Interface which provides easier access to the user and provide more clear dan detailed feedback, recommendation and improvements and integrates a community platform, which serves to promote opportunities among developers to provide feedback and code verification.

The advantage of this system is that it is built utilizing random forest machine learning algorithms to process and predict outcomes of the reviewed source code and to provide a detailed report for the users. The report is also formatted into a clear output of feedback, recommendation, and improvement for the user to fully understand what should be changed and the reason for the security check. The system also provides a community platform for the users to communicate and discuss relevant topics on secure programming. Besides that, there are several disadvantages of the systems as the system only provides static code analysis, it may not cover all types of vulnerabilities and there may be false positives when predicting the vulnerabilities. The system has also not been hosted into a domain which makes it inaccessible to the public.

Therefore, the system has its limitations and needs further improvements for future implementation. The system needs to be able to cover more security vulnerabilities and implement secure dynamic code review function to fully make this system a well-rounded secure code review system which has both Static and Dynamic Secure Code Review. Besides that, more datasets are needed to train the machine learning model to provide more accurate prediction. Lastly, the system should be hosted in the future to be accessible to the public.

Acknowledgment

The author would like to thank the Faculty of Computer Science and Information Technology, University of Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Y. L. Chin, I. R. A Hamid;* ***data collection:*** *Y. L. Chin, I. R. A Hamid;* ***draft manuscript preparation:*** *Y. L. Chin, I. R. A Hamid. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] E. Keary, L. Conklin, and Robinson Gary, "CODE REVIEW GUIDE RELEASE." [Online]. Available: https://www.owasp.org

[2] K. Akcura, R. Shalchian, A. Patil, R. Singh, and J. Tanna, “Static Versus Dynamic Source Code Analysis,” Nov. 2015.
[3] "About CWE - Common Weakness Enumeration," CWE, https://cwe.mitre.org/about/index.html.
[4] Norbert Tihanyi, Tamas Bisztray, Ridhi Jain, Mohamed Amine Ferrag, Lucas C. Cordeiro, Vasileios Mavroeidis, June 18, 2023, "FormAI Dataset: A Large Collection of AI-Generated C Programs and Their Vulnerability Classifications", IEEE Dataport, doi: https://dx.doi.org/10.21227/vp9n-wv96.
[5] "2023 CWE Top 25 Most Dangerous Software Weaknesses," Common Weakness Enumeration (CWE), https://cwe.mitre.org/top25/archive/2023/2023_top25_list.html, 2024.
[6] D. Wheeler, “Flawfinder Home Page,” dwheeler.com. https://dwheeler.com/flawfinder/
[7] “Cppcheck - A tool for static C/C++ code analysis,” cppcheck.sourceforge.io. https://cppcheck.sourceforge.io/
[8] “Splint Documentation,” splint.org. https://splint.org/documentation/
[9] JavaTpoint, “Object-oriented Analysis and Design SDLC | Software Development Life Cycle - Javatpoint,” www.javatpoint.com. https://www.javatpoint.com/Object-oriented Analysis and Design-sdlc
[10] D. Kung and J. Lei, "An Object-Oriented Analysis and Design Environment," 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET), Dallas, TX, USA, 2016, pp. 91-100, doi: 10.1109/CSEET.201B.20.
[11] Object-oriented analysis and Design, https://www.eeng.dcu.ie/~ee553/ee402notes/html/ch01s05.html#:~:text=Formulate%20the%20problem%20%2D%20The%20programmer,code%20to%20implement%20the%20design.
[12] J. Y. Koh, K. M. Mohamad, and S. N. Ramli, “Rand-Quiz: Web-Based Quiz System with OTP Using KJY Password Generator”, aitcs, vol. 4, no. 2, pp. 159–174, Nov. 2023, Accessed: Jul. 20, 2024. [Online]. Available: https://publisher.uthm.edu.my/periodicals/index.php/aitcs/article/view/12107
[13] Z. Q. Chong and S. Jamel, “Web-Based Auction System with Dual-Authentication for Syarikat Perniagaan Fong,” *Applied Information Technology and Computer Science*, vol. 4, no. 2, pp. 83–102, 2023, doi: 10.30880/aitcs.2023.04.02.006.
[14] “Abstract Syntax Trees Python 3.8.3 documentation,” *docs.python.org*. https://docs.python.org/3/library/ast.html
[15] B. Dathan and S. Ramnath, Object-Oriented Analysis, Design and Implementation: An Integrated Approach. Cham: Springer, 2015.
[16] NIST, "Security and Privacy Controls for Information Systems and Organizations," SP 800-53 Rev. 5, 2020. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final.
[17] Owasp, “Input Validation OWASP Cheat Sheet Series,” Owasp.org, 2019. https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
[18] L. Jupudi, "Machine learning techniques using python for data analysis in performance evaluation," International Journal of Intelligent Systems Technologies and Applications, vol. 17, no. 3, 2018, doi: 10.1504/IJISTA.2018.10012853.
[19] D. Fagbuyiro, “Benefits of using static code analysis tools for software testing,” StickyMinds, https://www.stickyminds.com/article/benefits-using-static-code-analysis-tools-software-testing#:~:text=Using%20static%20code%20analysis%20tools%20can%20help%20improve%20the%20efficiency,products%20more%20efficiently%20and%20effectively.
[20] “Clang Compiler User’s Manual - Clang 19.0.0git documentation,” clang.llvm.org. https://clang.llvm.org/docs/UsersManual.html