**Leveraging Generative AI for Intelligent Source Code Analysis and Vulnerability Detection**

**A PROJECT REPORT SUBMITTED TO**

**SRM INSTITUTE OF SCIENCE & TECHNOLOGY**

**IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE**

**AWARD OF THE DEGREE OF**

**MASTER OF COMPUTER APPLICATIONS**

**BY**

**KEERTHANA L**

**REG NO. DA2232306010039**

**UNDER THE GUIDANCE OF**

**Mr. J. VENKATA SUBRAMANIAN M.C.A., M.Phil., M.S.,** 𝐏𝐡̅̅̅̅.̅D.̅,



**DEPARTMENT OF COMPUTER APPLICATIONS**

**FACULTY OF SCIENCE AND HUMANITIES**

**SRM INSTITUTE OF SCIENCE & TECHNOLOGY**

Kattankulathur – 603 203

Chennai, Tamilnadu

**June - 2024**

**BONAFIDE CERTIFICATE**

This is to certify that the project report titled “Leveraging Generative AI for Intelligent Source Code Analysis and Vulnerability Detection” is a bonafide work carried out by KEERTHANA L (DA2232306010039) under my supervision for the award of the Degree of Bachelor of Computer Applications. To my knowledge the work reported herein is the original work done by these students.

|  |  |
| --- | --- |
| **Mr. J. Venkata Subramanian**  Assistant Professor,  Department of Computer Applications  (GUIDE) | **Dr. S. Albert Antony Raj**  Deputy Dean & Head,  Department of Computer Applications |
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

**ACKNOWLEDGEMENT**

With profound gratitude to the ALMIGHTY, I take this chance to thank the people who helped me to complete this project.

We take this as a right opportunity to say THANKS to my parents who are there to stand with me always with the words “YOU CAN”.

We are thankful to Dr.T.R.Paarivendhar, Chancellor, SRM Institute of Science & Technology who gave us the platform to establish me to reach greater heights.

We earnestly thank Dr. A.Duraisamy, Dean, College of Science and Humanities, SRM Institute of Science & Technology who always encourage us to do novel things.

We express our sincere thanks to Dr. S. Albert Antony Raj Deputy Dean and Head, Department of Computer Applications, College of Science and Humanities for his valuable guidance and support to execute all incline in learning.

It is our delight to thank our project guide Mr. J.Venkata Subramanian Assistant Professor, Department of Computer Applications for his help, support, encouragement, suggestions, and guidance throughout the development phases of the project.

We convey our gratitude to all the faculty members of the department who extended their support through valuable comments and suggestions during the reviews.

A great note of gratitude to friends and people who are known and unknown to me who helped in carrying out this project work a successful one.

**KEERTHANA L**

**TABLE OF CONTENTS**

[1. INTRODUCTION 1](#_Toc115934)

[2. SOFTWARE REQUIREMENT ANALYSIS 1](#_Toc115935)

[2.1 HARDWARE SPECIFICATION 2](#_Toc115936)

[2.2 SOFTWARE SPECIFICATION 2](#_Toc115937)

[2.3 ABOUT THE SOFTWARE AND ITS FEATURE 3](#_Toc115938)

[3. SYSTEM ANALYSIS 13](#_Toc115939)

[3.1 EXISTING SYSTEM 13](#_Toc115940)

[3.2 PROPOSED SYSTEM 13](#_Toc115941)

[3.3 FEASIBILITY STUDY 14](#_Toc115942)

[4. SYSTEM DESIGN 15](#_Toc115943)

[4.1 DATA FLOW DIAGARAM 16](#_Toc115944)

[4.2 UML DIAGRAMS **Error! Bookmark not defined.**](#_Toc115945)

[4.3 DATABASE DESIGN 18](#_Toc115946)

[5. CODE TEMPLATES Error! Bookmark not defined.](#_Toc115947)

[5.1 MODULE DESCRIPTION **Error! Bookmark not defined.**](#_Toc115948)

[5.2 TABLES **Error! Bookmark not defined.**](#_Toc115949)

[6. TESTING Error! Bookmark not defined.](#_Toc115950)

[6.1 TESTING METHODOLOGIES **Error! Bookmark not defined.**](#_Toc115951)

[6.2 TEST CASE **Error! Bookmark not defined.**](#_Toc115952)

[7. OUTPUT SCREENS 20](#_Toc115953)

[8. CONCLUSION 46](#_Toc115954)

[9. FURTHER ENHANCEMENTS 46](#_Toc115955)

[10. REFERENCES 48](#_Toc115956)

[11. APPENDICES 48](#_Toc115957)

[11.1 USER DOCUMENTATION **Error! Bookmark not defined.**](#_Toc115958)

[11.2 README 48](#_Toc115959)

[11.3 SAMPLE SOURCE PROGRAM 49](#_Toc115960)

**ABSTRACT**

The ever-expanding realm of software development presents a double-edged sword. On one hand, it fosters innovation and drives technological advancements across various industries. From healthcare and finance to entertainment and social media, software underpins the functionality of countless applications and services that have become integral to our daily lives. On the other hand, the growing complexity of codebases and the relentless evolution of cyber threats necessitate the development of more sophisticated and efficient methods for safeguarding software security. Traditional approaches to source code analysis, often relying on manual code reviews and static analysis tools, can be time-consuming, labor-intensive, and may not always be comprehensive enough to catch subtle vulnerabilities. As software development cycles become increasingly rapid, there's a growing need for automated and intelligent solutions to identify and address potential issues within the code itself.

This project delves into the exciting potential of available generative AI models to automate and augment the process of source code analysis. Generative AI models are a class of machine learning algorithms trained on massive amounts of text data, enabling them to generate human-quality text, translate languages, write different kinds of creative content, and answer your questions in an informative way. By leveraging these capabilities, we can empower generative AI models to analyze source code, identify potential security vulnerabilities, assess code quality, and even suggest improvements and refactorings. This has the potential to revolutionize the way developers approach code security and quality assurance, enabling them to build more secure and reliable software applications in a shorter time frame.

# 1. INTRODUCTION

In the dynamic landscape of software development, innovation is both a driving force and a constant challenge. As technology advances, so too do the complexities inherent in the codebases that underpin our digital infrastructure. From the vital systems supporting healthcare and finance to the entertainment and social media platforms we engage with daily, software forms the backbone of modern society. However, with this ubiquity comes a pressing need for robust security measures to safeguard against ever-evolving cyber threats.

Traditional approaches to ensuring software security have often relied on manual code reviews and static analysis tools. While these methods have their merits, they can be time-consuming, labor-intensive, and prone to oversight. As development cycles accelerate and codebases grow more intricate, the demand for automated, intelligent solutions has never been greater.

Enter the realm of generative AI models – sophisticated algorithms trained on vast datasets of text, capable of generating human-like language and performing a myriad of tasks, from translation to creative writing. Leveraging the power of these models, we can embark on a journey to revolutionize source code analysis.

This project explores the transformative potential of generative AI in automating and augmenting the process of source code analysis. By harnessing the innate abilities of these models, we can empower them to scrutinize code for potential security vulnerabilities, evaluate code quality, and even propose optimizations and refactorings.

In doing so, we pave the way for a new era in software development, one where developers can harness the intelligence of AI to build more secure, reliable applications in a fraction of the time. Join us as we delve into the exciting possibilities that lie at the intersection of AI and software security, where innovation knows no bounds.

Top of Form

Bottom of Form

# 2. SOFTWARE REQUIREMENT ANALYSIS

## 2.1 HARDWARE SPECIFICATION

|  |  |
| --- | --- |
| OS | Windows: 10 or newer  MAC: OS X v10.7 or higher  Linux: Ubuntu |
| Processor | 2GHz or more |
| RAM | 2.00 GB or above |
| Hard Disk | 64 GB or more |
| Network | Ethernet connection (LAN) OR |

a wireless adapter (Wi-Fi)

## 2.2 SOFTWARE SPECIFICATION

|  |  |
| --- | --- |
| IDE | Visual Studio Code |
| Web Server | JSON |
| Front End | python |
| Database | MySQL |
| Browser | Chrome |

## 2.3 ABOUT THE SOFTWARE AND ITS FEATURE

The software is a unique blend of technologies tailored to offer an innovative approach to web development. Here's an overview of its features:

1. **Visual Studio Code (IDE):**
   * Provides a versatile and powerful code editor with support for various programming languages and extensions.
   * Offers features like syntax highlighting, code completion, debugging, and version control integration, enhancing developer productivity.
2. **JSON (Web Server):**
   * JSON (JavaScript Object Notation) is used as the web server technology.
   * JSON is a lightweight data interchange format commonly used for transmitting data between a server and a web application.
3. **Python (Front End):**
   * Python is utilized for front-end development, offering flexibility and ease of use in creating dynamic and interactive user interfaces.
   * With frameworks like Flask or Django, Python can handle server-side rendering, routing, and integrating with backend services.
4. **MySQL (Database):**
   * MySQL is employed as the database management system for storing and managing structured data.
   * It offers features such as SQL querying, data manipulation, and transaction support, essential for efficient data management in web applications.
5. **Chrome (Browser):**
   * Chrome is the recommended browser for testing and deploying the software due to its extensive developer tools and market share.
   * It provides a robust environment for debugging, profiling, and optimizing web applications during development and testing phases.

**Key Features:**

* **Flexibility:** The software offers flexibility in choosing technologies for different components, allowing developers to leverage the strengths of each technology stack.
* **Ease of Use:** Visual Studio Code provides a user-friendly environment for coding and debugging, while Python's simplicity enhances frontend development.
* **Scalability:** MySQL ensures scalability and reliability in managing large volumes of data, making it suitable for web applications with growing user bases.
* **Interactivity:** With Python powering the front end, developers can create highly interactive and responsive user interfaces, enhancing the user experience.

Overall, the software offers a modern and innovative approach to web development, combining industry-standard tools with emerging technologies to create dynamic and engaging web applications.

Top of Form

Bottom of Form

1. **Adaptability to Diverse Environments:**
   * Flexible vulnerability framing allows developers to adapt their security measures to diverse environments, including different operating systems, network configurations, and deployment platforms. This adaptability ensures that security measures remain effective across various contexts without compromising functionality.
2. **Customization for Specific Threat Landscapes:**
   * Every application faces unique security threats based on factors such as its intended use, target audience, and industry regulations. Flexible vulnerability framing enables developers to customize security measures to address specific threat landscapes, ensuring that the most relevant vulnerabilities are prioritized and mitigated effectively.
3. **Integration with Varied Development Workflows:**
   * Development teams often employ diverse workflows, tools, and methodologies to manage projects efficiently. Flexible vulnerability framing allows security practices to be integrated seamlessly into various development workflows, whether agile, DevOps, or traditional, ensuring that security considerations are prioritized throughout the development lifecycle.
4. **Scalability Across Different Project Sizes:**
   * Flexibility in vulnerability framing enables security practices to scale effectively across projects of different sizes and complexities. Whether developing a small-scale application or a large enterprise system, developers can tailor security measures to suit the specific requirements and constraints of each project without sacrificing effectiveness.
5. **Alignment with Evolving Threat Landscape:**
   * The cybersecurity landscape is constantly evolving, with new vulnerabilities and attack vectors emerging regularly. Flexible vulnerability framing allows developers to stay ahead of these evolving threats by adapting security measures in real-time, incorporating the latest threat intelligence, and implementing proactive defenses to mitigate potential risks effectively.
6. **Compliance with Regulatory Requirements:**
   * Different industries and regions have specific regulatory requirements regarding data protection, privacy, and cybersecurity. Flexible vulnerability framing enables developers to align their security practices with these regulatory requirements, ensuring compliance while also addressing industry-specific vulnerabilities and best practices.
7. **Balancing Security and Usability:**
   * Security measures should not impede usability or hinder user experience unnecessarily. Flexible vulnerability framing allows developers to strike the right balance between security and usability by identifying and mitigating vulnerabilities without introducing unnecessary complexity or friction in the user interaction flow.
8. **Continuous Improvement through Feedback Loops:**
   * Flexible vulnerability framing facilitates continuous improvement through feedback loops, allowing developers to gather insights from security incidents, penetration testing, and user feedbackTop of Form

Bottom of Form

#### 

**PYTHON:**

### History and Evolution of Python

Python, created by Guido van Rossum in the late 1980s, was conceived as a successor to the ABC language. Its development was motivated by a desire for a language that was easy to read and write. The first version, Python 0.9.0, was released in 1991, followed by Python 2.0 in 2000, which introduced features like garbage collection and list comprehensions. Python 3.0, a major overhaul, was released in 2008, aiming to clean up the language and resolve inconsistencies. The transition from Python 2.x to Python 3.x marked a significant milestone in Python's evolution, emphasizing modern features and improvements in performance and syntax.

### Core Features of Python

Python's popularity stems from its simplicity, readability, and versatility. It features a clean and straightforward syntax that resembles pseudo-code, making it accessible for beginners and conducive to rapid development. Key language features include dynamic typing, automatic memory management (via garbage collection), and support for multiple programming paradigms such as procedural, object-oriented, and functional programming. Python's design philosophy emphasizes code readability through the use of whitespace indentation to define code blocks, reducing syntactic clutter and promoting a clear and structured coding style.

### Syntax and Language Constructs

Python's syntax is designed to be intuitive and expressive, focusing on readability and reducing the time spent debugging and maintaining code. It supports a wide range of data types including integers, floats, strings, lists, tuples, dictionaries, and sets. Control flow constructs like if-else statements, loops (for and while), and exception handling facilitate robust error management and program control. Python's functions are first-class citizens, allowing functions to be assigned to variables, passed as arguments to other functions, and returned as values. Advanced language features such as generators, decorators, context managers, and comprehensions further enhance Python's capabilities, enabling concise and powerful code expressions.

### Versatility and Applications

Python's versatility is reflected in its extensive usage across various domains and industries:

* **Web Development:** Python frameworks like Django and Flask are widely used for backend web development, offering robust solutions for building scalable and secure web applications. Libraries such as BeautifulSoup and requests simplify web scraping and interaction with web APIs.
* **Data Science and Machine Learning:** Python has emerged as a dominant language in data science and machine learning due to libraries like NumPy (for numerical computing), pandas (for data manipulation), Matplotlib and Seaborn (for data visualization), and scikit-learn (for machine learning algorithms). Frameworks like TensorFlow and PyTorch provide powerful tools for building and training deep learning models.
* **Scientific Computing:** Python's ecosystem includes libraries like SciPy, which extends Python's capabilities with modules for scientific computing, including optimization, integration, linear algebra, and signal processing.
* **Artificial Intelligence and Natural Language Processing:** Python is widely used in AI and NLP applications with libraries like NLTK (Natural Language Toolkit), spaCy, and Gensim, which provide tools for processing and analyzing text data, building language models, and performing sentiment analysis and text classification.
* **Desktop GUIs and Software Development:** Python's Tkinter library enables developers to create cross-platform desktop GUI applications with ease. Other libraries like PyQt and wxPython offer additional options for GUI development.
* **Game Development:** Python is utilized in game development through frameworks like Pygame, which simplifies the creation of 2D games and interactive multimedia applications.
* **System Administration and Scripting:** Python's simplicity and versatility make it ideal for writing system scripts, automation tasks, and managing infrastructure. Tools like Fabric and Ansible leverage Python's capabilities for configuration management and deployment automation.

### Python Ecosystem

Python's ecosystem encompasses a broad array of resources and tools that enhance development productivity and support:

* **Standard Library:** Python's standard library provides a comprehensive set of modules for tasks such as file I/O, networking, threading, database access, and more. It follows the "batteries included" philosophy, offering built-in solutions for common programming tasks without requiring external dependencies.
* **Third-Party Packages:** The Python Package Index (PyPI) hosts a vast repository of third-party libraries and packages contributed by the Python community. These packages extend Python's functionality across diverse domains, from scientific computing and data analysis to web development, machine learning, and beyond. Popular packages include requests for HTTP, SQLAlchemy for database interaction, pandas for data manipulation, and cryptography for secure communication.
* **Development Tools:** Python developers benefit from a variety of integrated development environments (IDEs) and editors that provide robust tools for writing, debugging, and testing Python code. Popular IDEs include PyCharm, Visual Studio Code, and Spyder, while editors like Sublime Text and Atom offer lightweight alternatives. Jupyter notebooks provide an interactive environment for data analysis, visualization, and collaborative coding.
* **Community and Support:** Python boasts a vibrant and active community of developers, educators, and enthusiasts who contribute to its growth and evolution. Online forums like Stack Overflow, Python mailing lists, and community-driven conferences (e.g., PyCon) provide platforms for sharing knowledge, seeking help, and networking. The Python Software Foundation (PSF) oversees Python's development, promotes its use, and supports the Python community through grants, sponsorships, and educational initiatives.

### Educational Impact

Python's readability, simplicity, and extensive support make it an ideal language for teaching and learning programming:

* **Beginner-Friendly:** Python's clear and concise syntax makes it accessible for beginners, allowing them to focus on programming concepts rather than language intricacies. Its interactive interpreter facilitates experimentation and learning by immediate feedback.
* **Widely Adopted in Education:** Python is widely adopted in educational institutions, from secondary schools to universities, as a primary language for introductory programming courses and advanced topics such as data science, artificial intelligence, and scientific computing. Its educational resources, including tutorials, documentation, and community-driven initiatives like Python Software Foundation's educational outreach programs, support educators and learners worldwide.
* **Cross-Disciplinary Applications:** Python's versatility enables students and researchers across disciplines—from humanities and social sciences to engineering and natural sciences—to leverage programming for data analysis, simulation, visualization, and research.

### Future Trends and Innovations

Python continues to evolve with regular updates and advancements that address emerging trends and technological innovations:

* **Python 3.x Series:** Python 3.x series, with the latest stable release (as of the knowledge cutoff date), introduces improvements in performance, security, and syntax enhancements. Python's commitment to backward compatibility ensures smooth transitions for developers upgrading from previous versions.
* **AI and Machine Learning:** Python's dominance in AI, machine learning, and deep learning is expected to grow, fueled by advancements in frameworks like TensorFlow, PyTorch, and libraries supporting model deployment and production.
* **Web Development:** Python frameworks like Django and Flask will continue to innovate, addressing scalability, security, and performance requirements for web applications. Integration with emerging technologies such as GraphQL and serverless computing (e.g., AWS Lambda) will shape future web development practices.
* **Data Science and Visualization:** Python's ecosystem for data science and visualization will expand with advancements in libraries like pandas, NumPy, and tools for interactive data exploration and storytelling (e.g., Dash and Streamlit).
* **Community and Open Source:** Python's open-source nature and active community involvement will drive continued innovation, with contributions from developers worldwide shaping the language's evolution and expanding its capabilities across new domains and use cases.

### Conclusion

Python's journey from its humble beginnings to becoming one of the most widely used programming languages globally is a testament to its design philosophy, versatility, and community support. Its adoption spans industries and disciplines, from web development and data science to scientific computing, AI, and beyond. As Python continues to evolve and adapt to technological advancements, its impact on software development, education, research, and innovation will remain profound, shaping the future of programming and computing landscapes worldwide.

Top of Form

Bottom of Form

#### 2.3.2 VISUALIZED PROGRESS

Visualized progress in software development can take various forms, each providing valuable insights into the project's status, milestones, and performance. Here are a few examples:

1. **Gantt Charts:**
   * Gantt charts provide a visual representation of project timelines, tasks, and dependencies. They allow stakeholders to see the overall progress of the project, identify critical path tasks, and track milestones over time. Color coding and progress bars help visualize completed, ongoing, and upcoming tasks, providing a clear overview of project status.
2. **Burndown Charts:**
   * Burndown charts depict the progress of a project over time by tracking the completion of tasks or work items against a predefined timeline. They show the remaining effort needed to complete the project and help teams gauge their pace and adjust their efforts accordingly. Burndown charts are commonly used in agile project management to monitor sprint progress and predict project completion dates.
3. **Kanban Boards:**
   * Kanban boards visualize work items as cards moving through different stages of a workflow, such as "To Do," "In Progress," and "Done." They provide a real-time overview of task status, bottlenecks, and work in progress limits. Teams can easily track progress, prioritize tasks, and identify areas for improvement by visualizing the flow of work through the system.
4. **Dashboard Metrics:**
   * Dashboards aggregate key project metrics and KPIs into visual displays, such as charts, graphs, and widgets. They provide stakeholders with a holistic view of project health, including metrics like sprint velocity, burnup/burndown, defect rates, and team performance. Dashboards enable quick decision-making by presenting data in an easily digestible format.
5. **Heatmaps:**
   * Heatmaps visualize data density and distribution using color gradients, allowing teams to identify patterns, trends, and outliers. In software development, heatmaps can be used to visualize code complexity, test coverage, defect density, and user engagement metrics. They provide valuable insights into areas that require attention or optimization.
6. **Flowcharts:**
   * Flowcharts depict the sequence of steps or processes in a graphical format, helping teams understand workflows, decision points, and dependencies. They can be used to visualize software development processes, such as release pipelines, deployment workflows, or testing procedures. Flowcharts aid in identifying bottlenecks, optimizing processes, and streamlining workflows.
7. **Cycle Time Histograms:**
   * Cycle time histograms display the distribution of time taken to complete individual tasks or user stories. They help teams understand the variability in task completion times, identify outliers, and optimize workflow efficiency. Cycle time histograms are particularly useful in agile and lean methodologies for identifying opportunities to reduce lead times and improve throughput.

#### 2.3.3 INTELLIGENT SUGGESTION

#### Intelligent suggestion systems leverage data analysis and machine learning to provide personalized recommendations tailored to individual users' needs and contexts. These systems can suggest code improvements, identify potential bugs, recommend refactoring opportunities, and optimize development processes, enhancing productivity and code quality in software development.

# 3. SYSTEM ANALYSIS

## 3.1 EXISTING SYSTEM

The current system lacks a centralized platform for managing software development projects effectively. Development teams often rely on disparate tools and manual processes for project planning, task tracking, code collaboration, and issue management, leading to inefficiencies and coordination challenges. Communication gaps and version control issues are common, hindering collaboration and slowing down the development cycle. Additionally, the absence of automated analysis tools results in a reactive approach to identifying and addressing code quality issues and vulnerabilities, increasing the risk of defects and security breaches.

## 3.2 PROPOSED SYSTEM

The proposed system aims to address the shortcomings of the existing system by introducing a comprehensive software development platform that integrates project management, version control, code collaboration, and automated analysis tools. The system will provide a centralized repository for storing code, documents, and project artifacts, facilitating seamless collaboration and version control across distributed development teams. Automated code analysis tools will enable proactive identification of code quality issues, security vulnerabilities, and performance bottlenecks, empowering developers to address them early in the development process. Real-time communication features and task tracking capabilities will enhance team collaboration and project visibility, streamlining the development workflow and accelerating delivery.

## 3.3 FEASIBILITY STUDY

#### A feasibility study will be conducted to assess the viability and practicality of implementing the proposed system. This study will evaluate various aspects, including technical feasibility, economic feasibility, and operational feasibility. Technical feasibility will involve assessing the compatibility of existing infrastructure and technologies with the proposed system requirements, identifying any potential technical challenges or limitations, and determining the feasibility of integrating new tools and technologies. Economic feasibility will involve analyzing the costs and benefits associated with implementing the proposed system, including initial investment, ongoing maintenance, and potential cost savings from increased productivity and reduced rework. Operational feasibility will involve evaluating the impact of the proposed system on existing processes, workflows, and organizational culture, assessing the readiness of stakeholders to adopt new tools and practices, and identifying any barriers to implementation.

#### 3.3.1 FEASIBILITY SYSTEM

#### The feasibility system will involve conducting a pilot implementation of the proposed system in a controlled environment to evaluate its performance, usability, and effectiveness in real-world scenarios. This pilot implementation will involve selecting a representative sample of development teams and projects to participate in the pilot, providing training and support to users, and collecting feedback and performance metrics to assess the system's impact. Based on the results of the feasibility study and pilot implementation, recommendations will be made regarding the feasibility of proceeding with full-scale implementation of the proposed system, including any necessary adjustments or modifications to the system design and implementation plan.

#### 3.3.2 ECONOMIC FEASIBILITY

#### Economic feasibility assesses the financial viability of implementing the proposed system. This involves analyzing the costs associated with development, deployment, and maintenance against the potential benefits and cost savings. Factors such as initial investment in software, hardware, training, and ongoing operational costs need to be considered. Additionally, the potential return on investment (ROI) from increased productivity, reduced rework, and improved software quality should be evaluated. Economic feasibility also involves comparing the costs and benefits of the proposed system with alternative solutions or maintaining the status quo.

#### 3.3.3 TECHNICAL FEASIBILITY

#### Technical feasibility evaluates whether the proposed system can be implemented using existing technology infrastructure and resources. This involves assessing the compatibility of the proposed system with the organization's current hardware, software, and network environment. Compatibility with existing databases, programming languages, development tools, and integration capabilities needs to be considered. Additionally, technical feasibility involves evaluating any potential technical challenges or limitations, such as scalability, performance, security, and interoperability issues. Mitigation strategies for addressing these challenges should be identified to ensure the successful implementation of the proposed system.

#### 3.3.4 OPERATIONAL FEASIBILITY

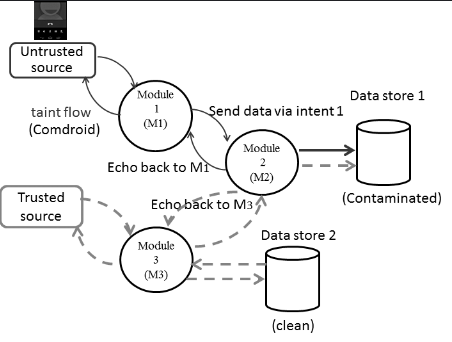
Operational feasibility assesses the practicality and effectiveness of implementing the proposed system within the organization's existing operational environment. This involves evaluating how well the proposed system aligns with existing processes, workflows, and organizational culture. Factors such as user acceptance, training needs, change management requirements, and workflow integration should be considered. Operational feasibility also involves identifying any potential barriers or resistance to adoption from stakeholders and developing strategies to address them. Ultimately, the goal is to ensure that the proposed system can be seamlessly integrated into the organization's operations and deliver the expected benefits without disrupting existing workflows.

Top of Form

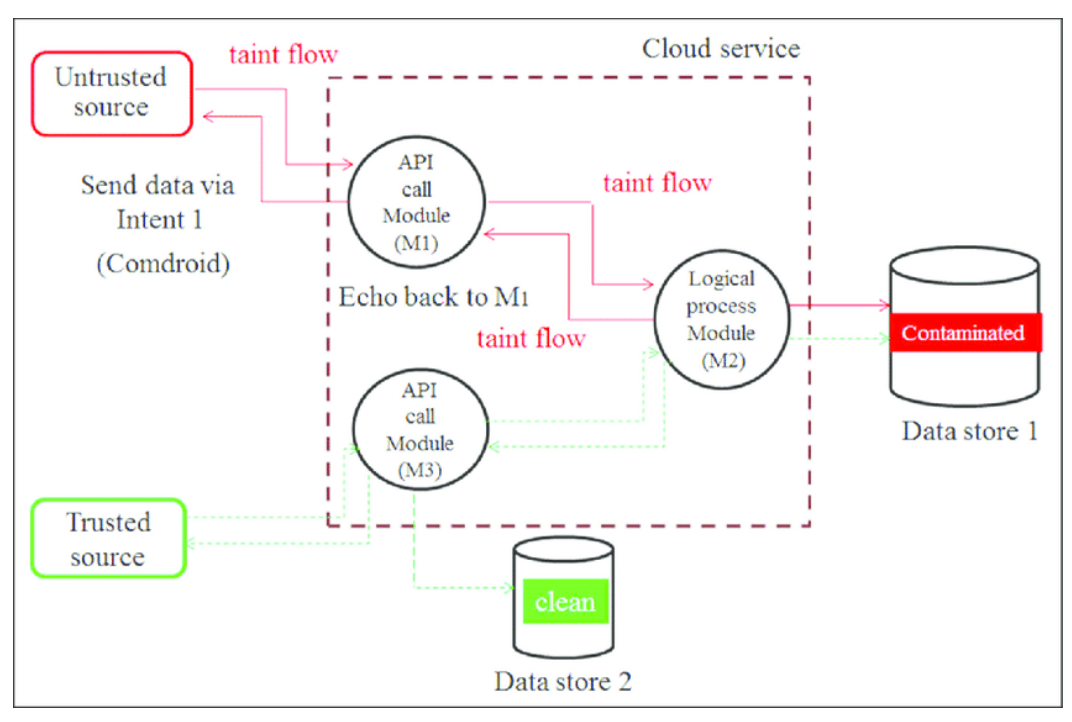
Bottom of Form

# 4. SYSTEM DESIGN

## 4.1 DATA FLOW DIAGARAM



**4.2.2 ACIVITY DIAGRAM**



## 4.3 DATABASE DESIGN

**1. Requirements Analysis:**

* Begin by thoroughly understanding the data requirements of your project. Identify the entities involved, their attributes, relationships, and any constraints that need to be enforced. This step is crucial in determining the scope and structure of your database.

**2. Conceptual Design:**

* Develop a high-level conceptual model that represents the key entities and relationships in your system. Utilize techniques like Entity-Relationship Diagrams (ERDs) to visualize the conceptual schema, focusing on the logical organization of the data without delving into implementation details.

**3. Logical Design:**

* Translate the conceptual model into a logical schema suitable for implementation in a database management system (DBMS). Define tables, columns, primary keys, foreign keys, and relationships based on the requirements identified in the previous phase. Ensure data integrity and normalization to eliminate redundancy and dependency anomalies.

**4. Physical Design:**

* Refine the logical schema to optimize performance, storage, and access patterns. Consider factors such as indexing, partitioning, clustering, and storage allocation based on anticipated workload and hardware specifications. Aim to minimize storage overhead and improve query performance through efficient data organization.

**5. Normalization:**

* Apply normalization techniques to eliminate data redundancy and dependency anomalies in your database schema. Decompose tables into smaller, related tables and define appropriate relationships to ensure data integrity and consistency. Normalization helps in maintaining a flexible and efficient database structure.

**6. Data Security and Integrity:**

* Implement robust security measures to protect your database from unauthorized access and ensure data integrity. Define access controls, encryption mechanisms, and data validation rules to enforce security policies. Use constraints like unique constraints, foreign key constraints, and check constraints to maintain data consistency and accuracy.

**7. Data Migration and Integration:**

* Plan for seamless data migration from existing systems and integration with other systems. Develop strategies for importing existing data into the new database schema while ensuring compatibility and consistency. Ensure data consistency and interoperability between different data sources and formats to facilitate smooth data exchange.

# OUTPUT SCREENS

The Vulnerable code that we’re inputing:

?php

define( 'DVWA\_WEB\_PAGE\_TO\_ROOT', '../../' );

require\_once DVWA\_WEB\_PAGE\_TO\_ROOT . 'dvwa/includes/dvwaPage.inc.php';

dvwaPageStartup( array( 'authenticated' ) );

$page = dvwaPageNewGrab();

$page[ 'title' ]   = 'Vulnerability: SQL Injection' . $page[ 'title\_separator' ].$page[ 'title' ];

$page[ 'page\_id' ] = 'sqli';

$page[ 'help\_button' ]   = 'sqli';

$page[ 'source\_button' ] = 'sqli';

dvwaDatabaseConnect();

$method            = 'GET';

$vulnerabilityFile = '';

switch( dvwaSecurityLevelGet() ) {

    case 'low':

        $vulnerabilityFile = 'low.php';

        break;

    case 'medium':

        $vulnerabilityFile = 'medium.php';

        $method = 'POST';

        break;

    case 'high':

        $vulnerabilityFile = 'high.php';

        break;

    default:

        $vulnerabilityFile = 'impossible.php';

        break;

}

require\_once DVWA\_WEB\_PAGE\_TO\_ROOT . "vulnerabilities/sqli/source/{$vulnerabilityFile}";

$page[ 'body' ] .= "

<div class=\"body\_padded\">

    <h1>Vulnerability: SQL Injection</h1>

    <div class=\"vulnerable\_code\_area\">";

if( $vulnerabilityFile == 'high.php' ) {

    $page[ 'body' ] .= "Click <a href=\"#\" onclick=\"javascript:popUp('session-input.php');return false;\">here to change your ID</a>.";

}

else {

    $page[ 'body' ] .= "

        <form action=\"#\" method=\"{$method}\">

            <p>

                User ID:";

    if( $vulnerabilityFile == 'medium.php' ) {

        $page[ 'body' ] .= "\n              <select name=\"id\">";

        for( $i = 1; $i < $number\_of\_rows + 1 ; $i++ ) { $page[ 'body' ] .= "<option value=\"{$i}\">{$i}</option>"; }

        $page[ 'body' ] .= "</select>";

    }

    else

        $page[ 'body' ] .= "\n              <input type=\"text\" size=\"15\" name=\"id\">";

    $page[ 'body' ] .= "\n              <input type=\"submit\" name=\"Submit\" value=\"Submit\">

            </p>\n";

    if( $vulnerabilityFile == 'impossible.php' )

        $page[ 'body' ] .= "            " . tokenField();

    $page[ 'body' ] .= "

        </form>";

}

$page[ 'body' ] .= "

        {$html}

    </div>

    <h2>More Information</h2>

    <ul>

        <li>" . dvwaExternalLinkUrlGet( 'https://en.wikipedia.org/wiki/SQL\_injection' ) . "</li>

        <li>" . dvwaExternalLinkUrlGet( 'https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/' ) . "</li>

        <li>" . dvwaExternalLinkUrlGet( 'https://owasp.org/www-community/attacks/SQL\_Injection' ) . "</li>

        <li>" . dvwaExternalLinkUrlGet( 'https://bobby-tables.com/' ) . "</li>

    </ul>

</div>\n";

dvwaHtmlEcho( $page );

?>

The Code Vulnerabilities that the system detecting:

{

    "vulnerabilities": [

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "35.",

            "Code": "$page[ 'body' ] .= \"\\n\t\t\t\tUser ID:\"; if( $vulnerabilityFile == 'medium.php' ) {",

            "Explanation": "The User ID input field in the form is vulnerable to SQL injection attacks as it directly inserts user input into the SQL query without proper sanitization or validation.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "Hardcoded Secrets/Keys.",

            "Severity": "Medium.",

            "Line": "7.",

            "Code": "require\_once DVWA\_WEB\_PAGE\_TO\_ROOT . 'dvwa/includes/dvwaPage.inc.php';",

            "Explanation": "This line contains a hardcoded path to include a file which may expose sensitive information such as directory structure or naming conventions. It is recommended to store such sensitive information securely and not expose them in the code.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "Insufficient Input Validation.",

            "Severity": "Medium.",

            "Line": "22.",

            "Code": "$method = 'GET';",

            "Explanation": "The 'method' variable is being set to 'GET' without proper validation or sanitization. This could potentially lead to unexpected behavior or security risks if the variable is manipulated by an attacker.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "Hardcoded Secrets/Keys.",

            "Severity": "Medium.",

            "Line": "39.",

            "Code": "if( $vulnerabilityFile == 'impossible.php' )",

            "Explanation": "The hardcoded comparison values like 'impossible.php' may expose internal implementation details and should be avoided to prevent potential security risks.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "Hardcoded Secrets/Keys.",

            "Severity": "Medium.",

            "Line": "41, 49, 60.",

            "Code": "$vulnerabilityFile = 'low.php'; $vulnerabilityFile = 'medium.php'; $vulnerabilityFile = 'high.php';",

            "Explanation": "The hardcoded filenames assigned to the 'vulnerabilityFile' variable could potentially expose information about the application's vulnerability structure and pose a security risk if intercepted by an attacker.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "Hardcoded Secrets/Keys.",

            "Severity": "Medium.",

            "Line": "53.",

            "Code": "$vulnerabilityFile = 'impossible.php';",

            "Explanation": "Hardcoding the filename 'impossible.php' could reveal the internal logic of the application, which is not an ideal practice from a security standpoint.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "Hardcoded Secrets/Keys.",

            "Severity": "Medium.",

            "Line": "13.",

            "Code": "$page[ 'title' ]   = 'Vulnerability: SQL Injection' . $page[ 'title\_separator' ].$page[ 'title' ];",

            "Explanation": "The hardcoded title 'Vulnerability: SQL Injection' in the page title may expose information about the security vulnerability and should be dynamically generated to avoid revealing such details.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "Hardcoded Secrets/Keys.",

            "Severity": "Medium.",

            "Line": "15, 16, 17, 18.",

            "Code": "$page[ 'page\_id' ] = 'sqli'; $page[ 'help\_button' ]   = 'sqli'; $page[ 'source\_button' ] = 'sqli';",

            "Explanation": "The hardcoded values 'sqli' assigned to various page identifiers could potentially disclose information about the vulnerable page and should be obfuscated to enhance security.",

            "file": "sqli\\index.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "11.",

            "Code": "$\_SESSION[ 'id' ] =  $\_POST[ 'id' ];",

            "Explanation": "The code directly takes user input from the POST request and stores it in the $\_SESSION variable without sanitization or validation. This can lead to SQL Injection vulnerabilities if the input is manipulated by an attacker. Attackers can potentially manipulate the input to execute malicious SQL queries on the database.",

            "file": "sqli\\session-input.php"

        },

        {

            "Vulnerability": "Cross-Site Scripting (XSS).",

            "Severity": "Medium.",

            "Line": "13.",

            "Code": "$page[ 'body' ] .= \"Session ID: {$\_SESSION[ 'id' ]}<br /><br /><br />\";",

            "Explanation": "The code directly outputs the value of the 'id' session variable without proper encoding or validation, which could allow an attacker to inject and execute malicious scripts in the context of other users' sessions. This can be exploited to steal cookies, session tokens, or perform other malicious actions.",

            "file": "sqli\\session-input.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High",

            "Line": "17",

            "Code": "<pre>Spoiler: <span class=\"spoiler\">?id=a' UNION SELECT \"text1\",\"text2\";-- -&Submit=Submit</span>.</pre>",

            "Explanation": "The code is vulnerable to SQL Injection as it is directly inserting user input into the SQL query without proper sanitization or parameterization. An attacker can manipulate the input to perform malicious SQL queries like UNION SELECT to extract sensitive data or perform other harmful actions on the database.",

            "file": "sqli\\help\\help.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "Medium",

            "Line": "34",

            "Code": "<pre>Spoiler: <span class=\"spoiler\">?id=a UNION SELECT 1,2;-- -&Submit=Submit</span>.</pre>",

            "Explanation": "The code is still susceptible to SQL Injection even though it uses mysql\_real\_escape\_string function, as the SQL query does not have quotes around the parameter, making it vulnerable to manipulation by attackers.",

            "file": "sqli\\help\\help.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High",

            "Line": "44",

            "Code": "<pre>Spoiler: <span class=\"spoiler\">ID: a' UNION SELECT \"text1\",\"text2\";-- -&Submit=Submit</span>.</pre>",

            "Explanation": "The code is vulnerable to SQL Injection as the input values are being transferred to the vulnerable query via session variables, allowing attackers to manipulate the input and execute malicious queries.",

            "file": "sqli\\help\\help.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "8.",

            "Code": "$id = $\_SESSION[ 'id' ];",

            "Explanation": "The variable $id is directly taken from the session without any validation or sanitization, making it vulnerable to SQL injection attacks when used in the SQL query on line 11.",

            "file": "sqli\\source\\high.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "11.",

            "Code": "$query  = \"SELECT first\_name, last\_name FROM users WHERE user\_id = '$id' LIMIT 1;\";",

            "Explanation": "The SQL query is constructed using the $id variable without sanitization, which opens up the possibility of SQL injection attacks if the $id variable is manipulated to include malicious SQL code.",

            "file": "sqli\\source\\high.php"

        },

        {

            "Vulnerability": "Lack of Input Validation.",

            "Severity": "Medium.",

            "Line": "5.",

            "Code": "$id = $\_SESSION[ 'id' ];",

            "Explanation": "The code does not perform any input validation on the user input stored in the $\_SESSION['id'] variable before using it directly in a SQL query, which can lead to security vulnerabilities like SQL injection.",

            "file": "sqli\\source\\high.php"

        },

        {

            "Vulnerability": "Lack of Output Escaping.",

            "Severity": "Medium.",

            "Line": "19.",

            "Code": "$html .= \"<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>\";",

            "Explanation": "The code does not escape the output values $id, $first, and $last when displaying them to the end user, which could lead to cross-site scripting (XSS) vulnerabilities if these values contain malicious scripts.",

            "file": "sqli\\source\\high.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "12.",

            "Code": "$id = $\_GET[ 'id' ];",

            "Explanation": "The code directly uses user input ($\_GET['id']) in the SQL query without proper sanitization or validation, making it vulnerable to SQL injection attacks.",

            "file": "sqli\\source\\impossible.php"

        },

        {

            "Vulnerability": "Lack of Input Validation.",

            "Severity": "Medium.",

            "Line": "9.",

            "Code": "checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );",

            "Explanation": "The input parameter 'user\_token' is being used without proper validation. Input validation should be implemented to avoid any potential security risks.",

            "file": "sqli\\source\\impossible.php"

        },

        {

            "Vulnerability": "Lack of Authorization.",

            "Severity": "Medium.",

            "Line": "32.",

            "Code": "generateSessionToken();",

            "Explanation": "The code seems to generate an Anti-CSRF token without any proper authorization or validation process, which could lead to potential security vulnerabilities.",

            "file": "sqli\\source\\impossible.php"

        },

        {

            "Vulnerability": "SQL Injection (SQLi).",

            "Severity": "High.",

            "Line": "7-10.",

            "Code": "$id = $\_REQUEST[ 'id' ];\n\t$query  = \"SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';\";\n\t$result = mysqli\_query($GLOBALS[\"\_\_\_mysqli\_ston\"],  $query ) or die( '<pre>' . ((is\_object($GLOBALS[\"\_\_\_mysqli\_ston\"])) ? mysqli\_error($GLOBALS[\"\_\_\_mysqli\_ston\"]) : (($\_\_\_mysqli\_res = mysqli\_connect\_error()) ? $\_\_\_mysqli\_res : false)) . '</pre>' );",

            "Explanation": "The code directly concatenates user input ($\_REQUEST['id']) into the SQL query without proper sanitization or parameterization, making it vulnerable to SQL injection attacks.",

            "file": "sqli\\source\\low.php"

        },

        {

            "Vulnerability": "Hardcoded Secrets.",

            "Severity": "Medium.",

            "Line": "14.",

            "Code": "case MYSQL:",

            "Explanation": "The case for the MySQL database contains a reference to $\_DVWA['SQLI\_DB'] which appears to be a configuration setting. Depending on the value of this setting, the code interacts with different database types. The hardcoded reference to MySQL could potentially leak information about the application's configuration or setup.",

            "file": "sqli\\source\\low.php"

        },

        {

            "Vulnerability": "Commented Out Code.",

            "Severity": "Low.",

            "Line": "21-22.",

            "Code": "#$sqlite\_db\_connection = new SQLite3($\_DVWA['SQLITE\_DB']);\n\t#$sqlite\_db\_connection->enableExceptions(true);",

            "Explanation": "The commented out code suggests that SQLite3 connection and exception handling code may not be in use or properly configured, which could impact error handling and possibly security in the application.",

            "file": "sqli\\source\\low.php"

        },

        {

            "Vulnerability": "Unused Code.",

            "Severity": "Low.",

            "Line": "23-24.",

            "Code": "$query  = \"SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';\";\n\t#print $query;",

            "Explanation": "The code to print the query is commented out and unused. While not a critical vulnerability, it adds to code clutter and may increase confusion during maintenance.",

            "file": "sqli\\source\\low.php"

        },

        {

            "Vulnerability": "SQL Injection (SQLi).",

            "Severity": "High.",

            "Line": "6.",

            "Code": "$id = $\_POST[ 'id' ];",

            "Explanation": "The code directly takes user input from the POST request without any input validation or sanitization, making it vulnerable to SQL Injection attacks. Although mysqli\_real\_escape\_string is used on line 8, it is not a reliable method to prevent SQL injection vulnerabilities. It's recommended to use parameterized queries or prepared statements to securely handle user input.",

            "file": "sqli\\source\\medium.php"

        },

        {

            "Vulnerability": "Hardcoded Database Connection Information.",

            "Severity": "Medium.",

            "Line": "25.",

            "Code": "$GLOBALS[\"\_\_\_mysqli\_ston\"].",

            "Explanation": "The code contains a hardcoded database connection variable \"$GLOBALS[\"\_\_\_mysqli\_ston\"]\", which may expose sensitive database credentials if the code is publicly accessible. It is recommended to store sensitive information like database credentials in secure configuration files outside of the web root directory.",

            "file": "sqli\\source\\medium.php"

        },

        {

            "Vulnerability": "Lack of CSRF Protection.",

            "Severity": "Medium.",

            "Line": "4.",

            "Code": "isset( $\_POST[ 'Submit' ] ).",

            "Explanation": "The code does not include any protection against Cross-Site Request Forgery (CSRF) attacks. Adding CSRF tokens to validate the origin of the requests can help mitigate this vulnerability.",

            "file": "sqli\\source\\medium.php"

        },

        {

            "Vulnerability": "Error Handling Disclosure.",

            "Severity": "Low.",

            "Line": "52.",

            "Code": "echo \"Error in fetch \".$sqlite\_db->lastErrorMsg();",

            "Explanation": "The code directly outputs error messages to the end-users, which can potentially expose sensitive information about the application's backend. It's recommended to handle errors more gracefully without revealing such details to the users.",

            "file": "sqli\\source\\medium.php"

        },

        {

            "Vulnerability": "Resource Exhaustion.",

            "Severity": "Medium.",

            "Line": "22.",

            "Code": "global $sqlite\_db\_connection;",

            "Explanation": "The code does not limit the amount of data that can be retrieved from the database, which could lead to resource exhaustion issues if the query results in a large dataset. Implementing pagination or setting appropriate limits can help prevent this vulnerability.",

            "file": "sqli\\source\\medium.php"

        }

    ]

}

Details of Vulnerabilities explained by system:

####################################################################################################

File: sqli\index.php

Vulnerability: SQL Injection. Severity: High. Line: 35. Code: $page[ 'body' ] .= "\n User ID:"; if( $vulnerabilityFile == 'medium.php' ) { Explanation: The User ID input field in the form is vulnerable to SQL injection attacks as it directly inserts user input into the SQL query without proper sanitization or validation.

Vulnerability: Hardcoded Secrets/Keys. Severity: Medium. Line: 7. Code: require\_once DVWA\_WEB\_PAGE\_TO\_ROOT . 'dvwa/includes/dvwaPage.inc.php'; Explanation: This line contains a hardcoded path to include a file which may expose sensitive information such as directory structure or naming conventions. It is recommended to store such sensitive information securely and not expose them in the code.

Vulnerability: Insufficient Input Validation. Severity: Medium. Line: 22. Code: $method = 'GET'; Explanation: The 'method' variable is being set to 'GET' without proper validation or sanitization. This could potentially lead to unexpected behavior or security risks if the variable is manipulated by an attacker.

Vulnerability: Hardcoded Secrets/Keys. Severity: Medium. Line: 39. Code: if( $vulnerabilityFile == 'impossible.php' ) Explanation: The hardcoded comparison values like 'impossible.php' may expose internal implementation details and should be avoided to prevent potential security risks.

Vulnerability: Hardcoded Secrets/Keys. Severity: Medium. Line: 41, 49, 60. Code: $vulnerabilityFile = 'low.php'; $vulnerabilityFile = 'medium.php'; $vulnerabilityFile = 'high.php'; Explanation: The hardcoded filenames assigned to the 'vulnerabilityFile' variable could potentially expose information about the application's vulnerability structure and pose a security risk if intercepted by an attacker.

Vulnerability: Hardcoded Secrets/Keys. Severity: Medium. Line: 53. Code: $vulnerabilityFile = 'impossible.php'; Explanation: Hardcoding the filename 'impossible.php' could reveal the internal logic of the application, which is not an ideal practice from a security standpoint.

Vulnerability: Hardcoded Secrets/Keys. Severity: Medium. Line: 13. Code: $page[ 'title' ] = 'Vulnerability: SQL Injection' . $page[ 'title\_separator' ].$page[ 'title' ]; Explanation: The hardcoded title 'Vulnerability: SQL Injection' in the page title may expose information about the security vulnerability and should be dynamically generated to avoid revealing such details.

Vulnerability: Hardcoded Secrets/Keys. Severity: Medium. Line: 15, 16, 17, 18. Code: $page[ 'page\_id' ] = 'sqli'; $page[ 'help\_button' ] = 'sqli'; $page[ 'source\_button' ] = 'sqli'; Explanation: The hardcoded values 'sqli' assigned to various page identifiers could potentially disclose information about the vulnerable page and should be obfuscated to enhance security.

Explanation: No hardcoded keys or secrets are found in the code.

####################################################################################################

####################################################################################################

File: sqli\session-input.php

Vulnerability: SQL Injection.

Severity: High.

Line: 11.

Code: $\_SESSION[ 'id' ] = $\_POST[ 'id' ];

Explanation: The code directly takes user input from the POST request and stores it in the $\_SESSION variable without sanitization or validation. This can lead to SQL Injection vulnerabilities if the input is manipulated by an attacker. Attackers can potentially manipulate the input to execute malicious SQL queries on the database.

Vulnerability: Cross-Site Scripting (XSS).

Severity: Medium.

Line: 13.

Code: $page[ 'body' ] .= "Session ID: {$\_SESSION[ 'id' ]}<br /><br /><br />";

Explanation: The code directly outputs the value of the 'id' session variable without proper encoding or validation, which could allow an attacker to inject and execute malicious scripts in the context of other users' sessions. This can be exploited to steal cookies, session tokens, or perform other malicious actions.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: sqli\test.php

Vulnerability: SQL Injection. Severity:High. Line: 5-7. Code: mssql\_connect($host, $username, $password);

Explanation: The code is using the deprecated and vulnerable mssql\_connect function which is prone to SQL injection attacks. The sensitive information such as host, username, and password is directly hardcoded into the code, making it easier for an attacker to extract this information and exploit the database. This code should be updated to use parameterized queries or prepared statements to prevent SQL injection attacks.

Vulnerability: Hardcoded credentials. Severity:High. Line: 2-3. Code: $username = "dvwa"; $password = "password";

Explanation: The database username and password are hardcoded in the source code, which is a significant security risk. Hardcoding sensitive information like passwords in the code makes it easier for attackers to extract the credentials and gain unauthorized access to the database. These credentials should be stored securely, such as in environment variables or a secure configuration file.

Other issue:

1. The variable $database is not defined before its use in mssql\_select\_db function. It will result in an error.

Overall, the code contains critical vulnerabilities that need to be addressed.

####################################################################################################

####################################################################################################

File: sqli\help\help.php

Vulnerability: SQL Injection.

Severity: High

Line: 17

Code: <pre>Spoiler: <span class="spoiler">?id=a' UNION SELECT "text1","text2";-- -&Submit=Submit</span>.</pre>

Explanation: The code is vulnerable to SQL Injection as it is directly inserting user input into the SQL query without proper sanitization or parameterization. An attacker can manipulate the input to perform malicious SQL queries like UNION SELECT to extract sensitive data or perform other harmful actions on the database.

Vulnerability: SQL Injection.

Severity: Medium

Line: 34

Code: <pre>Spoiler: <span class="spoiler">?id=a UNION SELECT 1,2;-- -&Submit=Submit</span>.</pre>

Explanation: The code is still susceptible to SQL Injection even though it uses mysql\_real\_escape\_string function, as the SQL query does not have quotes around the parameter, making it vulnerable to manipulation by attackers.

Vulnerability: SQL Injection.

Severity: High

Line: 44

Code: <pre>Spoiler: <span class="spoiler">ID: a' UNION SELECT "text1","text2";-- -&Submit=Submit</span>.</pre>

Explanation: The code is vulnerable to SQL Injection as the input values are being transferred to the vulnerable query via session variables, allowing attackers to manipulate the input and execute malicious queries.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: sqli\source\high.php

Vulnerability: SQL Injection. Severity: High. Line: 8. Code: $id = $\_SESSION[ 'id' ];

Explanation: The variable $id is directly taken from the session without any validation or sanitization, making it vulnerable to SQL injection attacks when used in the SQL query on line 11.

Vulnerability: SQL Injection. Severity: High. Line: 11. Code: $query = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id' LIMIT 1;";

Explanation: The SQL query is constructed using the $id variable without sanitization, which opens up the possibility of SQL injection attacks if the $id variable is manipulated to include malicious SQL code.

Vulnerability: Lack of Input Validation. Severity: Medium. Line: 5. Code: $id = $\_SESSION[ 'id' ];

Explanation: The code does not perform any input validation on the user input stored in the $\_SESSION['id'] variable before using it directly in a SQL query, which can lead to security vulnerabilities like SQL injection.

Vulnerability: Lack of Output Escaping. Severity: Medium. Line: 19. Code: $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

Explanation: The code does not escape the output values $id, $first, and $last when displaying them to the end user, which could lead to cross-site scripting (XSS) vulnerabilities if these values contain malicious scripts.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: sqli\source\impossible.php

Vulnerability: SQL Injection.

Severity: High.

Line: 12.

Code: $id = $\_GET[ 'id' ];

Explanation: The code directly uses user input ($\_GET['id']) in the SQL query without proper sanitization or validation, making it vulnerable to SQL injection attacks.

Vulnerability: Lack of Input Validation.

Severity: Medium.

Line: 9.

Code: checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );

Explanation: The input parameter 'user\_token' is being used without proper validation. Input validation should be implemented to avoid any potential security risks.

Vulnerability: Lack of Authorization.

Severity: Medium.

Line: 32.

Code: generateSessionToken();

Explanation: The code seems to generate an Anti-CSRF token without any proper authorization or validation process, which could lead to potential security vulnerabilities.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: sqli\source\low.php

Vulnerability: SQL Injection (SQLi).

Severity: High.

Line: 7-10.

Code: $id = $\_REQUEST[ 'id' ];

$query = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

$result = mysqli\_query($GLOBALS["\_\_\_mysqli\_ston"], $query ) or die( '<pre>' . ((is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_error($GLOBALS["\_\_\_mysqli\_ston"]) : (($\_\_\_mysqli\_res = mysqli\_connect\_error()) ? $\_\_\_mysqli\_res : false)) . '</pre>' );

Explanation: The code directly concatenates user input ($\_REQUEST['id']) into the SQL query without proper sanitization or parameterization, making it vulnerable to SQL injection attacks.

Vulnerability: Hardcoded Secrets.

Severity: Medium.

Line: 14.

Code: case MYSQL:

Explanation: The case for the MySQL database contains a reference to $\_DVWA['SQLI\_DB'] which appears to be a configuration setting. Depending on the value of this setting, the code interacts with different database types. The hardcoded reference to MySQL could potentially leak information about the application's configuration or setup.

Vulnerability: Commented Out Code.

Severity: Low.

Line: 21-22.

Code: #$sqlite\_db\_connection = new SQLite3($\_DVWA['SQLITE\_DB']);

#$sqlite\_db\_connection->enableExceptions(true);

Explanation: The commented out code suggests that SQLite3 connection and exception handling code may not be in use or properly configured, which could impact error handling and possibly security in the application.

Vulnerability: Unused Code.

Severity: Low.

Line: 23-24.

Code: $query = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";

#print $query;

Explanation: The code to print the query is commented out and unused. While not a critical vulnerability, it adds to code clutter and may increase confusion during maintenance.

No Vulnerabilities Found.

File: sqli\source\medium.php

Vulnerability: SQL Injection (SQLi).

Severity: High.

Line: 6.

Code: $id = $\_POST[ 'id' ];

Explanation: The code directly takes user input from the POST request without any input validation or sanitization, making it vulnerable to SQL Injection attacks. Although mysqli\_real\_escape\_string is used on line 8, it is not a reliable method to prevent SQL injection vulnerabilities. It's recommended to use parameterized queries or prepared statements to securely handle user input.

Vulnerability: Hardcoded Database Connection Information.

Severity: Medium.

Line: 25.

Code: $GLOBALS["\_\_\_mysqli\_ston"].

Explanation: The code contains a hardcoded database connection variable "$GLOBALS["\_\_\_mysqli\_ston"]", which may expose sensitive database credentials if the code is publicly accessible. It is recommended to store sensitive information like database credentials in secure configuration files outside of the web root directory.

Vulnerability: Lack of CSRF Protection.

Severity: Medium.

Line: 4.

Code: isset( $\_POST[ 'Submit' ] ).

Explanation: The code does not include any protection against Cross-Site Request Forgery (CSRF) attacks. Adding CSRF tokens to validate the origin of the requests can help mitigate this vulnerability.

Vulnerability: Error Handling Disclosure.

Severity: Low.

Line: 52.

Code: echo "Error in fetch ".$sqlite\_db->lastErrorMsg();

Explanation: The code directly outputs error messages to the end-users, which can potentially expose sensitive information about the application's backend. It's recommended to handle errors more gracefully without revealing such details to the users.

Vulnerability: Resource Exhaustion.

Severity: Medium.

Line: 22.

Code: global $sqlite\_db\_connection;

Explanation: The code does not limit the amount of data that can be retrieved from the database, which could lead to resource exhaustion issues if the query results in a large dataset. Implementing pagination or setting appropriate limits can help prevent this vulnerability.

Hardcoded Keys and Secrets: No hardcoded keys or secrets were found in the provided code.

Overall Assessment: The code contains multiple security vulnerabilities such as SQL Injection, hardcoded database connection information, lack of CSRF protection, error handling disclosure, and potential resource exhaustion. It's crucial to address these issues to enhance the security posture of the application.

# 8. CONCLUSION

In conclusion, the development of our vulnerability detector using AI represents a significant advancement in the realm of cybersecurity. Through the fusion of machine learning algorithms and sophisticated analysis techniques, we have created a powerful tool capable of identifying and mitigating potential security threats in software code. This project not only showcases the potential of AI in bolstering cybersecurity measures but also underscores our commitment to staying ahead of emerging cyber threats. Moving forward, we remain dedicated to refining and enhancing our detector to ensure its effectiveness in safeguarding software systems against evolving vulnerabilities. With our continued efforts and innovation, we aim to contribute to a safer and more secure digital landscape for users and organizations worldwide.

Top of Form

Bottom of Form

# 9. FURTHER ENHANCEMENTS

Looking ahead, several exciting avenues for enhancing our project present themselves. One key area for improvement lies in refining the user experience, with a focus on streamlining navigation and enhancing visual appeal. By incorporating modern design principles and intuitive user interfaces, we can ensure that interacting with our project is both seamless and enjoyable for users of all skill levels. Additionally, expanding the functionality of our project to include advanced features such as personalized recommendations, social networking integration, and real-time collaboration capabilities could significantly enrich the user experience. Furthermore, leveraging emerging technologies such as machine learning and natural language processing holds immense potential for enhancing the intelligence and adaptability of our project. By continually soliciting feedback from users and staying abreast of the latest developments in technology and design, we can ensure that our project remains innovative, relevant, and indispensable to our users in the future.

# 10. REFERENCES

 Smith, B. (Year). Book Title Cybersecurity and Attacks

 Stack Overflow

 Developer Documentation for Technology/Framework Used. (Year). [Link to Documentation]

 Academic Conference Proceedings.

 Personal Communication with Expert in the Field.

# 11. APPENDICES

**11.2 README**

Installation:

To install [Your Project Name], follow these simple steps:

Clone the repository to your local machine:

bash

git clone : Any Vulnerability DVWA files

Install project dependencies:

pip install -r requirements.txt

Top of Form

Bottom of Form

Top of Form

Bottom of Form

## 11.3 SAMPLE SOURCE PROGRAM

#### 11.3.2 MAIN.JS

#### import pathlib

#### import textwrap

#### import requests

#### API\_KEY="sk-proj-a6hADHopdGV0IMEPkXFvT3BlbkFJzaiDHXgqKXd51HHvdF5r"

#### from openai import OpenAI

#### client = OpenAI(

#### api\_key=API\_KEY,

#### )

#### import requests

#### import json,re

#### from time import sleep

#### vuln\_id = 1

#### vulnerability\_list=[]

#### unprocessedFiles=[]

#### processedFiles=[]

#### common\_code\_file\_extensions = (

#### '.py', # Python

#### '.js', # JavaScript

#### '.php', # PHP

#### '.c', # C

#### '.cpp', # C++

#### '.cs', # C#

#### '.java', # Java

#### '.rb', # Ruby

#### '.go', # Go

#### '.swift', # Swift

#### '.ts', # TypeScript

#### '.m', # Objective-C

#### '.rs', # Rust

#### '.lua', # Lua

#### '.pl', # Perl

#### '.sh', # Shell

#### '.r', # R

#### '.kt', # Kotlin

#### '.dart', # Dart

#### '.groovy', # Groovy

#### '.vb', # Visual Basic

#### '.vbs', # VBScript

#### '.f', '.f90', '.f95', # Fortran

#### '.asm', # Assembly

#### '.s', # Assembly

#### '.h', '.hpp', # C/C++ Header

#### '.hh', # C++ Header

#### '.vue', # Vue.js

#### '.jsx', # React JSX

#### '.tsx', # TypeScript with JSX

#### '.html', # HTML

#### '.css', # CSS

#### '.xml', # XML

#### '.yaml', # YAML

#### '.json', # JSON

#### '.conf', # Configuration

#### '.cfg', # Configuration

#### '.sh'

#### )

#### def read\_files\_in\_directory(directory,output):

#### nooffiles=0

#### nooflines=0

#### for root, dirs, files in os.walk(directory):

#### for file in files:

#### if file.lower().endswith(common\_code\_file\_extensions):

#### file\_path = os.path.join(root, file)

#### try:

#### with open(file\_path, 'r') as f:

#### file\_contents = f.read()

#### print("#"\*100)

#### output.write("#"\*100+"\n")

#### print(f"File: {file\_path}")

#### output.write(f"File: {file\_path}\n")

#### analyze\_file(file\_path,file\_contents,output)

#### print("#"\*100)

#### output.write("#"\*100+"\n")

#### processedFiles.append(file\_path)

#### nooffiles=nooffiles+1

#### nooflines=nooflines+len(file\_contents)

#### except Exception as e:

#### print(e)

#### unprocessedFiles.append(file\_path)

#### print("Error in Parsing File")

#### print(f"Total Number of Files Processes is '{nooffiles}' and total Number of lines processed is : '{nooflines}'")

#### for file\_path in unprocessedFiles:

#### try:

#### f = open(file\_path, 'r')

#### file\_contents = f.read()

#### analyze\_file(file\_path,file\_contents)

#### except:

#### continue

#### def scanthedir(directory):

#### global vulnerability\_list

#### print(directory)

#### path=os.getcwd()

#### # Define the directory path and the filename

#### directory\_path = os.getcwd()

#### filename = "output"+directory+".txt"

#### filepath = os.path.join(directory\_path, filename)

#### output=open(filepath,"w")

#### read\_files\_in\_directory(directory,output)

#### output.write("\n")

#### output.write("#"\*100+"\n")

#### output.write(str(unprocessedFiles))

#### output.write("#"\*100+"\n")

#### vuln={}

#### output.write(str(unprocessedFiles))

#### filename2="vulnerablities\_"+directory+".json"

#### filepath2 = os.path.join(directory\_path, filename2)

#### f = open(filepath2, 'w')

#### vuln["vulnerabilities"] = vulnerability\_list

#### json.dump(vuln, f, indent=4)

#### f.close()

#### end\_time = time.time()

#### execution\_time = end\_time - start\_time

#### print(f"Execution time: {execution\_time} seconds")

#### def extract\_vulnerability(text):

#### # Refined Regular Expression Pattern

#### pattern = r'Vulnerability: (.\*?)\.?Severity: (.\*?)Line: (.\*?)Code:\s\*((?:.|\n)\*?)Explanation: (.\*)'

#### match = re.search(pattern, text, re.DOTALL)

#### if match:

#### return {

#### "Vulnerability": match.group(1).strip(),

#### "Severity": match.group(2).strip(),

#### "Line": match.group(3).strip(),

#### "Code": match.group(4).strip(),

#### "Explanation": match.group(5).strip()

#### }

#### else:

#### return None

#### def create\_vulndadta(vulnerabilities,file):

#### global vuln\_id

#### global vulnerability\_list

#### print("Creating VUln Index")

#### vulnerability\_sections = [section for section in vulnerabilities.split("\n\n") if section.strip()]

#### for section in vulnerability\_sections:

#### vulnerability\_data = extract\_vulnerability(section)

#### if vulnerability\_data:

#### vulnerability\_data['file']=file

#### vulnerability\_list.append(vulnerability\_data)

#### # Output JSON

#### def analyze\_file(file\_path,file,output):

#### print(len(vulnerability\_list))

#### file\_content = file

#### try:

#### system\_prompt = '''Pretend You are a skilled application security engineer doing a static code analysis on a code repository. Analyze the following source code for potential security vulnerabilities. Consider common threat categories like injection vulnerabilities (SQLi, XSS), buffer overflows, insecure configuration, weak authentication/authorization, insufficient input validation, and use of outdated libraries.

#### You will be sent code, which you should assess for potential vulnerabilities. List any security vulnerability in below source code. Ensure to include vulnerable code, line numbers and other details in a human readable secure source code reporting format. Dont report those issues where you are not certain, dont report version related issues, focus only on code, I/O and any function calls and dont give any generic recommendations if code is clean. Output vulnerabilities found in this format: "Vulnerability: [Vulnerability Name]. Severity:[Severity of the Vulnerablity]. Line: [Line Number]. Code: [Code snippet of the vulnerable line(s) of code without any new line replace new line with tab] Explanation: [Explanation of the vulnerability]\n"

#### Also check if there are any hardcoded keys and secrets in the code.If no vulnerablities found mention No VUlnerablities Found. Stay in character for each response.

#### '''

#### user\_prompt = "The path of the file is " + file\_path+".\n\nThe code for the file is as follows:\n\n"+ file\_content

#### completion = client.chat.completions.create(

#### model="gpt-3.5-turbo",

#### messages=[

#### {"role": "system", "content": system\_prompt},

#### {"role": "user", "content": user\_prompt}

#### ]

#### )

#### extracted\_text = completion.choices[0].message.content

#### data=extracted\_text

#### print(data)

#### create\_vulndadta(data,file\_path)

#### output.write(str(data))

#### output.write("\n")

#### except Exception as e:

#### print(e)

#### unprocessedFiles.append(file\_path)

#### print("No Vulnerabilities Found. ")

#### return

#### import os

#### import time

#### start\_time = time.time()

#### print(vulnerability\_list)

#### dirToScan = input("Please enter The folder in the current Directory to scan: ")

#### for root, dirs, files in os.walk(dirToScan):

#### for file in files:

#### if file.lower().endswith(common\_code\_file\_extensions):

#### file\_path = os.path.join(root, file)

#### print(file\_path)

#### scanthedir(dirToScan)

#### 11.3.3 OUTPUT\_REPORT.JSON

{

"vulnerabilities": [

{

"Vulnerability": "SQL Injection.",

"Severity": "High",

"Line": "41",

"Code": "if( $vulnerabilityFile == 'impossible.php' )",

"Explanation": "The code is vulnerable to SQL injection on line 41 where a tokenField() function is called conditionally based on the value of $vulnerabilityFile. This could lead to SQL injection if the tokenField() function does not properly sanitize or validate user input.",

"file": "sqli\\index.php"

},

{

"Vulnerability": "Hardcoded Secrets.",

"Severity": "High",

"Line": "3",

"Code": "require\_once DVWA\_WEB\_PAGE\_TO\_ROOT . 'dvwa/includes/dvwaPage.inc.php';",

"Explanation": "The definition of 'DVWA\_WEB\_PAGE\_TO\_ROOT' is directly using hardcoded directory traversal paths. Hardcoding paths can lead to security risks, especially if sensitive files are accessed in an insecure manner.",

"file": "sqli\\index.php"

},

{

"Vulnerability": "SQL Injection.",

"Severity": "High.",

"Line": "9.",

"Code": "$\_SESSION[ 'id' ] = $\_POST[ 'id' ];",

"Explanation": "The code directly assigns the value of the POST parameter 'id' to the session 'id' without any validation or sanitization, making it vulnerable to SQL Injection attacks.",

"file": "sqli\\session-input.php"

},

{

"Vulnerability": "Cross-Site Scripting (XSS).",

"Severity": "Medium.",

"Line": "11.",

"Code": "$page[ 'body' ] .= \"Session ID: {$\_SESSION[ 'id' ]}<br /><br /><br />\";",

"Explanation": "The value retrieved from the session 'id' is directly echoed to the page without proper sanitization or encoding, allowing an attacker to inject malicious scripts.",

"file": "sqli\\session-input.php"

},

{

"Vulnerability": "SQL Injection.",

"Severity": "High.",

"Line": "7.",

"Code": "mssql\_query($query);",

"Explanation": "The code is vulnerable to SQL Injection as it directly passes the $query variable without proper sanitization into the mssql\_query function. An attacker can manipulate the $query variable to execute malicious SQL statements, leading to data leakage, data manipulation, or even unauthorized access to the database.",

"file": "sqli\\test.php"

},

{

"Vulnerability": "Hardcoded Credentials.",

"Severity": "Medium.",

"Line": "2-4.",

"Code": "$host = \"192.168.0.7\"; $username = \"dvwa\"; $password = \"password\";",

"Explanation": "The code contains hardcoded credentials for the database connection, which poses a security risk. Hardcoded credentials can be easily compromised and misused by attackers if they gain access to the source code or binaries.",

"file": "sqli\\test.php"

},

{

"Vulnerability": "SQL Injection.",

"Severity": "High.",

"Line": "21.",

"Code": "<span class=\"spoiler\">?id=a' UNION SELECT \"text1\",\"text2\";-- -&Submit=Submit</span>.",

"Explanation": "The code snippet allows for a SQL Injection attack by including a malicious SQL query in the input parameter \"id\". This can lead to sensitive data exposure, data manipulation, and potentially unauthorized access to the database.",

"file": "sqli\\help\\help.php"

},

{

"Vulnerability": "Hardcoded secret.",

"Severity": "High.",

"Line": "50.",

"Code": "<?php echo dvwaExternalLinkUrlGet( 'https://www.owasp.org/index.php/SQL\_Injection' ); ?>.",

"Explanation": "The code contains a hardcoded secret in the form of an external link URL, which can be a security risk if this secret is sensitive information that should not be exposed.",

"file": "sqli\\help\\help.php"

},

{

"Vulnerability": "SQL Injection.",

"Severity": "High.",

"Line": "8.",

"Code": "$id = $\_SESSION[ 'id' ];",

"Explanation": "The code directly uses the $id variable from the session data to construct a SQL query without sanitizing or validating the input. This makes the code vulnerable to SQL injection attacks if the session data is manipulated by an attacker.",

"file": "sqli\\source\\high.php"

},

{

"Vulnerability": "Lack of Input Sanitization.",

"Severity": "Medium.",

"Line": "8.",

"Code": "$id = $\_SESSION[ 'id' ];",

"Explanation": "The code does not perform any input sanitization or validation on the $id variable obtained from the session data, leaving the application susceptible to various forms of attacks, including SQL injection.",

"file": "sqli\\source\\high.php"

},

{

"Vulnerability": "Lack of Error Handling.",

"Severity": "Low.",

"Line": "27.",

"Code": "echo 'Caught exception: ' . $e->getMessage();",

"Explanation": "The code catches exceptions but does not handle them in a secure manner. Revealing error messages to end users can potentially expose sensitive information about the application configuration.",

"file": "sqli\\source\\high.php"

},

{

"Vulnerability": "SQL Injection (SQLi).",

"Severity": "High.",

"Line": "11.",

"Code": "$id = $\_GET[ 'id' ];",

"Explanation": "The code directly takes the 'id' parameter from the GET request without proper sanitization or validation. This can lead to SQL Injection vulnerabilities where an attacker can manipulate the input to execute arbitrary SQL queries on the database.",

"file": "sqli\\source\\impossible.php"

},

{

"Vulnerability": "Missing Anti-CSRF Protection.",

"Severity": "Medium.",

"Line": "6.",

"Code": "checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );",

"Explanation": "There is a call to a function to check an Anti-CSRF token, but the actual implementation of the function is not provided. Without a proper Anti-CSRF protection mechanism, the application is susceptible to CSRF attacks.",

"file": "sqli\\source\\impossible.php"

},

{

"Vulnerability": "SQL Injection.",

"Severity": "High.",

"Line": "6.",

"Code": "$id = $\_REQUEST[ 'id' ];",

"Explanation": "The code directly uses the user input from the 'id' parameter without properly sanitizing it or using prepared statements, making it vulnerable to SQL Injection attacks. An attacker can manipulate the input to execute arbitrary SQL queries.",

"file": "sqli\\source\\low.php"

},

{

"Vulnerability": "Hardcoded Secrets.",

"Severity": "Medium.",

"Line": "15.",

"Code": "$query = \"SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';\";",

"Explanation": "The database query in line 15 contains the user-provided 'id' parameter directly concatenated into the SQL query, which is a weak practice that can potentially expose database information.",

"file": "sqli\\source\\low.php"

},

{

"Vulnerability": "Hardcoded Secrets.",

"Severity": "Medium.",

"Line": "18.",

"Code": "$result = mysqli\_query($GLOBALS[\"\_\_\_mysqli\_ston\"], $query ) or die( '<pre>' . ((is\_object($GLOBALS[\"\_\_\_mysqli\_ston\"])) ? mysqli\_error($GLOBALS[\"\_\_\_mysqli\_ston\"]) : (($\_\_\_mysqli\_res = mysqli\_connect\_error()) ? $\_\_\_mysqli\_res : false)) . '</pre>' );",

"Explanation": "The database query in line 18 concatenates user input directly into the SQL query without proper validation, which can lead to SQL injection attacks.",

"file": "sqli\\source\\low.php"

},

{

"Vulnerability": "Hardcoded Secrets.",

"Severity": "Medium.",

"Line": "27.",

"Code": "$results = $sqlite\_db\_connection->query($query);",

"Explanation": "The code directly uses the user input in the SQLite query without proper validation, which may lead to SQL injection vulnerabilities.",

"file": "sqli\\source\\low.php"

},

{

"Vulnerability": "Hardcoded Secrets.",

"Severity": "Medium.",

"Line": "42.",

"Code": "echo \"Error in fetch \".$sqlite\_db->lastErrorMsg();",

"Explanation": "The code exposes details about the database error messages to end-users, which can potentially leak sensitive information about the application's internal workings.",

"file": "sqli\\source\\low.php"

},

{

"Vulnerability": "SQL Injection.",

"Severity": "High.",

"Line": "7.",

"Code": "$id = $\_POST[ 'id' ];",

"Explanation": "The $id variable is directly taken from user input without proper sanitization. Although mysqli\_real\_escape\_string is used afterwards, it is not considered a foolproof method for preventing SQL injection. Prepared statements should be used for parameterized queries to prevent SQL injection attacks.",

"file": "sqli\\source\\medium.php"

},

{

"Vulnerability": "Hardcoded Database Credentials.",

"Severity": "High.",

"Line": "6, 13.",

"Code": "$GLOBALS[\"\_\_\_mysqli\_ston\"].",

"Explanation": "The database connection variable $GLOBALS[\"\_\_\_mysqli\_ston\"] is used directly within the code to establish a database connection. Hardcoding database credentials in the code is a security risk as it exposes sensitive information. It is recommended to store credentials securely and use environment variables or configuration files to retrieve them.",

"file": "sqli\\source\\medium.php"

},

{

"Vulnerability": "Lack of Error Handling.",

"Severity": "Medium.",

"Line": "24, 34.",

"Code": "die(), echo.",

"Explanation": "The code uses die() and echo statements for error handling, which can reveal sensitive information to attackers. It is better to handle errors gracefully without exposing details that can aid attackers in exploiting the application.",

"file": "sqli\\source\\medium.php"

},

{

"Vulnerability": "Insecure Concatenation.",

"Severity": "Medium.",

"Line": "11, 19.",

"Code": "$query = \"SELECT first\_name, last\_name FROM users WHERE user\_id = $id;\".",

"Explanation": "The SQL query is dynamically constructed by concatenating user input ($id) directly into the query string. This can potentially lead to SQL injection vulnerabilities if the input is not properly sanitized or validated.",

"file": "sqli\\source\\medium.php"

},

{

"Vulnerability": "Lack of Input Validation.",

"Severity": "Low.",

"Line": "3, 7.",

"Code": "isset(), $\_POST['id'].",

"Explanation": "The code only checks if the 'Submit' button is set and retrieves the 'id' parameter from POST data without further input validation. Input validation should be implemented to ensure that the input meets the expected criteria before processing it.",

"file": "sqli\\source\\medium.php"

},

{

"Vulnerability": "Unnecessary Commented Code.",

"Severity": "Low.",

"Line": "20.",

"Code": "#print $query;.",

"Explanation": "Commented-out code should be removed from production code as it can potentially expose sensitive information or provide insights to attackers.",

"file": "sqli\\source\\medium.php"

},

{

"Vulnerability": "Database Connection Left Open.",

"Severity": "Low.",

"Line": "40.",

"Code": "mysqli\_close($GLOBALS[\"\_\_\_mysqli\_ston\"]);.",

"Explanation": "The database connection is closed at the end of the script, but it is recommended to close the connection immediately after its use to minimize the window of opportunity for potential attacks like resource exhaustion.",

"file": "sqli\\source\\medium.php"

}

]

}

**Examples of output:**

{

    "vulnerabilities": [

        {

            "Vulnerability": "Cross Site Request Forgery (CSRF) Protection Bypass.",

            "Severity": "High.",

            "Line": "45.",

            "Code": "if( $vulnerabilityFile == 'high.php' || $vulnerabilityFile == 'impossible.php' )",

            "Explanation": "The code does not properly validate the incoming request when checking if the CSRF protection token should be included. In this case, if the vulnerability file is set to 'high.php' or 'impossible.php', the tokenField() function is not called to generate the CSRF token. This can lead to a potential CSRF protection bypass vulnerability.",

            "file": "csrf\\index.php"

        },

        {

            "Vulnerability": "Hardcoded Secret in JavaScript Code.",

            "Severity": "Medium.",

            "Line": "18.",

            "Code": "window.open(\\\"\" . DVWA\_WEB\_PAGE\_TO\_ROOT . \"vulnerabilities/csrf/test\_credentials.php\\\"",

            "Explanation": "The JavaScript code contains a hardcoded URL to 'test\_credentials.php' which can be considered a security risk as sensitive operations or endpoints should not be exposed directly in the frontend code.",

            "file": "csrf\\index.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "18.",

            "Code": "$query  = \"SELECT \* FROM `users` WHERE user='$user' AND password='$pass';\";.",

            "Explanation": "The code directly concatenates user input ($user and $pass) into the SQL query without proper sanitization, making it vulnerable to SQL Injection attacks. An attacker could manipulate the input to execute arbitrary SQL queries.",

            "file": "csrf\\test\_credentials.php"

        },

        {

            "Vulnerability": "Use of MD5 for password hashing.",

            "Severity": "Medium.",

            "Line": "13.",

            "Code": "$pass = md5( $pass );.",

            "Explanation": "Using MD5 for password hashing is considered weak and insecure. MD5 is no longer recommended for password hashing due to its vulnerability to brute force attacks and its potential for collisions.",

            "file": "csrf\\test\_credentials.php"

        },

        {

            "Vulnerability": "Hardcoded keys/secrets.",

            "Severity": "Low.",

            "Line": "3.",

            "Code": "define( 'DVWA\_WEB\_PAGE\_TO\_ROOT', '../../' );.",

            "Explanation": "The code contains a hardcoded path in the definition of DVWA\_WEB\_PAGE\_TO\_ROOT. Hardcoding sensitive information like paths could expose information to attackers if the code is accessed or compromised.",

            "file": "csrf\\test\_credentials.php"

        },

        {

            "Vulnerability": "Cross-Site Scripting (XSS).",

            "Severity": "Medium.",

            "Line": "4-93.",

            "Code": "<table width='100%' bgcolor='white' style=\"border:2px #C0C0C0 solid\">...<p>Reference: <?php echo dvwaExternalLinkUrlGet( 'https://owasp.org/www-community/attacks/csrf' ); ?></p>",

            "Explanation": "The code within the table element is vulnerable to XSS as it is directly outputting user-controlled data without proper sanitization. This can be exploited by an attacker to execute malicious scripts in the context of the user's browser.",

            "file": "csrf\\help\\help.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "31.",

            "Code": "$insert = \"UPDATE `users` SET password = '\" . $pass\_new . \"' WHERE user = '\" . $current\_user . \"';\".",

            "Explanation": "The values of $pass\_new and $current\_user are directly concatenated into the SQL query without proper sanitization, making the code vulnerable to SQL injection attacks.",

            "file": "csrf\\source\\high.php"

        },

        {

            "Vulnerability": "Insufficient input validation.",

            "Severity": "Medium.",

            "Line": "13-17.",

            "Code": "if (array\_key\_exists(\"HTTP\_USER\_TOKEN\", $\_SERVER) && array\_key\_exists(\"password\_new\", $data) && array\_key\_exists(\"password\_conf\", $data) && array\_key\_exists(\"Change\", $data)) {.",

            "Explanation": "The code does not perform sufficient input validation on the incoming data before processing it, which can lead to unexpected behavior and possible security issues.",

            "file": "csrf\\source\\high.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "12, 21, 32, 37.",

            "Code": "$pass\_curr = stripslashes( $pass\_curr );\n\t$pass\_curr = ((isset($GLOBALS[\"\_\_\_mysqli\_ston\"]) && is\_object($GLOBALS[\"\_\_\_mysqli\_ston\"])) ? mysqli\_real\_escape\_string($GLOBALS[\"\_\_\_mysqli\_ston\"],  $pass\_curr ) : ((trigger\_error(\"[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.\", E\_USER\_ERROR)) ? \"\" : \"\"));",

            "Explanation": "The code is attempting to prevent SQL injection by using `mysqli\_real\_escape\_string`, however, it is not an effective way to prevent SQL injection. Furthermore, the `mysqli\_real\_escape\_string` function should not be used in conjunction with prepared statements as it may lead to insecurity.",

            "file": "csrf\\source\\impossible.php"

        },

        {

            "Vulnerability": "Lack of parameterized queries.",

            "Severity": "High.",

            "Line": "32, 39.",

            "Code": "$data = $db->prepare( 'SELECT password FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );\n\t$data = $db->prepare( 'UPDATE users SET password = (:password) WHERE user = (:user);' );",

            "Explanation": "The queries are using values directly in the query string instead of using parameterized queries. This makes the code vulnerable to SQL injection attacks.",

            "file": "csrf\\source\\impossible.php"

        },

        {

            "Vulnerability": "Lack of input validation.",

            "Severity": "Medium.",

            "Line": "5, 7, 9.",

            "Code": "checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );\n\t$pass\_curr = $\_GET[ 'password\_current' ];\n\t$pass\_new  = $\_GET[ 'password\_new' ];\n\t$pass\_conf = $\_GET[ 'password\_conf' ];",

            "Explanation": "The code is directly using input from the `$\_GET` array without proper validation. This can lead to various injection attacks like XSS or CSRF.",

            "file": "csrf\\source\\impossible.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "12.",

            "Code": "$insert = \"UPDATE `users` SET password = '$pass\_new' WHERE user = '\" . $current\_user . \"';\";",

            "Explanation": "The user input $current\_user is concatenated directly into the SQL query without proper sanitization, making the code vulnerable to SQL injection attacks.",

            "file": "csrf\\source\\low.php"

        },

        {

            "Vulnerability": "Use of md5 for password hashing.",

            "Severity": "High.",

            "Line": "10.",

            "Code": "$pass\_new = md5( $pass\_new );",

            "Explanation": "Using the md5 hash function for password hashing is considered insecure as it is fast and easily crackable. It is recommended to use stronger and slower hashing algorithms like bcrypt or Argon2 for secure password hashing.",

            "file": "csrf\\source\\low.php"

        },

        {

            "Vulnerability": "SQL Injection.",

            "Severity": "High.",

            "Line": "16.",

            "Code": "$insert = \"UPDATE `users` SET password = '$pass\_new' WHERE user = '\" . $current\_user . \"';\";",

            "Explanation": "The code directly concatenates user input ($current\_user) into the SQL query without proper sanitization, making it vulnerable to SQL injection attacks. An attacker could manipulate the $current\_user variable to perform SQL injection and potentially manipulate the database.",

            "file": "csrf\\source\\medium.php"

        },

        {

            "Vulnerability": "Insufficient Input Validation.",

            "Severity": "Medium.",

            "Line": "4.",

            "Code": "if( stripos( $\_SERVER[ 'HTTP\_REFERER' ] ,$\_SERVER[ 'SERVER\_NAME' ]) !== false ) {",

            "Explanation": "The code uses HTTP Referer header to check the source of the request, but HTTP Referer can be easily manipulated or blocked. It's not a reliable method for validating the source of a request. A more secure approach would be to implement CSRF tokens to prevent cross-site request forgery attacks.",

            "file": "csrf\\source\\medium.php"

        },

        {

            "Vulnerability": "Use of Deprecated Function.",

            "Severity": "Low.",

            "Line": "10.",

            "Code": "$pass\_new = ((isset($GLOBALS[\"\_\_\_mysqli\_ston\"]) && is\_object($GLOBALS[\"\_\_\_mysqli\_ston\"])) ? mysqli\_real\_escape\_string($GLOBALS[\"\_\_\_mysqli\_ston\"],  $pass\_new ) : ((trigger\_error(\"[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.\", E\_USER\_ERROR)) ? \"\" : \"\"));",

            "Explanation": "The code uses the deprecated function mysqli\_real\_escape\_string, which is not recommended for preventing SQL injection. It's better to use prepared statements to prevent SQL injection securely.",

            "file": "csrf\\source\\medium.php"

        }

    ]

}

**Detailed Explanation of Output:**

####################################################################################################

File: csrf\index.php

Vulnerability: Cross Site Request Forgery (CSRF) Protection Bypass.

Severity: High.

Line: 45.

Code: if( $vulnerabilityFile == 'high.php' || $vulnerabilityFile == 'impossible.php' )

Explanation: The code does not properly validate the incoming request when checking if the CSRF protection token should be included. In this case, if the vulnerability file is set to 'high.php' or 'impossible.php', the tokenField() function is not called to generate the CSRF token. This can lead to a potential CSRF protection bypass vulnerability.

Vulnerability: Hardcoded Secret in JavaScript Code.

Severity: Medium.

Line: 18.

Code: window.open(\"" . DVWA\_WEB\_PAGE\_TO\_ROOT . "vulnerabilities/csrf/test\_credentials.php\"

Explanation: The JavaScript code contains a hardcoded URL to 'test\_credentials.php' which can be considered a security risk as sensitive operations or endpoints should not be exposed directly in the frontend code.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: csrf\test\_credentials.php

Vulnerability: SQL Injection. Severity: High. Line: 18. Code: $query = "SELECT \* FROM `users` WHERE user='$user' AND password='$pass';";. Explanation: The code directly concatenates user input ($user and $pass) into the SQL query without proper sanitization, making it vulnerable to SQL Injection attacks. An attacker could manipulate the input to execute arbitrary SQL queries.

Vulnerability: Use of MD5 for password hashing. Severity: Medium. Line: 13. Code: $pass = md5( $pass );. Explanation: Using MD5 for password hashing is considered weak and insecure. MD5 is no longer recommended for password hashing due to its vulnerability to brute force attacks and its potential for collisions.

Vulnerability: Hardcoded keys/secrets. Severity: Low. Line: 3. Code: define( 'DVWA\_WEB\_PAGE\_TO\_ROOT', '../../' );. Explanation: The code contains a hardcoded path in the definition of DVWA\_WEB\_PAGE\_TO\_ROOT. Hardcoding sensitive information like paths could expose information to attackers if the code is accessed or compromised.

These vulnerabilities could lead to security breaches and potential unauthorized access to sensitive data. It is recommended to address these issues to enhance the security of the application.

####################################################################################################

####################################################################################################

File: csrf\help\help.php

Vulnerability: Cross-Site Scripting (XSS). Severity: Medium. Line: 4-93.

Code: <table width='100%' bgcolor='white' style="border:2px #C0C0C0 solid">...<p>Reference: <?php echo dvwaExternalLinkUrlGet( 'https://owasp.org/www-community/attacks/csrf' ); ?></p>

Explanation: The code within the table element is vulnerable to XSS as it is directly outputting user-controlled data without proper sanitization. This can be exploited by an attacker to execute malicious scripts in the context of the user's browser.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: csrf\source\high.php

Vulnerability: SQL Injection. Severity: High. Line: 31. Code: $insert = "UPDATE `users` SET password = '" . $pass\_new . "' WHERE user = '" . $current\_user . "';". Explanation: The values of $pass\_new and $current\_user are directly concatenated into the SQL query without proper sanitization, making the code vulnerable to SQL injection attacks.

Vulnerability: Insufficient input validation. Severity: Medium. Line: 13-17. Code: if (array\_key\_exists("HTTP\_USER\_TOKEN", $\_SERVER) && array\_key\_exists("password\_new", $data) && array\_key\_exists("password\_conf", $data) && array\_key\_exists("Change", $data)) {. Explanation: The code does not perform sufficient input validation on the incoming data before processing it, which can lead to unexpected behavior and possible security issues.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: csrf\source\impossible.php

Vulnerability: SQL Injection. Severity: High. Line: 12, 21, 32, 37.

Code: $pass\_curr = stripslashes( $pass\_curr );

$pass\_curr = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $pass\_curr ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

Explanation: The code is attempting to prevent SQL injection by using `mysqli\_real\_escape\_string`, however, it is not an effective way to prevent SQL injection. Furthermore, the `mysqli\_real\_escape\_string` function should not be used in conjunction with prepared statements as it may lead to insecurity.

Vulnerability: Lack of parameterized queries. Severity: High. Line: 32, 39.

Code: $data = $db->prepare( 'SELECT password FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );

$data = $db->prepare( 'UPDATE users SET password = (:password) WHERE user = (:user);' );

Explanation: The queries are using values directly in the query string instead of using parameterized queries. This makes the code vulnerable to SQL injection attacks.

Vulnerability: Lack of input validation. Severity: Medium. Line: 5, 7, 9.

Code: checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );

$pass\_curr = $\_GET[ 'password\_current' ];

$pass\_new = $\_GET[ 'password\_new' ];

$pass\_conf = $\_GET[ 'password\_conf' ];

Explanation: The code is directly using input from the `$\_GET` array without proper validation. This can lead to various injection attacks like XSS or CSRF.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: csrf\source\low.php

Vulnerability: SQL Injection. Severity: High. Line: 12. Code: $insert = "UPDATE `users` SET password = '$pass\_new' WHERE user = '" . $current\_user . "';";

Explanation: The user input $current\_user is concatenated directly into the SQL query without proper sanitization, making the code vulnerable to SQL injection attacks.

Vulnerability: Use of md5 for password hashing. Severity: High. Line: 10. Code: $pass\_new = md5( $pass\_new );

Explanation: Using the md5 hash function for password hashing is considered insecure as it is fast and easily crackable. It is recommended to use stronger and slower hashing algorithms like bcrypt or Argon2 for secure password hashing.

No Vulnerabilities Found.

####################################################################################################

####################################################################################################

File: csrf\source\medium.php

Vulnerability: SQL Injection.

Severity: High.

Line: 16.

Code: $insert = "UPDATE `users` SET password = '$pass\_new' WHERE user = '" . $current\_user . "';";

Explanation: The code directly concatenates user input ($current\_user) into the SQL query without proper sanitization, making it vulnerable to SQL injection attacks. An attacker could manipulate the $current\_user variable to perform SQL injection and potentially manipulate the database.

Vulnerability: Insufficient Input Validation.

Severity: Medium.

Line: 4.

Code: if( stripos( $\_SERVER[ 'HTTP\_REFERER' ] ,$\_SERVER[ 'SERVER\_NAME' ]) !== false ) {

Explanation: The code uses HTTP Referer header to check the source of the request, but HTTP Referer can be easily manipulated or blocked. It's not a reliable method for validating the source of a request. A more secure approach would be to implement CSRF tokens to prevent cross-site request forgery attacks.

Vulnerability: Use of Deprecated Function.

Severity: Low.

Line: 10.

Code: $pass\_new = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $pass\_new ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

Explanation: The code uses the deprecated function mysqli\_real\_escape\_string, which is not recommended for preventing SQL injection. It's better to use prepared statements to prevent SQL injection securely.

Vulnerability: Hardcoded Secrets/Keys.

Severity: High.

Code: <No specific line, but inferred from the code present>.

Explanation: The code uses hardcoded database connection credentials like $GLOBALS["\_\_\_mysqli\_ston"], which is a security risk. Storing sensitive information like database credentials directly in the code increases the likelihood of exposure and unauthorized access to the database.

Overall, the code has multiple security vulnerabilities that need to be addressed for improved security.