UNIVERSITÄT PASSAU

Security Insider Lab II

TITLE

AUTHOR

1. xxxx 2. xxxxx

---

29. Juni 2024

Contents

Abstract

The Vulnerability Scanner project aims to enhance software security by developing a tool that identifies and reports security vulnerabilities within software projects. The tool provides functionalities for user authentication, project creation, and management, as well as the capability to scan code repositories or Python source files for various vulnerabilities. By leveraging advanced machine learning models, including LSTM, CNN, and MLP, the system detects vulnerabilities and generates comprehensive reports. Users can rescan their code after implementing fixes to ensure improved security. The system architecture includes a user-friendly interface, a robust backend for service management, a secure database for storing credentials and project data, and pre-trained machine learning models for effective vulnerability detection. The workflow involves user login, project setup, code retrieval and conversion, vulnerability analysis, report generation, and rescanning. This approach ensures a continuous cycle of security assessment and improvement, addressing vulnerabilities such as XSS, Path Disclosure, Remote Code Execution, and Command Injection.

Acknowledgments

List of Figures

List of Tables

1 Introduction

The SafeScript project addresses the critical need for enhanced security in software development by developing an advanced tool that identifies security vulnerabilities within software projects. This project aims to provide users with comprehensive capabilities to log in, create, and manage projects, and scan code repositories or Python source files for various vulnerabilities. The system employs sophisticated machine learning models to detect potential security issues and generates detailed reports. Additionally, users can rescan their code after making necessary fixes to ensure that security improvements are effectively implemented.

The project encompasses several functional requirements. First, it includes a user authentication feature where users must log in with a username and password to access the system, ensuring secure access. Second, it allows for project creation, enabling users to create new projects by providing a name and selecting a code repository or uploading a Python source code file. Third, repository management lets users specify the source of the code, either via a repository link or direct file upload. Fourth, the code retrieval function automates the process of fetching the code from the specified repository or processing the uploaded file. Fifth, data conversion converts the retrieved code into word2vec (w2v) format for subsequent vulnerability analysis. Sixth, vulnerability detection utilizes pre-trained machine learning models (LSTM, ChatGPT API) to identify vulnerabilities in the code. Seventh, report generation produces a comprehensive report detailing detected vulnerabilities, their severity, and recommendations for remediation. Eighth, the rescan capability allows users to rescan their code post-fix to confirm that vulnerabilities have been addressed. Lastly, the system supports the detection of specific types of vulnerabilities, including XSS (Cross-Site Scripting), Path Disclosure, Remote Code Execution, and Command Injection.

The technical architecture of the SafeScript system comprises several key components. The user interface (UI) includes a login page, a dashboard for project management, and options for report viewing and rescanning. The backend includes services for authentication, project management, code retrieval, vulnerability detection, and report generation. The database stores user credentials, project details, and scanning results and reports. The system also incorporates pre-trained machine learning models (LSTM and ChatGPT API) for vulnerability detection.

The workflow of the SafeScript system involves several steps. Initially, the user logs in using credentials. The user then creates a new project and selects the source of the code. The system fetches the code from the repository or processes the uploaded file. The code is then converted to w2v format, and the selected machine learning model analyzes the code for vulnerabilities. Subsequently, the system generates and displays a report. Finally, the user can rescan the code after making fixes to ensure that vulnerabilities have been addressed. Figure 1.1 shows the complete workflow of the applications.

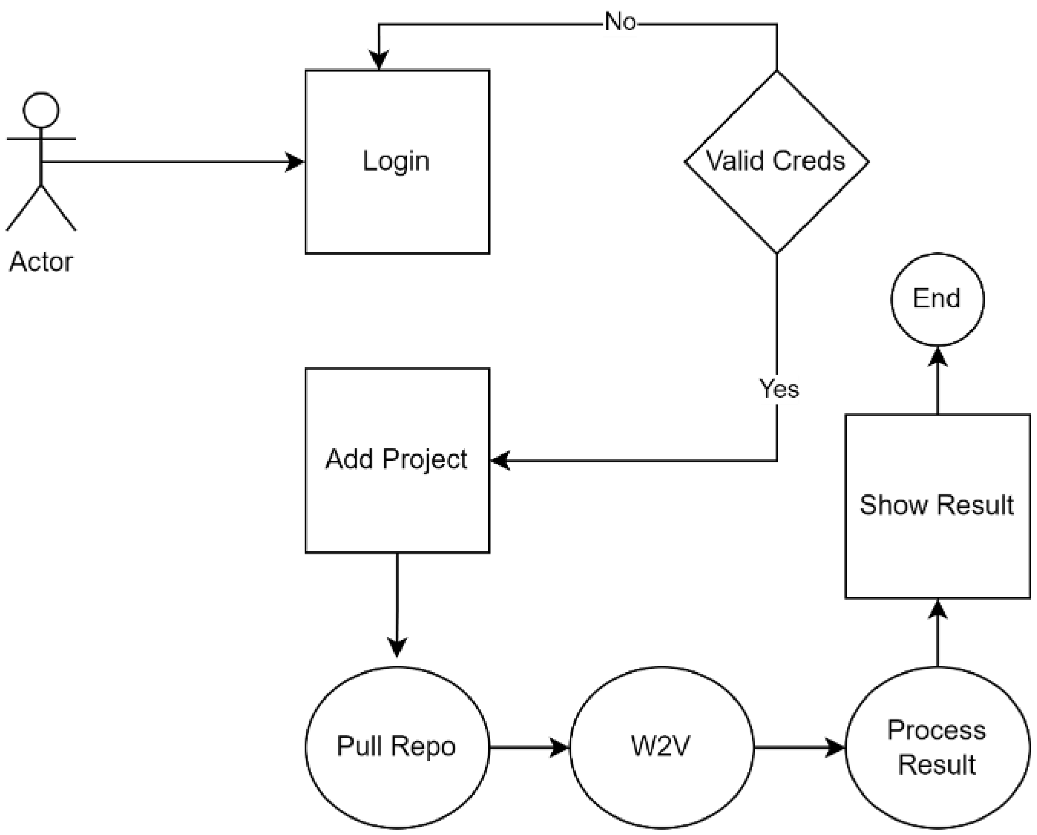

Figure 1.1: SafeScript workflow diagram

This structured approach ensures that the system not only identifies security vulnerabilities effectively but also facilitates continuous improvement in code security through iterative scans and fixes.

2 Methods

2.1 Technologies

The SafeScript project utilizes a comprehensive stack of technologies to deliver a robust and efficient solution for identifying and addressing security vulnerabilities in software projects. The key technologies employed in this project include:

- **Django Framework**: A high-level Python web framework that encourages rapid development and clean, pragmatic design.
- **Python**: The primary programming language used for developing the backend logic and machine learning models.
- **SQLite v3**: A lightweight, disk-based database used for storing user credentials, project details, and scanning results.
- **JavaScript**: Used for adding interactive elements to the web pages and handling client-side logic.
- **HTML**: The standard markup language for creating web pages.
- **CSS**: A style sheet language used for describing the presentation of the web pages.

- **Bootstrap**: A front-end framework used for developing responsive and mobile-first web pages.
- **Ajax**: A set of web development techniques used to create asynchronous web applications.
- **jQuery**: A fast, small, and feature-rich JavaScript library used to simplify HTML DOM tree traversal and manipulation.
- **GitPython**: A Python library used to interact with Git repositories.

2.2 Implementation

The implementation of the SafeScript project involves several key components and workflows designed to ensure seamless operation and effective vulnerability detection.

2.2.1 User Options

Users are provided with two primary options for submitting their code for analysis:

1. **Single File Import** Users can upload a single Python source file.
2. **Repository Cloning** Users can provide a link to a code repository (e.g., GitHub, Bitbucket), which the system will clone and analyze.

2.2.2 Core Functionalities

The core functionalities of the system as follows:

1. **User Authentication**: Users authenticate themselves using a username and password to access the system.
2. **Project Creation and Management**: Users create and manage projects, selecting either to import a single file or clone an entire repository.
3. **Code Retrieval**: The system fetches the code from the specified source.
4. **Model Selection** During the vulnerability detection phase, users can choose between the ChatGPT API and the LSTM model. This flexibility allows users to leverage the strengths of both models, ensuring robust and reliable security analysis.
5. **Data Conversion**: The retrieved code is converted into word2vec (w2v) format, preparing it for vulnerability analysis.
6. **Vulnerability Detection**: Users can select either the integrated ChatGPT API or the LSTM model for vulnerability detection.
7. **Report Generation**: The system generates detailed reports outlining detected vulnerabilities, their severity, and recommendations for fixes.
8. **Rescan Capability**: Users can rescan their code after making fixes to ensure that vulnerabilities have been addressed.

2.2.3 Frontend and Backend Integration

The frontend, built with HTML, CSS, Bootstrap, and JavaScript (including jQuery and Ajax), provides a user-friendly interface for interaction. The backend, developed using Django, handles the business logic, user authentication, project management, code retrieval, data conversion, vulnerability detection, and report generation. SQLite v3 is used for database management, ensuring efficient data storage and retrieval.

2.2.4 AI Integration

The AI integration in the SafeScript project leverages advanced machine learning models to enhance the accuracy and efficiency of vulnerability detection. At the moment the system is integrated with GPT API and LSTM model, which we have trained over past labs. Beside, those two the system have the capability to be integrated with other AI models as well.

Integrated ChatGPT API

The system integrates the ChatGPT API, enabling it to utilize the powerful capabilities of the ChatGPT model for analyzing code and identifying potential security vulnerabilities. This integration allows the system to provide context-aware analysis and generate comprehensive vulnerability reports. This integration allows the user to detect deffirent types of vulnerabilities in their python code.

LSTM Model

In addition to the ChatGPT API, the system incorporates an in-house developed LSTM (Long Short-Term Memory) model. This model is specifically trained to detect security vulnerabilities in Python code. Users have the option to select the LSTM model for vulnerability detection, benefiting from its specialized focus and accuracy. The LSTM model for has the capability to detect for specific type of vulnerabilities. The system can detect Cross-Site Scripting (XSS), Path Disclosure, Remote Code Execution, Command Injection. The system has the capability to support more vulnerabilities types through LSTM model in the future.

By integrating these AI technologies, the SafeScript project enhances its capability to detect and report vulnerabilities effectively, providing users with actionable insights to improve the security of their software projects.

2.3 Data Storage

The database for the Vulnerability Scanner project is structured using SQLite v3, providing a lightweight, disk-based database engine. This database includes several tables designed to handle user authentication, project management, and feedback functionalities.

The auth_group table manages the different user groups within the system, facilitating role-based access control. The auth_user table stores user information, including login credentials and profile details, to support user authentication and management. The auth_user_groups table links users to the groups they belong to, enabling group-based permissions and access control. Similarly, the auth_user_user_permissions table maps individual users to specific permissions, allowing for fine-grained access control. The auth_permission table defines various permissions that can be assigned to users or groups, ensuring secure and controlled access to different system functionalities.

The django_admin_log table records administrative actions performed within the system, providing an audit trail for monitoring and security purposes. The django_content_type table is used by the Django framework to keep track of all the models installed in the project, facilitating content type management. The authtoken_token table stores authentication tokens for users, supporting secure API access.

For project-specific data, the main_feedback table collects user feedback, including the subject and body of feedback messages, along with timestamps and user associations. The main_price table manages pricing information related to the system's services or products, along with timestamps for creation and updates. The main_response table records responses generated by the system, including response type, token usage, timestamps, and associated user and request information. The main_log table logs user interactions with the system, capturing details such as user identity, URL accessed, user agent, IP address, request method, status code, and timestamps. Lastly, the main_project table manages project information, including project identifiers, names, descriptions, repository details, status, timestamps, and associated user information.

These tables collectively support the core functionalities of the Vulnerability Scanner project, including user authentication, project management, feedback collection, and logging of interactions and administrative actions. This structured approach ensures efficient data management and retrieval, facilitating seamless operation and robust security.

Figure 2.1 shows the data storage in Sqlite database.

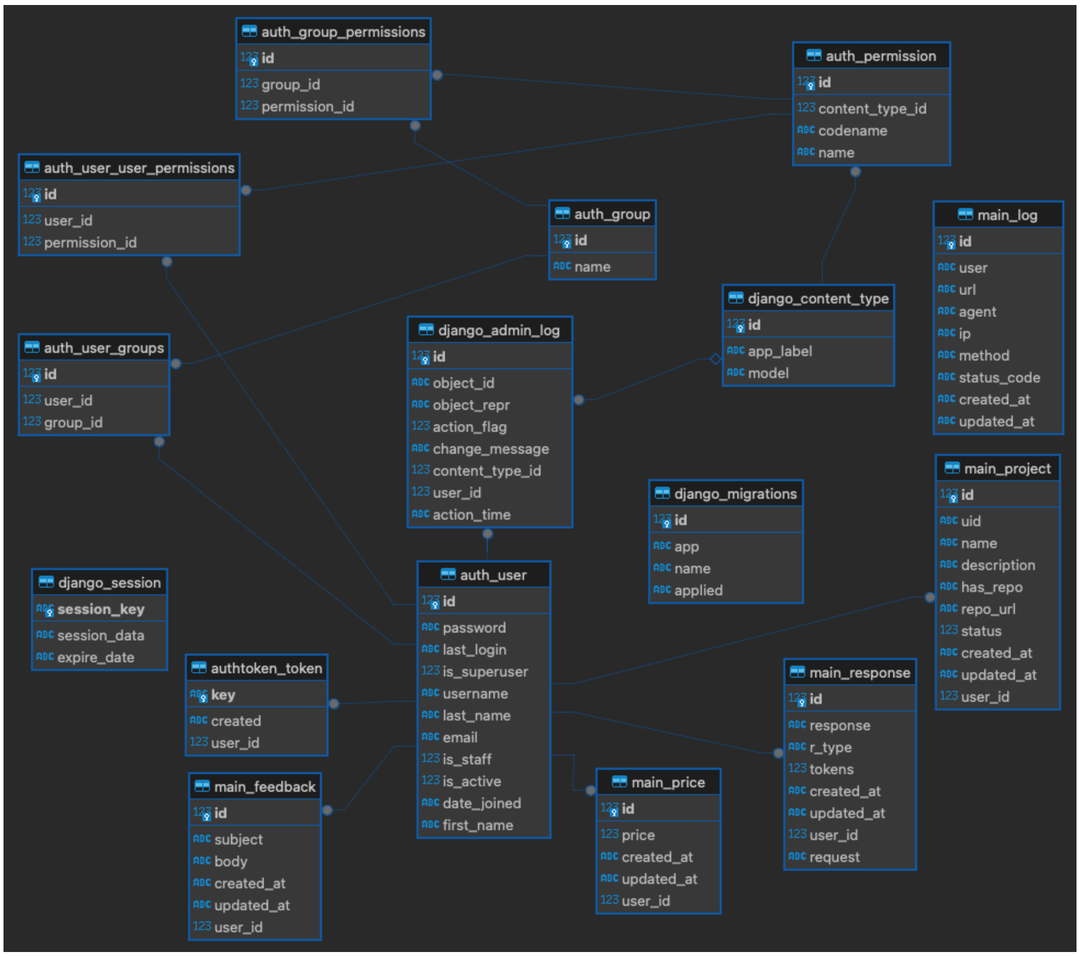

Figure 2.1: System storage ER diagram

2.4 Challenges

The development and implementation of the Vulnerability Scanner project presented several challenges, particularly in the integration of AI models and prompt engineering. These challenges required innovative solutions to ensure seamless functionality and reliable performance.

2.4.1 Integration of AI Models

The integration of AI models posed a significant challenge due to the need to support both the ChatGPT API and our in-house LSTM model. Each model has distinct requirements and operational characteristics, necessitating careful planning and execution to ensure smooth integration.

Challenge Details:

1. **Different Architectures**: The ChatGPT API and the LSTM model operate on fundamentally different architectures. The ChatGPT API, provided by OpenAI, is a transformer-based model optimized for natural language understanding and generation, while the LSTM model is a recurrent neural network (RNN) variant designed specifically for sequential data analysis.

2. **API Integration**: Integrating the external ChatGPT API involved ensuring secure and efficient communication with the OpenAI servers, handling API requests and responses, and managing API rate limits and potential latency.

3. **Internal Model Management**: The LSTM model, being developed and hosted internally, required robust deployment infrastructure, including model serving, scaling, and maintenance.

Solution:

1. **Unified Interface**: We developed a unified interface that abstracts the differences between the two models. This interface allows the system to interact with either model seamlessly, providing a consistent experience regardless of the selected AI model.

2. **API Wrapper**: For the ChatGPT API, we implemented a wrapper that handles API calls, error management, and response parsing. This wrapper ensures reliable communication and error handling when interacting with the external API.

3. **Model Server**: For the LSTM model, we set up a dedicated model server using Django and REST framework, enabling efficient model inference requests and scalable deployment. This server handles requests from the main application, processes the input data, and returns the analysis results.

4. **Configuration Management**: A configuration management system was implemented to manage model-specific settings and parameters, allowing the system to dynamically switch between the ChatGPT API and the LSTM model based on user preference.

2.4.2 Prompt Engineering

Prompt engineering was another critical challenge due to the variability in responses from the AI models. Effective prompt engineering is essential for obtaining accurate and relevant results from the models.

Challenge Details:

1. **Response Variability**: The ChatGPT model, in particular, can generate diverse responses depending on the prompt's phrasing. This variability can lead to inconsistent results, making it challenging to ensure the reliability of vulnerability detection.

2. **Context Management**: Ensuring that the models receive sufficient context to accurately analyze code and identify vulnerabilities required careful crafting of prompts.

3. **Error Handling**: Handling ambiguous or incorrect responses from the models necessitated robust error detection and correction mechanisms.

Solution:

1. **Iterative Prompt Development**: We adopted an iterative approach to prompt development, continually refining the prompts based on testing and feedback. This iterative process involved experimenting with different prompt structures and evaluating their impact on response quality.

2. **Template-Based Prompts**: We designed template-based prompts that ensure consistency in the input provided to the models. These templates include placeholders for code snippets and context information, standardizing the prompts and reducing response variability.

3. **Context Enrichment**: To provide sufficient context, we implemented mechanisms to extract relevant information from the code and include it in the prompts. This includes details about the code structure, dependencies, and known issues.

4. **Post-Processing**: We developed post-processing algorithms to analyze and validate the responses from the AI models. These algorithms check for consistency, filter out irrelevant information, and ensure that the final output meets the required standards.

By addressing these challenges through thoughtful solutions, we were able to integrate the AI models effectively and enhance the reliability of the Vulnerability Scanner project.

This ensures that users receive accurate and actionable insights for improving the security of their software projects.

3 Results

4 Discussion

A Code

B Math

C Dataset

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese MasterLab selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die MasterLabin gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, 29. Juni 2024

AUTHOR