**Real-Time Network Intrusion Detection Using Wireshark and Advanced Ensemble Learning Techniques**

**Software Requirements Specification**

Version 1.1



**Group Id:** F24PROJECTCE3A6

**Supervisor Name:** Laraib Sana

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date (dd/mm/yyyy)** | **Version** | **Description** | **Author** |
| 08/12/2024 | 1.0 | Introduction of the project | BC210424643 BC210401276 |
| 01/05/2025 | 1.1 | Introduction of the project | BC210424643 |

**Table of Contents**

1. [Scope (of the project)](#scope)
2. [Functional Requirements Non Functional requirements](#FRNFR)
3. [Use Case Diagram](#UCD)
4. [Usage Scenarios](#UCS)
5. [Adopted Methodology](#Adopted)
6. [Work Plan (Use MS Project to create Schedule/Work Plan)](#Gantt)

**SRS Document**

**Scope of Project:**

This project focuses on the design and development of a **Real-Time Network Intrusion Detection System (IDS)** that leverages machine learning and real-time data processing to identify and classify malicious network behavior. As cyber threats continue to evolve and grow in complexity, traditional rule-based detection mechanisms often fall short in terms of adaptability and real-time responsiveness. This system addresses those limitations by incorporating **Wireshark-based packet capturing**, advanced **data preprocessing**, and **ensemble machine learning models** for dynamic, accurate threat detection.

The system is intended to serve a wide range of users, including **cybersecurity analysts**, **network administrators**, and **researchers**, offering them an intuitive and powerful tool to monitor network traffic, detect anomalies, and assess threat patterns with minimal manual intervention.

The project adopts a **modular architecture** to ensure that all core functions—data capture, preprocessing, detection, and result presentation—are independently scalable and maintainable. The integration of a **web application** makes the system accessible and user-friendly, enabling both technical and non-technical users to interact with it effectively.

### ****Key Objectives and Deliverables****

* **Real-Time Packet Capture**: Use Wireshark to capture live network traffic from the system’s active network interface and save the output in a structured .csv format for further analysis.
* **Preprocessing and Feature Engineering Pipeline**:
  + Clean and normalize raw data.
  + Encode categorical fields such as protocol types.
  + Extract and label features relevant to intrusion detection (e.g., IP addresses, ports, packet size, timestamp, protocol).
* **Machine Learning Integration**:
  + Train ensemble learning models (TabNet, CatBoost, LightGBM) on labeled network traffic data.
  + Evaluate model performance using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.
  + Deploy the best-performing model(s) for use in real-time and batch classification scenarios.
* **Web Application Development**:
  + Build an intuitive, role-based web interface using Flask or Django.
  + Enable users to upload traffic files, trigger real-time analysis, and view classification results.
  + Provide graphical result summaries, downloadable reports, and an administrative dashboard.
* **User Feedback System**:
  + Allow users to rate or correct detection results.
  + Store feedback for potential model retraining and continuous improvement.

### ****Tools and Technologies****

| ****Component**** | ****Tool/Technology**** |
| --- | --- |
| **Programming Language** | Python |
| **Traffic Capture** | Wireshark |
| **Machine Learning** | PyTorch-TabNet, LightGBM, CatBoost, Scikit-learn |
| **Data Processing** | Pandas, NumPy |
| **Web Development** | Flask (or Django), HTML/CSS, Bootstrap |
| **Visualization** | Matplotlib, Seaborn |
| **Database** | SQLite (development), MySQL (optional for production deployment) |
| **IDE & Tools** | PyCharm, Jupyter Notebook, VS Code, GitHub |
| **Diagramming** | Draw.io, Lucidchart, MySQL Workbench |

### ****Final Product Description****

The final deliverable of this project will be a fully functional, **web-based Intrusion Detection System** capable of:

* **Real-Time and File-Based Intrusion Detection**: Capturing live traffic or uploading CSV-formatted data for instant analysis.
* **Automated Processing and Classification**: Using trained ML models to classify traffic as normal or malicious, with confidence scores and feature-based explanations.
* **User-Friendly Dashboard**:
  + Displays analysis results in real time.
  + Provides historical detection logs.
  + Allows result export in formats like CSV or PDF.
* **Security and Role Management**:
  + Admin and user roles with separate permissions.
  + Secure file handling and input validation.
* **Feedback-Driven Improvement**:
  + Users can submit feedback on detection results.
  + System logs feedback for future model optimization.

**Functional and Non-Functional Requirements:**

**Functional Requirements**

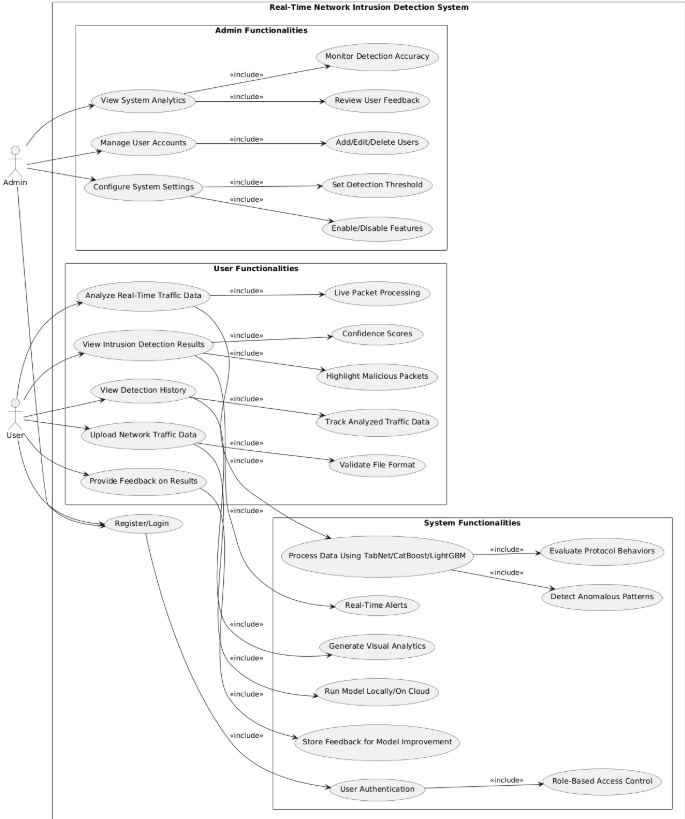
* **Traffic Capture:**
  + The system must use Wireshark to capture real-time network traffic data.
  + Captured data must be saved as a CSV file for preprocessing and analysis.
* **Data Upload:**
  + Users must upload CSV files containing network traffic data for analysis.
  + The system should validate the uploaded files for format and content consistency.
* **Data Preprocessing:**
  + Clean raw traffic data by handling missing or inconsistent values.
  + Encode categorical features such as protocol types and normalize numerical features like packet size and time intervals.
  + Label the dataset with normal and malicious traffic for supervised machine learning.
* **Model Training and Evaluation:**
  + Train ensemble models, including TabNet, CatBoost, and LightGBM, using the preprocessed dataset.
  + Evaluate models using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.
* **Intrusion Detection:**
  + Analyze uploaded network traffic data using trained machine learning models to detect potential intrusions.
  + Classify network traffic as normal or malicious and generate detailed results.
* **Web Application:**
  + Provide an intuitive interface for users to upload CSV files of network traffic data.
  + Display real-time analysis results, including key metrics and influencing factors.
  + Allow users to save or export the analysis results for further review.

**Non-Functional Requirements**

* **Performance:**
  + The system must process and analyze data within 2 seconds for real-time functionality.
* **Scalability:**
  + The system must handle datasets of up to 1 GB without performance degradation.
* **Usability:**
  + The web interface must be user-friendly and accessible to technical and non-technical users.
* **Reliability:**
  + The system should provide consistent results under various traffic loads.
* **Security:**
  + Ensure secure upload and processing of data to prevent unauthorized access or manipulation.
  + Validate input files to protect against malicious or corrupt data uploads.
* **Portability:**
  + The application must be compatible with multiple operating systems, including Windows and Linux.
* **Maintainability:**
  + Use a modular design to facilitate updates and additions, such as integrating new machine learning models.
* **Data Integrity:**
  + Ensure the integrity of processed and analyzed data, avoiding loss or corruption.

**These requirements provide a clear foundation for developing a robust, efficient, and user-friendly intrusion detection system.**

**Use Case Diagram(s):**



**Usage Scenarios:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Use Case Title** | **Use Case ID** | **Actions** | **Description** | **Alternative Paths** | **Preconditions** | **Postconditions** | **Author** | **Exceptions** |
| Capture Traffic | UC01 | 1. User launches Wireshark. 2. Selects the network interface. 3. Starts capturing network traffic. 4. Saves the traffic data as a CSV file. | This use case captures real-time network traffic using Wireshark and saves the captured traffic data for analysis. | 1. User selects incorrect interface and needs to restart the capture. 2. Save file as another format if CSV is unavailable. | Wireshark must be installed and configured. Network interface should be active. | A CSV file with captured traffic data is generated. | Wasif Nawaz | 1. Network interface unavailable. 2. Insufficient permissions to capture traffic. 3. Application crash. |
| Upload Data | UC02 | 1. User accesses the web app. 2. Uploads a CSV file. 3. System validates file format. 4. System preprocesses the data for analysis. | Allows users to upload captured network traffic data in CSV format to the web application for preprocessing and analysis. | User uploads an invalid file; system prompts for re-upload. Preprocessing can skip certain rows with invalid data. | A valid CSV file containing network traffic data must be available. | Data is preprocessed and ready for analysis. | Wasif Nawaz | 1. Invalid file format. 2. Missing or corrupt data. 3. Server fails to preprocess the file. |
| Preprocess Data | UC03 | 1. Clean the uploaded data. 2. Encode categorical features. 3. Normalize numerical values. 4. Label data as normal or malicious. | This use case prepares the uploaded network traffic data for machine learning analysis by cleaning, encoding, and normalizing it. | Skip preprocessing for small datasets that are already clean and formatted. | Uploaded data must be in CSV format. System must have preprocessing modules deployed. | Data is ready for machine learning analysis. | Wasif Nawaz | 1. Incomplete preprocessing due to corrupt data. 2. Incorrect encoding format. 3. Timeout during large file processing. |
| |  | | --- | | Model Training |  |  | | --- | |  | | UC04 | 1. Preprocessed data is loaded. 2. Models (TabNet, CatBoost, LightGBM) are trained using labeled data.  3. Metrics (e.g., accuracy, precision, recall) are calculated and stored. | This use case handles training the machine learning models using preprocessed and labeled datasets. | |  | | --- | | Allow re-training specific models if others fail. |  |  | | --- | |  | | |  | | --- | | Preprocessed and labeled data must be available. Trained models must be saved for deployment. |  |  | | --- | |  | | |  | | --- | | Trained models with performance metrics are available for analysis. |  |  | | --- | |  | | |  | | --- | | Wasif Nawaz |  |  | | --- | |  | | 1. Insufficient labeled data.  2. Training failure due to incompatible data formats or model errors. |
| Analyze Data | UC05 | 1. Preprocess the uploaded data. 2. Classify the traffic using trained models. 3. Generate intrusion detection results. | This use case uses trained machine learning models (TabNet, CatBoost, and LightGBM) to classify the uploaded data and identify potential intrusions. | Use an alternate model if the primary model fails. Skip classification for datasets with missing essential features. | Data must be preprocessed. Machine learning models must be trained and deployed. | Analysis results (normal or malicious traffic) are generated. | Wasif Nawaz | 1. Data preprocessing failure. 2. Model timeout or incorrect output. 3. Large dataset causes memory overflow. |

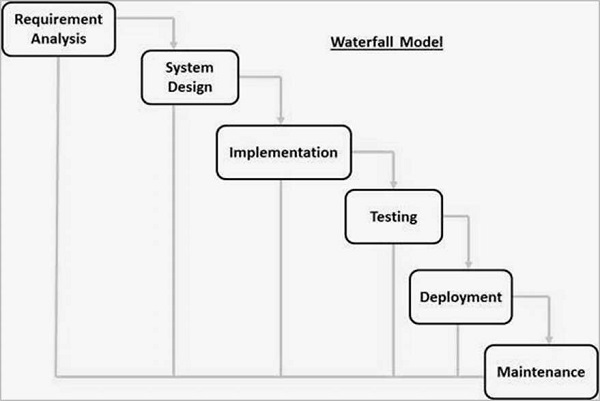
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Display Results | UC06 | 1. User views results on the web app. 2. Key metrics and influencing factors are displayed. 3. User saves or exports the results (optional). | Displays the results of the intrusion detection analysis in an intuitive interface, highlighting key metrics and traffic features influencing the classification. | User chooses to view detailed logs or summary report only. Allow exporting data in various formats (CSV, PDF). | Analysis must be completed successfully. Web app must be functional. | Results are displayed on the interface with options for saving/exporting. | Wasif Nawaz | 1. Server error during result display. 2. Incomplete analysis results. 3. Export function failure. |
| Save Results | UC07 | 1. User clicks the "Save" or "Export" button. 2. System validates file format. 3. Saves results in the chosen format. | This optional use case allows users to save or export the analysis results for offline review or documentation purposes. | Skip the save/export step if not required. Default to CSV format if no format is selected. | Results must be ready and displayed in the interface. | Results are saved successfully in the chosen file format. | Wasif Nawaz | 1. File write permission denied. 2. Insufficient storage space. 3. Export format not supported. |

## ****Adapted Methodology****

In this project, a hybrid approach known as the **VU Process Model** has been adopted. It effectively combines the structured nature of the **Waterfall Model** with the iterative flexibility of the **Spiral Model**, resulting in a development methodology that is both disciplined and adaptable.

#### ****Waterfall Model****

The **Waterfall Model** is a traditional, linear software development lifecycle (SDLC) methodology. It divides the entire process into discrete phases, each of which must be completed before the next one begins.



**Phases of the Waterfall Model**:

1. **Requirement Analysis** – Gathering and documenting functional and non-functional system requirements.
2. **System Design** – Architecting the system's components, data flow, and UI.
3. **Implementation** – Coding the designed components and modules.
4. **Testing** – Verifying and validating the developed system.
5. **Deployment** – Delivering the final product to the end users.
6. **Maintenance** – Addressing issues and applying updates post-deployment.

**Advantages**:

* Clear structure and documentation.
* Easy to manage and understand.
* Well-suited for projects with stable requirements.

**Limitations**:

* Not flexible for changes after development starts.
* Late discovery of issues (e.g., at testing or deployment phase).
* Not ideal for complex or research-intensive systems where requirements evolve.

**Relevance to This Project**:  
The Waterfall approach supports the **early planning** of your IDS project—helping to define requirements, design architecture, and structure milestones. It’s especially useful for organizing documentation such as SRS, design documents, and test plans.

#### ****Spiral Model****

The **Spiral Model** is a risk-driven software development process model that combines the iterative nature of prototyping with the controlled aspects of the Waterfall model. It is especially suited for large, complex, and evolving projects.

**Phases in Each Spiral Cycle**:

1. **Objective Setting** – Define goals, features, and alternatives.
2. **Risk Assessment** – Identify, analyze, and mitigate risks.
3. **Engineering and Prototyping** – Build the product incrementally with feedback loops.
4. **Evaluation and Planning** – Review outcomes and plan the next iteration.

**Advantages**:

* Encourages early and continuous risk management.
* Highly flexible and adaptive to changes.
* Supports incremental releases and evolving requirements.
* Strong focus on validation and stakeholder feedback.

**Limitations**:

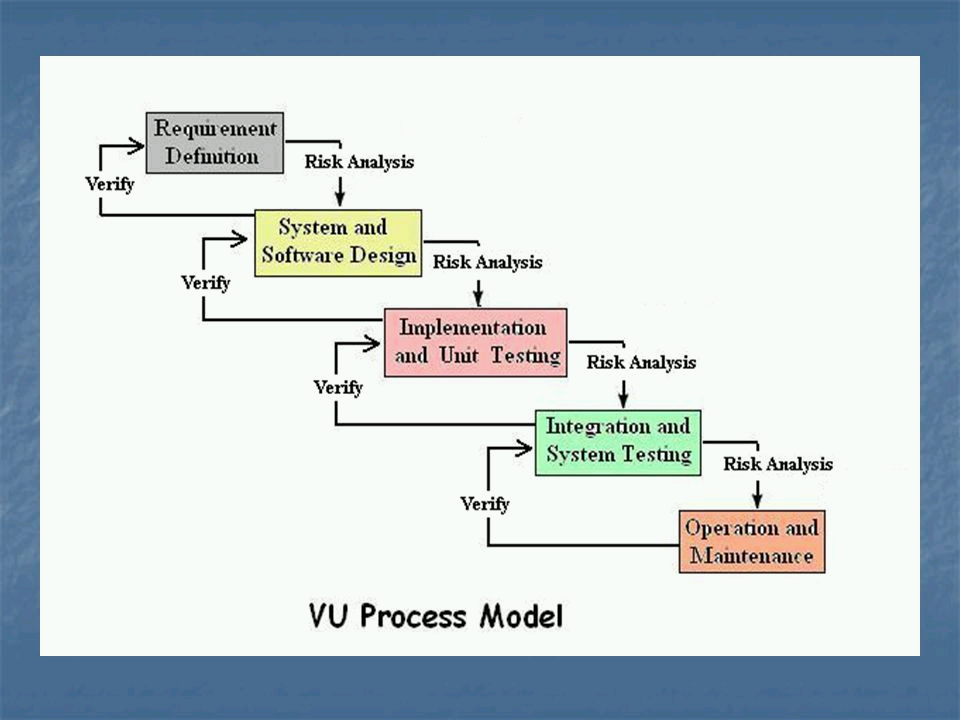
* Can be expensive and complex to manage.
* May lead to extended development cycles if not well-controlled.
* Not suitable for small or tightly time-boxed projects.



**Relevance to This Project**:  
The Spiral Model enables **iterative model training**, continuous integration of new techniques, and adaptive fine-tuning—crucial for machine learning components in your IDS. It helps incorporate regular testing, evaluation, and risk mitigation, particularly when working with evolving datasets and tuning multiple ensemble models.

#### ****VU Process Model (Hybrid Approach)****

The **VU Process Model** is a hybrid methodology recommended by the **Virtual University of Pakistan**, which integrates the Waterfall and Spiral models. It is specifically designed to align with both **academic structure** and **practical software development** requirements.



**Model Structure**:

* **Phase 1: Requirement Analysis** (Waterfall)  
  Document all system needs before design or coding begins.
* **Phase 2: System Design** (Waterfall)  
  Create architecture diagrams, data flow models, and GUI mockups.
* **Phase 3: Development & Testing** (Spiral)  
  Iteratively build the system in parts. Integrate ML models, perform testing, and refine based on outcomes.
* **Phase 4: Deployment and Evaluation** (Waterfall + Spiral)  
  Deliver final system. Gather feedback and apply final adjustments.

**Advantages**:

* Combines the predictability of Waterfall with the adaptability of Spiral.
* Provides structured milestones for academic tracking (e.g., SRS, Design Doc).
* Supports iterative model development and user testing cycles.
* Suitable for mid-scale projects like FYPs involving ML and web interfaces.

**Why It Was Chosen**:

* Offers a balance between structure and flexibility.
* Fits academic project deliverables and timelines.
* Allows for incremental development of ML models.
* Provides early planning and clear documentation.
* Reduces risk through iterative refinement and early testing.

### ****Reasons for Chosen Methodology****

The **VU Process Model** was selected for the following reasons:

1. **Balanced Approach**: It provides a structured framework while supporting flexibility for iterative model improvement and GUI enhancements.
2. **Academic Alignment**: Matches the required phases and document submissions (e.g., SRS, design document, mid-evaluation, final report).
3. **Risk Management**: Allows risk identification and mitigation during the development and testing of ML models and web interfaces.
4. **Efficient Planning**: Supports early planning of architecture, design, and functional modules.
5. **Iterative Enhancements**: Facilitates gradual improvement of the system, especially important for data preprocessing, training, and evaluating ensemble models like TabNet, CatBoost, and LightGBM.
6. **Stakeholder Visibility:** Encourages progress tracking, feedback integration, and clear communication between the team and supervisor.

**Work Plan (Use MS Project to create Schedule/Work Plan):**

