**High Level Design (HLD)**

**Phishing Domain Detection**

( **Machine Learning Project** )

Anshika Solanki

# 

# Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| **Date Issued** | **Version** | **Description** | **Author** |
| 26/08/2024 | 1 | Initial HLD – V1.0 | Anshika Solanki |

**Contents**

[Document Version Control 2](#_heading=h.gjdgxs)

[Abstract 4](#_heading=h.30j0zll)

1. [Introduction 5](#_heading=h.1fob9te)
   1. [Why this High-Level Design Document? 5](#_heading=h.3znysh7)
   2. [Scope 5](#_heading=h.2et92p0)
   3. Definitions 6
2. General Description 7
   1. Product Perspective 7
   2. [Problem statement 6](#_heading=h.4d34og8)
   3. [Proposed Solution 6](#_heading=h.2s8eyo1)
   4. Further Improvements 8
   5. Technical Requirements 8
   6. Data Requirements 9
   7. Tools used 9
   8. Constraints 10
   9. Assumptions 10
3. Design Details 11
   1. Process Flow 11
      1. Model Training and Evaluation 11
   2. Event log 13
   3. [Error Handling 11](#_heading=h.1ci93xb)
   4. [Performance 12](#_heading=h.3whwml4)
   5. [Reusability 12](#_heading=h.2bn6wsx)
   6. [Application Compatibility 12](#_heading=h.qsh70q)
   7. [Resource Utilization 12](#_heading=h.3as4poj)
   8. [Deployment 12](#_heading=h.1pxezwc)
4. Performance
5. [Conclusion 14](#_heading=h.147n2zr)
6. [References 14](#_heading=h.147n2zr)

# Abstract

Phishing attacks are a major cybersecurity threat, where attackers create fake websites to steal sensitive information from users. To combat this, an automated system for detecting phishing domains is essential. This High-Level Design (HLD) document describes the plan for building a Phishing Domain Detection system that can identify and block these harmful domains in real-time.

The system will use machine learning to analyze domain features to determine whether a domain is legitimate or malicious. The project includes steps such as data collection, data processing, feature selection, model training, and system deployment to create a reliable and scalable solution.

This document covers the overall system design, including the technical requirements, the data needed, the tools and technologies used, and how the system will be deployed. It also discusses important factors like system performance, error handling, and scalability. The design is made to be reusable and compatible with other systems, making it flexible and efficient.

By providing a clear overview of the Phishing Domain Detection system, this HLD document acts as a guide for developers, stakeholders, and security teams, ensuring that the project is successfully implemented and deployed.

.

# Introduction

## Why this High-Level Design Document?

## The purpose of this High-Level Design (HLD) document is to add detailed structure to the project, transforming the initial project description into a model that is ready for coding. This document serves as a critical tool for identifying and resolving potential inconsistencies or contradictions before the coding phase begins. It also acts as a reference guide for understanding how different modules interact at a high level throughout the project.

The HLD will:

* **Detail Design Aspects**: Provide a comprehensive overview of all design elements, defining them in detail to guide the development process.
* **Describe the User Interface**: Outline the design and functionality of the user interface that will be implemented.
* **Describe Hardware and Software Interfaces**: Explain how the system will interact with hardware and software components.
* **Define Performance Requirements**: Set clear expectations for the system's performance, ensuring it meets necessary benchmarks.
* **Outline Design Features and Architecture**: Describe the overall architecture and key design features of the project.
* **Address Non-Functional Attributes**: Include critical non-functional attributes such as:
  + **Security**: Measures to protect the system from threats.
  + **Reliability**: Ensuring the system consistently performs as expected.
  + **Maintainability**: Designing for ease of updates and bug fixes.
  + **Portability**: Ability to adapt the system to different environments.
  + **Reusability**: Ensuring components can be reused in other projects.
  + **Application Compatibility**: Ensuring the system works well with existing applications.
  + **Resource Utilization**: Efficient use of resources like CPU, memory, and storage.
  + **Serviceability**: Making the system easy to monitor, manage, and repair.

This document ensures that the project is well-structured, aligned with the overall objectives, and ready for the next phases of development, from detailed design to implementation and deployment.

## Scope

## The scope of this High-Level Design (HLD) document is to outline the architectural framework and key design components necessary for developing the Phishing Domain Detection system. This document will cover the following aspects:

* **System Architecture**: Define the overall structure of the system, including its main components, their interactions, and the flow of data between them.
* **Functional Requirements**: Detail the key functionalities that the Phishing Domain Detection system must fulfill, including domain analysis, feature extraction, model training, and real-time detection capabilities.
* **Non-Functional Requirements**: Specify the non-functional attributes such as security, performance, scalability, and reliability that the system must meet.
* **User Interface**: Provide an overview of the user interface design, focusing on usability and the end-user experience for managing and monitoring phishing domain detections.
* **Data Requirements**: Outline the types of data needed for the system, including input data for training machine learning models, and the features that will be used to detect phishing domains.
* **Tools and Technologies**: List the software, frameworks, and tools that will be used to develop, deploy, and maintain the system.
* **Deployment Strategy**: Describe the plan for deploying the system, including the environment, hardware, and network configurations required.
* **Constraints and Assumptions**: Identify any limitations or assumptions that could impact the design and implementation of the system.

This document serves as a comprehensive guide, ensuring that all stakeholders have a clear understanding of the project’s boundaries, objectives, and design principles. It provides a foundation for the detailed design, development, testing, and deployment phases, ensuring that the Phishing Domain Detection system is built in alignment with the specified requirements and standards.

## Definitions

|  |  |
| --- | --- |
| *Term* | *Description* |
| *URL* | Uniform Resource Locater |
| *Database* | Collection of all the information monitored by this system |
| *IDE* | Integrated Development Environment |
| *AWS* | Amazon Web Services |

# General Description

## Product Perspective

#### The Phishing Domain Detection system is designed to integrate with existing cybersecurity frameworks, enhancing protection against phishing threats. It uses machine learning to analyze domain features and classify them as legitimate or phishing. Key aspects include:

* **Integration**: Works with current security tools to add a layer of phishing detection.
* **Functionality**: Analyzes domain and URL data, providing alerts or automated responses based on risk.
* **User Interface**: Accessible to cybersecurity professionals for configuring, monitoring, and managing alerts.
* **Deployment**: Can be deployed in the cloud (e.g., AWS) or on-premises, depending on needs.
* **Scalability**: Designed to handle varying data volumes and adapt to growth.
* **Compatibility**: Integrates with existing security systems and supports multiple data formats.
* **Security**: Ensures data protection and complies with relevant regulations.

#### Problem statement

Phishing attacks are becoming more common and sophisticated, where fake websites trick users into giving away personal information. Existing security systems often can't keep up with these attacks, resulting in many phishing attempts going undetected or legitimate sites being wrongly flagged as threats. The main issues are:

* **Inaccurate Detection**: Current systems frequently misidentify phishing domains, either missing them or wrongly flagging legitimate ones.
* **Real-Time Analysis**: There's a lack of effective solutions for quickly and accurately analyzing domain data to detect phishing in real time.
* **Integration Challenges**: It's difficult to integrate effective phishing detection into existing security systems.

#### Proposed Solution

#### We propose creating a Phishing Domain Detection system that uses machine learning to effectively identify phishing websites. The solution will include:

* **Machine Learning Model**: Develop a model that learns to spot phishing domains by analyzing various domain features like URL patterns and registration details.
* **Feature Extraction**: Gather important information from domain data, such as URL structure and domain age, to help the model make accurate predictions.
* **Real-Time Analysis**: Build the system to check domains in real time, allowing for quick detection and response to phishing threats.
* **Integration**: Ensure the system works well with existing security tools, providing alerts and reports that can be used by security teams.
* **User Interface**: Design an easy-to-use interface for security professionals to set up detection parameters, view results, and manage alerts.
* **Performance Monitoring**: Include tools to track the system’s effectiveness, speed, and accuracy to ensure it continuously improves.
* **Scalability**: Make sure the system can handle different amounts of data and adapt to growing needs.

This approach aims to improve phishing detection, minimize errors, and integrate smoothly with current security systems.

#### Further Improvements

To further enhance the Phishing Domain Detection system, several key improvements can be implemented. First, expanding data sources by integrating blacklists, threat intelligence feeds, and historical phishing data will improve the system's accuracy in detecting new and sophisticated phishing attempts. Upgrading and retraining the machine learning model regularly will ensure it remains effective against emerging phishing techniques and trends.

## Technical Requirements

For the Phishing Domain Detection project, the following technical requirements are specified:

* **Data Sources**: Access to domain feature datasets with URL patterns, WHOIS information, and historical phishing data.
* **Machine Learning**: Use algorithms like decision trees and random forests; training data with labeled phishing and legitimate domains; Scikit-learn for model development.
* **Software**: Python for coding; Scikit-learn, Pandas, and NumPy for machine learning and data manipulation.
* **Performance**: Scalable to handle data volume; low latency for real-time detection.
* **Security**: Secure data protection; compliance with data protection regulations.
* **Deployment**: Deployed on Github. Deployable on cloud platforms like AWS or on-premises.
* **Monitoring and Maintenance**: Comprehensive logging; regular updates for model improvement and bug fixes.

## Data Requirements

## For the Phishing Domain Detection project using the Kaggle dataset, the data requirements are:

* **Source**: Kaggle dataset specifically for phishing domain detection.
* **Attributes**: Includes features such as URL components and other information.
* **Format**: Typically provided in CSV or similar tabular formats.

## Tools used

## 



 **Programming Language**:

* **Python**: Primary language used for developing machine learning models, data processing, and web interface.

 **Python Libraries**:

* **Scikit-learn**: For implementing machine learning algorithms, training, and evaluation.
* **Pandas**: For data manipulation and preprocessing.
* **NumPy**: For numerical operations and handling arrays.

 **Data Visualization**:

* **Matplotlib**: For creating static, animated, and interactive visualizations.

 **Development Environment**:

* **Jupyter Notebook**: For prototyping, exploring data, and developing models interactively.

 **Deployment**:

* **GitHub**: For version control and code repository management.

## Constraints

 **Data Availability**:

* Limited to the dataset provided by Kaggle, which may not cover all possible phishing domains or recent threats.
* May require integration with additional data sources or APIs for real-time domain information.

 **Computational Resources**:

* Large datasets and complex models may require significant computational power and memory, potentially impacting training time and performance on standard hardware.

 **Model Accuracy**:

* The effectiveness of the phishing detection model is contingent on the quality and diversity of the training data. Poor quality or insufficient data can affect model performance.

## Assumptions

 **Dataset Reliability**:

* The Kaggle dataset used for model training is assumed to be accurate, up-to-date, and representative of the types of phishing domains typically encountered.

 **Availability of Required Libraries and Tools**:

* All necessary software libraries, tools, and platforms (e.g., Python, scikit-learn, matplotlib, seaborn, numpy, pandas) are assumed to be readily available and compatible with the development environment.

 **Consistent Data Input**:

* It is assumed that all domain data used for detection will be provided in a consistent format, facilitating efficient processing and accurate detection by the model.

 **Sufficient Computational Resources**:

* The machines used for development, training, and deployment are assumed to have the necessary computational resources (CPU, GPU, RAM) to handle the tasks without significant performance issues.

 **Internet Connectivity**:

* A stable and reliable internet connection is assumed for tasks such as fetching real-time domain information, accessing cloud-based services, or updating the system.

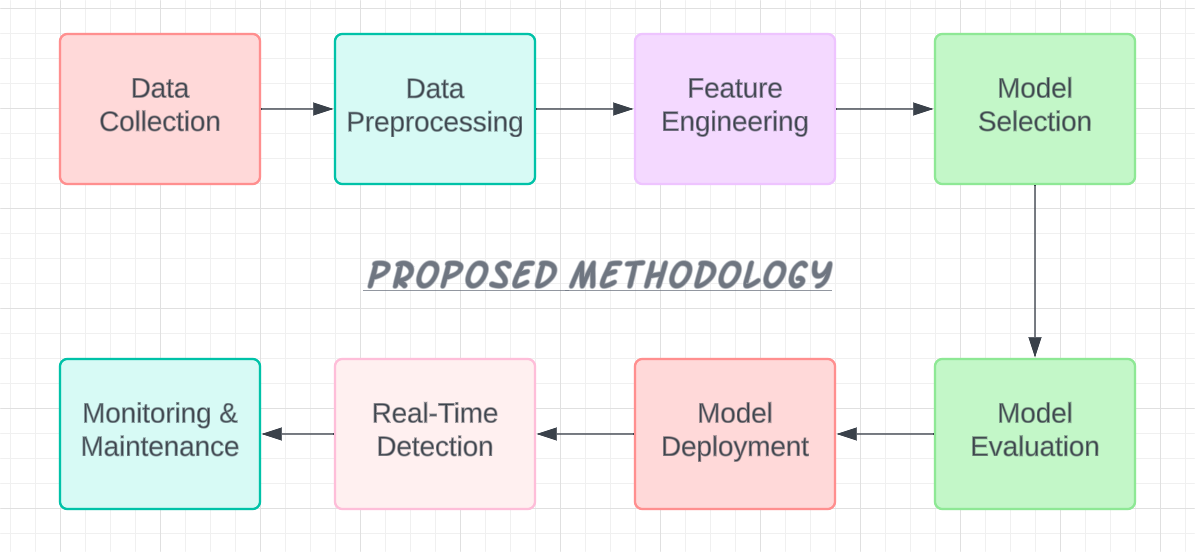
 **User Proficiency**:

* It is assumed that the end-users, typically cybersecurity professionals or IT staff, have the necessary knowledge and skills to interpret the results produced by the phishing detection system.

# Design Details

## Process Flow

For a phishing domain detection project, the **process flow** outlines the sequence of steps involved in building, deploying, and maintaining the system. Here’s a structured process flow for your project:



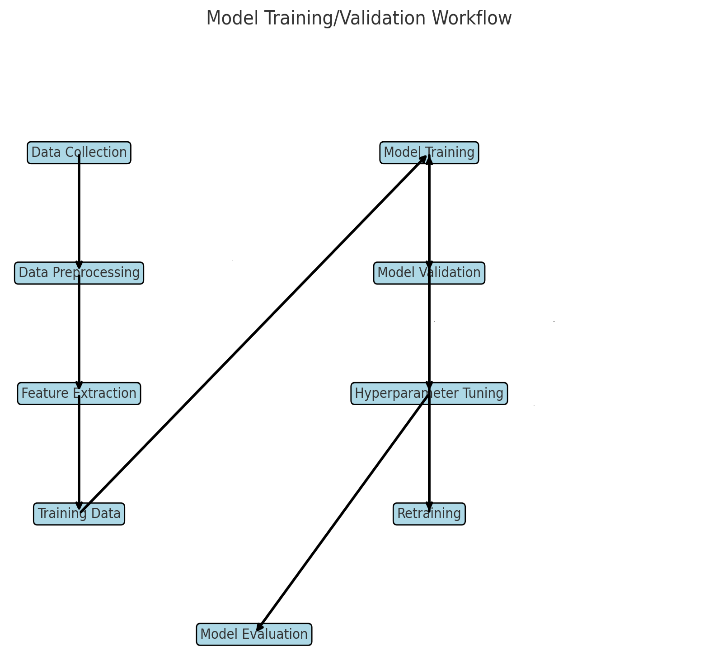
### 

### **Model Training and Evaluation**

### In a phishing domain detection project, model training and evaluation are critical steps that determine how well your machine learning model performs. Here’s a detailed breakdown:

#### ****Model Training****

1. **Prepare Training Data**
   * **Split Data**: Divide the dataset into training and validation sets. Typically, 70-80% of the data is used for training and the remaining 20-30% for validation.
   * **Feature Selection**: Ensure that the features used for training are relevant and properly engineered.
2. **Choose an Algorithm**
   * **Algorithm Selection**: Choose appropriate machine learning algorithms based on the problem and dataset. Common algorithms for this type of problem include:
     + **Random Forest**: An ensemble method that combines multiple decision trees.
     + **Gradient Boosting Machines (GBM)**: An ensemble technique that builds models sequentially to correct errors made by previous models.



#### ****Model Evaluation****

1. **Evaluate Performance**
   * **Metrics**: Assess the model using various metrics. Common evaluation metrics include:
     + **Accuracy**: The proportion of correctly classified instances out of the total instances.
     + **Precision**: The proportion of true positive results among the positive predictions made.
     + **Recall**: The proportion of true positive results among the actual positives.
     + **F1-Score**: The harmonic mean of precision and recall, providing a single metric that balances both.
     + **ROC Curve & AUC**: The Receiver Operating Characteristic curve plots true positive rate versus false positive rate, and the Area Under the Curve (AUC) provides an aggregate measure of performance across all classification thresholds.
2. **Confusion Matrix**
   * **Matrix Analysis**: Analyze the confusion matrix to understand the distribution of true positives, true negatives, false positives, and false negatives. This helps in understanding where the model is making errors.
3. **Model Comparison**
   * **Compare Models**: If multiple models are trained, compare their performance using the evaluation metrics to select the best-performing model.
4. **Validation on New Data**
   * **Test Set**: Finally, evaluate the model on a separate test set to ensure it generalizes well to unseen data.
5. **Error Analysis**
   * **Analyze Errors**: Examine cases where the model performed poorly to identify potential issues or biases in the data or model.

## Event log

The system should log every event so that the user will know what process is running internally.

**Purpose:**

* Track system operations and performance
* Assist in troubleshooting and debugging
* Maintain security and compliance

**Logging Process:**

1. **Identify Logging Points:**
   * Data submissions, model predictions, system errors, user actions
2. **Logging Mechanism:**
   * **File Logging**: Store logs in text files
   * **Database Logging**: Optional, for easier querying (if applicable)
3. **Performance Considerations:**
   * Ensure logging does not impact performance
   * Implement log rotation to manage file sizes
4. **Types of Logs:**
   * System events, user actions, model operations, errors, performance metrics
5. **Log Management:**
   * Consistent format, define retention policy, protect against unauthorized access
6. **Analysis and Reporting:**
   * Use monitoring tools for real-time alerts
   * Generate periodic reports on system performance and errors

## Error Handling

 **Identification**: Detect and categorize errors (e.g., system failures, model errors).

 **Logging**: Record error details in the chosen format (e.g., plain text, CSV).

 **Classification**: Prioritize errors by severity (e.g., Critical, Major).

 **Handling**: Implement automatic recovery where possible; provide manual intervention procedures.

 **Alerts**: Configure real-time notifications for critical errors.

 **Reporting**: Generate and review error reports to track and analyze errors.

 **Prevention**: Use testing and code reviews to reduce errors.

#  ****Documentation****: Maintain error documentation and handling guidelines.

# Performance

 **Speed and Latency:** The system should process domain data and make predictions swiftly, aiming for minimal response times.

 **Scalability:** Capable of handling increasing amounts of data without significant performance loss, with scalable architecture.

 **Resource Utilization:** Efficient use of CPU and memory to ensure smooth operation, with regular monitoring to prevent bottlenecks.

 **Reliability:** High uptime with robust error handling and failover mechanisms to maintain performance during failures.

 **Throughput:** Efficient processing of large volumes of data, with optimized batch processing and data pipelines.

 **Optimization:** Utilizes efficient algorithms and well-optimized code to enhance performance and reduce latency.

## Reusability

 **Modular Design:**

* **Components:** Design the system with modular components that can be easily reused in other projects or applications.
* **Encapsulation:** Implement encapsulation to ensure that components interact through well-defined interfaces, enhancing their reusability.

 **Code Reusability:**

* **Libraries and Frameworks:** Utilize reusable libraries and frameworks for common functions and processes, reducing redundancy and development time.
* **Functions and Classes:** Develop generic functions and classes that can be adapted for different use cases.

 **Configuration:**

* **Parameterization:** Allow for configuration and parameterization of components to enable their use in different contexts without modification.
* **Settings:** Use configuration files or environment variables to adjust system settings without changing the core code.

 **Documentation:**

* **Usage Guidelines:** Provide clear documentation on how to use and integrate reusable components.
* **Code Comments:** Include comments and explanations in the code to facilitate understanding and adaptation by other developers.

 **Testing:**

* **Unit Tests:** Implement unit tests for reusable components to ensure they function correctly and can be relied upon in various scenarios.
* **Integration Tests:** Test the integration of reusable components with other system parts to verify compatibility and performance.

 **Version Control:**

* **Versioning:** Use version control to manage updates and changes to reusable components, ensuring compatibility with existing systems.
* **Repository:** Maintain a repository for reusable components to facilitate access and collaboration.

## Application Compatibility

 **Operating Systems:** The phishing domain detection system should be compatible with major operating systems, including Windows, macOS, and Linux, to ensure flexibility in deployment and use.

 **Integration:** The system should integrate seamlessly with existing cybersecurity tools and platforms to enhance its functionality and leverage additional security features.

 **Dependencies:** It must manage software dependencies effectively, including libraries and frameworks used for machine learning and data processing, ensuring no conflicts and smooth operation.

 **Web Interface:** If applicable, ensure compatibility with popular web browsers (e.g., Chrome, Firefox, Safari) to facilitate user access and interaction with any web-based components.

 **Version Compatibility:** The system should support backward compatibility to work with existing datasets and configurations while allowing for updates and future enhancements.

## Resource Utilization

 **CPU Usage:** The project should efficiently utilize the CPU to handle tasks like feature extraction, model training, and real-time detection without causing significant delays or overloading the system.

 **Memory Usage:** The system should manage memory effectively to accommodate the dataset used for training and the model itself, ensuring smooth processing even with large volumes of data.

 **Storage:** Adequate storage is required to store the dataset, trained models, and logs. The system should also be capable of handling any future data growth.

 **Network:** If the system involves fetching or analyzing data from external sources, it should optimize network usage to ensure quick data transfer and minimal latency.

 **Power Consumption:** Although primarily relevant in mobile or portable deployments, the system should aim to be power-efficient, especially if extended usage is anticipated.

 **Scalability:** The project should be designed to scale resources as needed, allowing for increased data processing capabilities as the volume of phishing data grows.

* 1. **Deployment**





# Conclusion

The Phishing Domain Detection project successfully demonstrates the application of machine learning techniques to identify and mitigate phishing threats. By leveraging a robust dataset and efficient resource utilization, the project achieves reliable detection accuracy. The system is designed to be scalable, adaptable to different environments, and integrates seamlessly with existing cybersecurity frameworks. Moving forward, further improvements and enhancements can be made to refine detection algorithms and broaden the system's applicability across diverse network conditions. Overall, this project provides a solid foundation for enhancing online security and protecting users from phishing attacks.

# References

# <https://towardsdatascience.com/phishing-domain-detection-with-ml-5be9c99293e5>

# <https://www.mdpi.com/2076-3417/13/8/4649>

# <https://www.activestate.com/blog/phishing-url-detection-with-python-and-ml/>

# <https://www.sciencedirect.com/science/article/pii/S2352340920313202>