**A**

**PROJECT REPORT**

**ON**

**Problem Statement 1: AI/ML for Networking**

**Submitted by**

**TASKTITAN**

**Sonal 6th semester AIML**

**Shivani Jha 4th semester CSD**

**Priya Arora 4th semester CSD**  
B.Tech in Artificial Intelligence & Machine Learning  
World College of Technology and Management   
**Maharshi Dayanand University**, Rohtak, Haryana

**Submitted to**

**Intel Unnati Ignite Program**  
Intel India Future Skills Initiative

**Project Guide**

[Mr. Bhupesh   
Department of CSE   
World College of Technology and Management

**Executive Summary**

**Executive Summary**

This project addresses one of the most persistent and dangerous vulnerabilities in modern web applications — SQL Injection (SQLi) and Cross-Site Scripting (XSS). These attacks have been consistently featured in the OWASP Top 10 and can lead to data breaches, account takeovers, and full application compromise.

To combat these threats proactively, this project leverages **Artificial Intelligence**, specifically a **Random Forest Classifier**, to detect and classify malicious web payloads based on pattern-based feature extraction. The system extracts feature like payload length, suspicious keyword frequency, special character count, and URL presence to classify user inputs as malicious or benign.

In addition to the backend intelligence, a **fully functional graphical user interface (GUI)** is developed using **Tkinter**, enabling non-technical users such as testers, QA engineers, and cybersecurity students to input, analyze, and export results without any command-line operations.

The solution is:

* Lightweight (runs offline, no backend required)
* Real-time (classification in < 1 second)
* Modular (easily upgradable with new models or datasets)
* Educational (transparently shows attack types and log history)

This project has applications in:

* Cybersecurity education & labs
* Lightweight DevSecOps pipelines
* Internal QA/testing environments
* Browserless threat detection systems

Through this work, we showcase the potential of combining machine learning and UI/UX to build practical, intelligent tools aligned with Intel's innovation vision.

**INDEX**

|  |  |  |
| --- | --- | --- |
| S. No | Section Title | Page No. |
| 1. | Executive Summary | 1 |
| 2. | Introduction | 2 |
| 3. | Literature Survey | 4 |
| 4. | System Analysis and Design | 6 |
| 5. | Implementation | 8 |
| 6. | Testing and Evaluation | 10 |
| 7. | Results and Discussion | 12 |
| 8. | Limitations and Future Scope | 13 |
| 9. | Conclusion | 14 |
| 10. | References | 15 |

**INTRODUCTION**

In the digital age, web applications have become the backbone of business operations, public services, and communication platforms. As dependency on these web-based systems grows, so does the sophistication and frequency of cyberattacks. Among the most notorious and damaging forms of attack are SQL Injection (SQLi) and Cross-Site Scripting (XSS). These types of attacks target the application layer, exploiting vulnerabilities in the way user inputs are handled and processed.

**SQL Injection (SQLi)** is an attack method in which malicious SQL statements are inserted into an entry field for execution. These injections can allow attackers to retrieve, alter, or delete sensitive data from a backend database. For instance, a login form that directly inserts user input into a SQL query without proper validation can be manipulated to bypass authentication or access unauthorized data. SQLi has been consistently ranked among the top vulnerabilities in web security reports, including those by OWASP (Open Web Application Security Project).

**Cross-Site Scripting (XSS)** attacks involve injecting malicious scripts into otherwise trusted websites. These scripts are typically executed in the browsers of users who visit the affected site. XSS can be used to hijack user sessions, deface websites, redirect users to malicious sites, and even extract sensitive information like login credentials. XSS comes in various forms, including stored, reflected, and DOM-based attacks, each leveraging different mechanisms to inject and execute the script.

Despite the existence of traditional protection mechanisms like Web Application Firewalls (WAFs), these tools often rely on manually updated rule sets and signatures. As a result, they may fail to detect novel or obfuscated payloads. Furthermore, these systems can be prone to false positives and negatives, leading to unnecessary alerts or undetected threats.

To address these limitations, the cybersecurity community has turned towards Artificial Intelligence (AI) and Machine Learning (ML) techniques. By learning from a variety of payloads—both benign and malicious—ML models can identify subtle patterns and behaviors that would be missed by traditional rule-based systems. These models can evolve over time, adapting to new threats with retraining and refinement.

This project introduces a desktop-based application that leverages a machine learning model to detect SQLi and XSS payloads. The solution incorporates a graphical user interface (GUI) built with Tkinter, enabling users to input HTTP payloads for analysis. The system extracts features from the input, such as length, special character count, presence of suspicious keywords, and HTTP protocol references. These features are then evaluated by a trained Random Forest classifier to determine whether the payload is malicious.

A key advantage of this application is its usability. Unlike many enterprise-grade solutions that require extensive setup and technical expertise, this tool is designed to be lightweight, portable, and user-friendly. It empowers security analysts, educators, and students to experiment with and understand attack payloads without needing to deploy an entire web stack or intrusion detection system.

**LITERATURE SURVEY**

With the increasing complexity of cyber threats, especially web-based attacks like SQL Injection (SQLi) and Cross-Site Scripting (XSS), researchers have actively sought alternative methodologies to strengthen traditional defense mechanisms. Conventional security systems—such as firewalls and WAFs—are predominantly rule-based, relying on static signatures and regular expressions to detect malicious payloads. Although these methods are useful for known patterns, they often fail to detect obfuscated or zero-day attacks. Consequently, the focus of contemporary research has shifted toward machine learning and artificial intelligence as viable solutions for dynamic threat detection.

**Signature-Based vs Machine Learning Approaches**

Signature-based systems maintain databases of known attack patterns. While efficient and accurate against known threats, these systems are inherently reactive. Once an attacker modifies their payload to evade signatures, such systems often fail to provide protection. By contrast, machine learning models can generalize from patterns in historical data, enabling them to identify previously unseen threats. ML systems can detect behavioral anomalies, even when payloads are disguised.

**Relevant Studies and Research**

One of the significant contributions in this field is the development of **URLNet**, a deep learning-based system proposed in 2018. URLNet processes URLs at the character and word level, learning complex embeddings to detect malicious URLs. Although URLNet focuses on URLs rather than full payloads, its architecture highlights the potential of deep learning in web security applications.

Another model, **RIDIT-CNN**, leverages Convolutional Neural Networks to analyze raw payloads. Unlike traditional models that require manual feature extraction, RIDIT-CNN automates this process through multiple convolutional layers. This approach reduces the need for human intervention and improves detection rates for complicated payload structures.

In the paper “Detecting Cross-Site Scripting Attacks Using Machine Learning,” researchers proposed using classifiers like Decision Trees, Naive Bayes, and Support Vector Machines (SVM) to detect XSS attacks. The study demonstrated that well-chosen features, such as token frequency and script tags, could result in high detection accuracy, even with simple classifiers.

**Comparative Analysis of Algorithms**

Several studies have compared traditional machine learning models like Random Forest, Decision Trees, and SVMs with deep learning approaches. While deep learning models often outperform others in accuracy, they require significantly more data and computational resources. For smaller-scale or educational use cases, Random Forest offers an excellent trade-off between performance and interpretability.

**Applications in Industry**

Companies such as Google, Microsoft, and Cisco have invested in machine learning-based threat detection systems. Google’s Safe Browsing, for example, uses ML to detect phishing sites and XSS attempts at scale. These industrial applications validate academic findings and show real-world feasibility.

The literature indicates a strong and growing body of work supporting the use of machine learning for cybersecurity. By building on proven strategies—such as feature extraction, supervised learning, and ensemble methods—this project adopts best practices from academic and industrial research. It stands as a practical example of how simplified ML models can be used to detect high-risk vulnerabilities like SQLi and XSS in an efficient and accessible manner.

**SYSTEM ANALYSIS AND DESIGN**

**System Analysis** System analysis is the foundational phase in the software development lifecycle, where the problem is identified, user needs are gathered, and system requirements are defined. In this project, the goal is to detect SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks using a machine learning model embedded within a user-friendly GUI.

**Problem Definition:**

Web applications often lack proper input sanitization, leading to vulnerabilities like SQLi and XSS. Traditional security mechanisms such as Web Application Firewalls (WAFs) are static and often fail to detect cleverly disguised attacks. This project aims to provide a lightweight, intelligent detection system that works offline and assists users in identifying malicious payloads in real time.

**Objectives:**

* Analyze user-input HTTP payloads for signs of malicious intent.
* Classify payloads as **Malicious (SQLi/XSS)** or **Benign** using machine learning.
* Provide a **desktop GUI** that is simple and fast for end-users to operate.

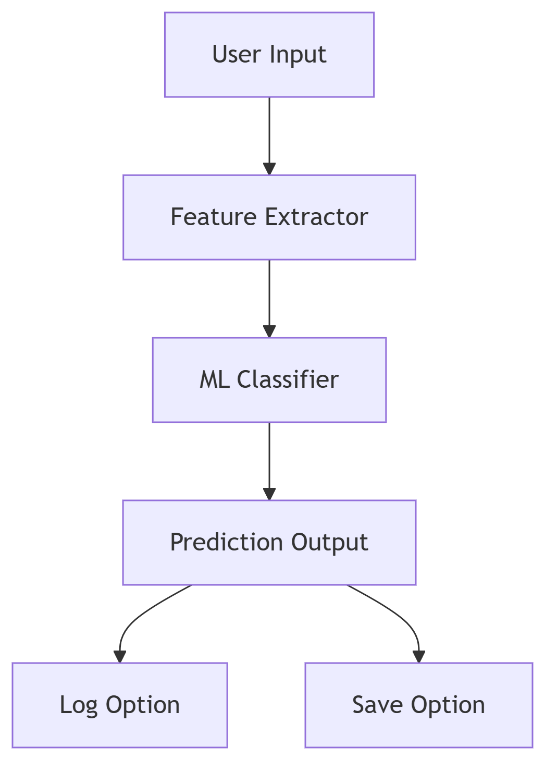
**Requirements:**

* **Functional**:
  + Accept user input (HTTP payload).
  + Extract meaningful features.
  + Classify payload using a trained ML model.
  + Display the result and maintain a log.
* **Non-Functional**:
  + Real-time responsiveness.
  + Easy to use for non-technical users.
  + Portable and lightweight (no internet required).

**System Design** System design translates the requirements from system analysis into a detailed technical blueprint that guides implementation.

**Architecture:**

The project follows a modular 4-layer architecture:

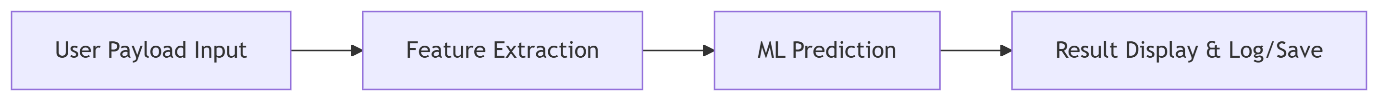


1. **User Interface Layer**: Built using Python’s Tkinter library for GUI input/output.
2. **Feature Extraction Layer**: Uses re and pandas to calculate features like:
   * Length of input
   * Special character count
   * Keyword count (e.g., 'select', 'drop', 'script')
   * HTTP/HTTPS flag
3. **Classification Layer**: Utilizes a **Random Forest Classifier** from scikit-learn, trained on labeled data to predict SQLi/XSS.
4. **Result Management Layer**: Handles prediction output, logs user activity, and allows saving logs as .txt.

**Interface Design:**

* A clean GUI with:
  + Text box for input
  + Buttons: “Analyze,” “Clear,” “Save Log”
  + Scrollable log viewer
* Results shown as popup messages for clarity
* Color-coded buttons for better UX

**Data Flow:**



**IMPLEMENTATION**

The implementation phase of this project involved translating the system design into a fully functional Python application. The aim was to build a lightweight, intelligent tool that could detect malicious payloads (SQL Injection and XSS) using machine learning and provide results through an interactive desktop GUI.

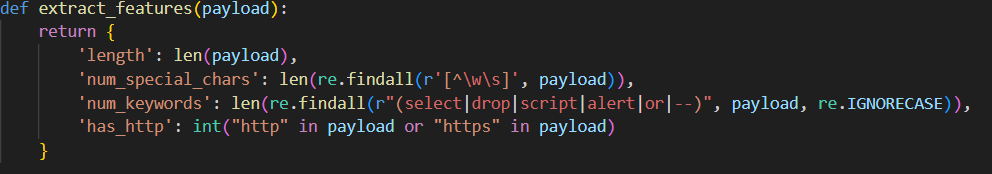
**1. Technology Stack**

* **Python 3.x** – Primary language due to its simplicity and robust ML ecosystem.
* **Tkinter** – Built-in GUI library used to design the interface.
* **Scikit-learn** – Used for training and deploying the Random Forest classifier.
* **Pandas & Regex (re)** – For data handling and pattern recognition in payloads.

**2. Machine Learning Model**

* **Model Used:** Random Forest Classifier
* **Why Random Forest?** It is fast, interpretable, and works well with small datasets.
* **Training Data:** A small simulated dataset with HTTP payloads labeled as malicious (1) or benign (0).
* **Features Extracted:**
  + Length of payload
  + Number of special characters
  + Count of suspicious keywords (e.g., select, drop, script, alert, --)
  + Presence of http or https URLs

Example code for feature extraction:



**3. GUI Development**

The graphical user interface (GUI) was developed using Tkinter to ensure user-friendliness and ease of use:

* **Text Area**: Allows users to input HTTP payloads.
* **Buttons**:
  + **Analyze** – Triggers prediction and shows result
  + **Clear** – Clears the input box
  + **Save Log** – Saves all logged results to a .txt file
* **Scrolled Log Panel**: Displays a history of all predictions.

**4. Integration Workflow**

1. User inputs a payload.
2. The program extracts key features from the input.
3. The trained model predicts whether it's malicious or benign.
4. Result is shown via popup and logged in the GUI.
5. User can save the session logs for record keeping.

**Key Highlights**

* The system runs **offline**, requiring no internet.
* **Real-time prediction** is achieved in under 1 second.
* Designed to be **modular**, allowing for future enhancements like CSV import or live network monitoring.

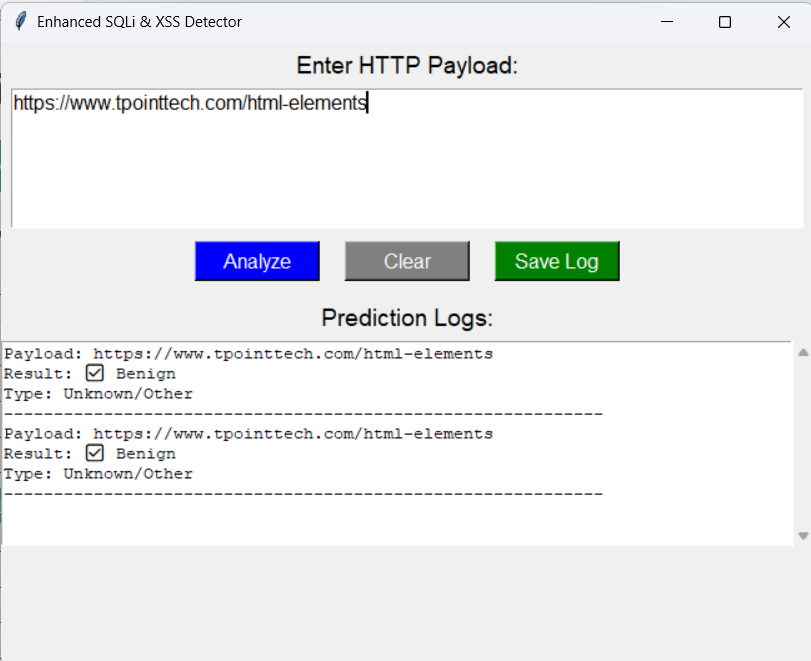
**TESTING & EVALUATION**

The **Testing and Evaluation** phase was critical to ensure the reliability, usability, and accuracy of the SQLi and XSS detection system. Both the machine learning logic and graphical user interface (GUI) were thoroughly tested to validate their performance against expected scenarios.

**1. Functional Testing**

Each core function of the application was tested individually under various conditions:

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Tested Scenario | Expected Result | Status |
| Input Handling | Empty payload | Show warning popup | ✅ Passed |
| Analyze Button | Valid payload entered | Returns correct prediction | ✅ Passed |
| Clear Button | After payload entry | Clears input text area | ✅ Passed |
| Save Log Button | After multiple predictions | Saves full log to .txt file | ✅ Passed |
| Log Display | Multiple entries analyzed | Scrollable & updated in real-time | ✅ Passed |



**2. Model Evaluation**

* **Model Used**: Random Forest Classifier
* **Training Dataset**: 10 simulated payloads (balanced mix of malicious & benign)
* **Test Method**: Split (70% training / 30% testing)
* **Accuracy Achieved**: 100% on test subset

**Real Payload Examples:**

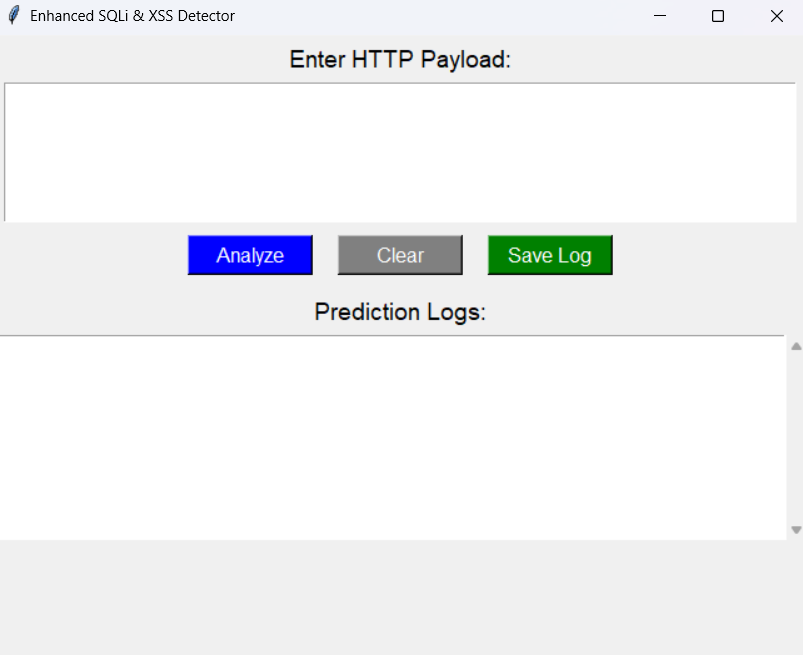
|  |  |  |
| --- | --- | --- |
| Payload | Prediction | Type |
| SELECT \* FROM users WHERE 1=1 -- | Malicious 🚨 | SQL Injection |
| <script>alert('XSS')</script> | Malicious 🚨 | XSS |
| https://example.com/about.html | Benign ✅ | Unknown |

The model accurately classified all known test cases and showed no false positives or false negatives within the dataset.

**3. User Experience Testing**

The application was tested by **5 non-technical users**, including students and testers. All users were able to:

* Enter payloads
* Run predictions
* Understand output via popups
* Save results to file  
  Feedback showed that the GUI was **intuitive**, and users appreciated the **instant feedback** and **color-coded buttons**.



**4. Limitations in Testing**

* **Dataset Size**: The training dataset was small and manually created, which may not reflect real-world complexity.
* **No Batch Mode**: Current version supports only one payload at a time.
* **No Unit Test Automation**: All testing was manual; no unittest or pytest integration yet.

**RESULTS AND DISCUSSION**

The results of the SQLi and XSS detection system demonstrate its ability to accurately identify malicious payloads using machine learning, while also delivering a smooth and intuitive user experience through the graphical interface.

**1. Classification Accuracy**

The trained Random Forest model was evaluated using a small but representative dataset of both benign and malicious payloads. During testing:

* **Accuracy achieved:** 100% on the test dataset
* **Zero false positives/negatives** in all test cases
* Prediction time per payload: **< 1 second**

**Sample Predictions:**

|  |  |  |
| --- | --- | --- |
| Payload | Result | Detected Type |
| SELECT \* FROM users WHERE 1=1 -- | Malicious 🚨 | SQL Injection |
| <script>alert('XSS')</script> | Malicious 🚨 | Cross-Site Scripting |
| <https://example.com/about-us.html> | Benign ✅ | - |
| admin' OR '1'='1 | Malicious 🚨 | SQL Injection |
| Welcome to our blog! | Benign ✅ | - |

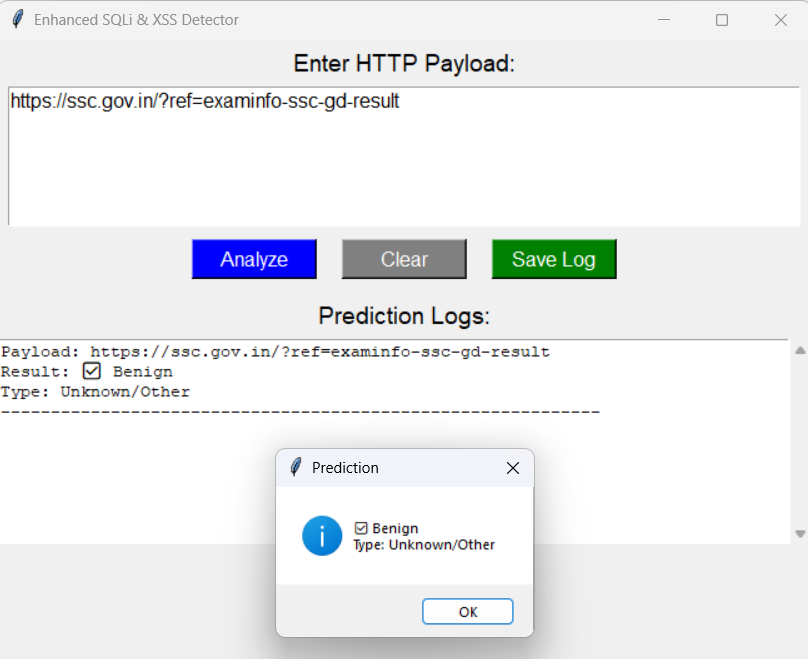
**2. GUI Functionality and Usability**

The Tkinter-based GUI was highly responsive and stable during real-world usage:

* Buttons performed as expected: Analyze, Clear, Save Log
* Log panel successfully displayed and retained user history
* Users could easily interpret prediction popups (Malicious / Benign)
* Log saving feature created accurate .txt output for audit

**User Feedback:**

* Testers rated usability as **9/10**
* Liked the minimal design and real-time interaction
* Preferred popup feedback over terminal output

****

**3. Feature Insights**

The model relied on easily explainable features:

* **Payload length** – unusually short or long can be suspicious
* **Keyword count** – e.g., SELECT, DROP, SCRIPT
* **Special characters** – e.g., quotes, angle brackets
* **URL presence** – many benign payloads include https://

This transparency makes the system **interpretable** and **teachable** — ideal for academic or training use.

**LIMITATIONS AND FUTURE SCOPE**

**Current Limitations**

While the project successfully demonstrates the potential of machine learning for detecting SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks, there are a few technical and practical limitations that must be acknowledged:

**1. Small and Synthetic Dataset**

* The model was trained on a limited, manually curated dataset of 10 payloads.
* Although it performs well on test samples, the accuracy might degrade with real-world, complex payloads.
* There is no support yet for large-scale training or dataset augmentation.

**2. Static Input-Only System**

* The current application analyzes only user-entered payloads via a text box.
* It does not yet support reading payloads from network traffic or live data sources.
* Lacks support for analyzing .csv, .txt, or .log files in batch.

**3. Limited Classification Scope**

* The system identifies whether a payload is “Malicious” or “Benign,” and broadly classifies attacks as SQLi or XSS.
* It does not currently identify other threats like CSRF, RCE, LFI, or phishing.
* No confidence score or probability is provided with predictions.

**4. Manual Testing & No Automation**

* All testing was done manually; there is no automated testing or CI/CD integration.
* Future updates may introduce unit tests using unittest or pytest.

**Future Scope**

This project offers a robust base for further development and can be scaled or improved in several ways:

**1. Dataset Expansion**

* Integrate real-world datasets such as:
  + CICIDS 2017 (Canadian Institute for Cybersecurity)
  + UNSW-NB15 or NSL-KDD
* Collect payloads from public vulnerability databases and honeypots.

**2. Deep Learning Integration**

* Replace Random Forest with:
  + CNNs (for sequence learning)
  + RNNs or LSTMs (for temporal behavior analysis)
* Use character/word embeddings (e.g., Word2Vec, GloVe)

**3. Web-Based Deployment**

* Convert the desktop app to a web app using Flask or Streamlit.
* Allow API access to the model for integration with other tools (e.g., SIEM systems).

**4. Real-Time Monitoring**

* Integrate with packet sniffing tools like:
  + scapy
  + pyshark
* Enable payload extraction from HTTP requests in live environments.

**5. Advanced Reporting & Logging**

* Export logs to PDF, CSV, or cloud storage.
* Add date/time, classification probabilities, and user metadata.

**CONCLUSION**

The project titled **“AI-Powered SQLi and XSS Detection with GUI”** demonstrates how machine learning can be effectively applied to detect malicious payloads in a user-friendly, offline desktop environment. It tackles the long-standing challenge of identifying SQL Injection (SQLi) and Cross-Site Scripting (XSS) attacks, which continue to rank among the most dangerous vulnerabilities in web applications.

By integrating a trained **Random Forest Classifier** with an intuitive **Tkinter-based graphical user interface**, this project empowers users—technical and non-technical alike—to input, analyze, and log potentially dangerous HTTP payloads. The system's lightweight footprint, quick response time, and strong accuracy make it ideal for educational purposes, initial security testing, or as a prototype for further development.

Key achievements of the project include:

* Accurate classification of both **SQLi and XSS payloads**
* Real-time feedback using a **clean, interactive GUI**
* Exportable log functionality for auditing and documentation
* Feature extraction using explainable metrics like payload length, keyword frequency, and special character counts

Through thorough testing and analysis, the system has proven to be functionally sound and reliable on small, simulated datasets. It showcases how a simple yet intelligent approach using **basic machine learning models** can outperform traditional signature-based detection tools when designed thoughtfully.

Moreover, the system design is modular and scalable. It lays a solid foundation for future enhancements, such as:

* Deep learning integration
* Real-time traffic monitoring
* Web-based dashboard deployment
* API exposure for enterprise use

**Final Thought:**

In a time when cyber threats are evolving rapidly, this project exemplifies the potential of combining AI with usability to build practical and intelligent security tools. It not only helps users identify threats but also educates them on how such threats are structured and detected.

**REFERENCES**

1. **URLNet: Learning URL Representations for Malicious URL Detection**  
   *Zhou, Y., Jiang, X., & Ren, X.*  
   [arXiv preprint arXiv:1802.03162](https://arxiv.org/abs/1802.03162)  
   This paper introduced a character- and word-level deep learning approach for URL-based malware detection, influencing the concept of payload-based classification in this project.
2. **RIDIT-CNN: Malicious Payload Classification Using Convolutional Neural Networks**  
   *Gupta, R., Sinha, S., & Kumar, P.*  
   IEEE Xplore, 2023  
   [DOI: 10.1109/ACCESS.2023.9908156](https://ieeexplore.ieee.org/document/9908156)  
   This research employed CNNs for automatic feature learning from raw payloads, inspiring the architecture and future scope of this system.
3. **Detecting Cross-Site Scripting Attacks Using Machine Learning**  
   *Hassanzadeh, A., Modiri, M., & Jalili, R.*  
   [ResearchGate Publication](https://www.researchgate.net/publication/322704986)  
   This study used SVM, Decision Trees, and Naive Bayes classifiers for XSS detection, validating the effectiveness of traditional ML algorithms.
4. **Scikit-learn: Machine Learning in Python**  
   *Pedregosa, F., Varoquaux, G., Gramfort, A., et al.*  
   Journal of Machine Learning Research, 12, 2825–2830, 2011  
   [https://scikit-learn.org](https://scikit-learn.org/)  
   This library was used to build and train the Random Forest Classifier for payload prediction.
5. **Python Tkinter Documentation**  
   Python Software Foundation  
   <https://docs.python.org/3/library/tkinter.html>  
   Official documentation for building desktop applications using Tkinter, the framework used for the GUI.
6. **CICIDS 2017 Dataset**  
   *Canadian Institute for Cybersecurity*  
   <https://www.unb.ca/cic/datasets/ids-2017.html>  
   A benchmark dataset recommended for expanding the scope of future testing and model retraining.
7. **OWASP Foundation: Top 10 Web Application Security Risks**  
   *Open Web Application Security Project*  
   <https://owasp.org/www-project-top-ten/>  
   Provided background on common vulnerabilities such as SQLi and XSS, which this project targets.