Software Modelling Project

On

Network Malware Scanner

SUBMITTED BY:
RUPESH KUMAR(20232732)
BRAJESH KUMAR(20232709)

SUBMITTED TO: Dr. Neha

B.Voc. (Software Devlopment)

RAMANUJAN COLLEGE,
UNIVERSITY
OF
DELHI

Project : Network Malware Scanner

Problem Statement: Network Malware Scanner

In today's hyper-connected digital environment, computer networks are continuously exposed to a growing variety of cyber threats. Malware — which includes viruses, worms, Trojans, ransomware, spyware, and other malicious software — is one of the most prevalent and dangerous forms of cyber threats. These malicious entities can infiltrate networks, disrupt operations, steal sensitive data, and cause extensive financial and reputational damage to individuals, organizations, and governments.

Traditional anti-malware solutions, often endpoint-based, may not provide adequate visibility or protection at the network level. Moreover, with the increasing complexity and volume of data traversing networks, identifying malicious traffic using manual or signature-based detection methods is no longer sufficient. These methods fail to detect zero-day attacks, encrypted malware communications, and polymorphic malware that frequently change their code signatures.

Thus, there is a growing need for an intelligent, real-time Network Malware Scanner that can detect and mitigate malware threats by analyzing network traffic.

The aim of this project is to design and implement a Network Malware Scanner that operates at the network level, capable of:

- Capturing and analyzing network packets in real time.
- Identifying suspicious patterns or payloads using signature-based and anomaly-based detection techniques.
- Classifying traffic as malicious using advanced methods such as machine learning.
- Generating alerts and reports for detected threats, allowing for timely response and mitigation.

This system should address the following challenges:

- False positives and negatives in malware detection.
- Efficient processing and classification of packets with minimal system overhead.

By developing this system, the project contributes to enhancing network security by proactively identifying and responding to malware threats before they cause damage.

Process Model: Classical Waterfall Model

The Classical Waterfall Model is a linear and sequential software development approach where each phase must be completed before the next begins. It is best suited for projects with well-defined requirements and minimal expected changes. For the **Network Malware Scanner**, the Waterfall model phases can be detailed as follows:

1. Requirements Analysis and Specification

- **Objective**: Clearly define the functional and non-functional requirements of the malware scanner.
- **Activities**:
  - Identify user needs: Real-time scanning, traffic logging, malware detection alerts.
  - Define system constraints: Performance limits, types of networks supported.
  - Document requirements: Use Software Requirements Specification (SRS).
- **Output**: SRS document.

2. System and Software Design

- **Objective**: Design the architecture of the system based on the requirements.
- **Activities**:
  - Define system architecture: Components such as Packet Sniffer, Detection Engine, and Reporting Module.
  - Design data flow: How network packets are processed and analyzed.
  - Select technologies: Python/Java for development, Wireshark/tshark for packet capture, Scikit-learn/TensorFlow for ML-based detection.
- **Output**: System Design Document (SDD), Data Flow Diagrams (DFDs), and architectural diagrams.

3. Implementation (Coding)

- **Objective**: Develop the individual modules and integrate them.
- **Activities**:
  - Develop modules: Packet Capturer, Signature Matching, Report Generator, Alert Generator.

    - Integration: Combine modules into a single, functional system.
    - Follow coding standards and practices.
- **Output**: Source code, build files, and internal documentation.

4. Integration and Testing

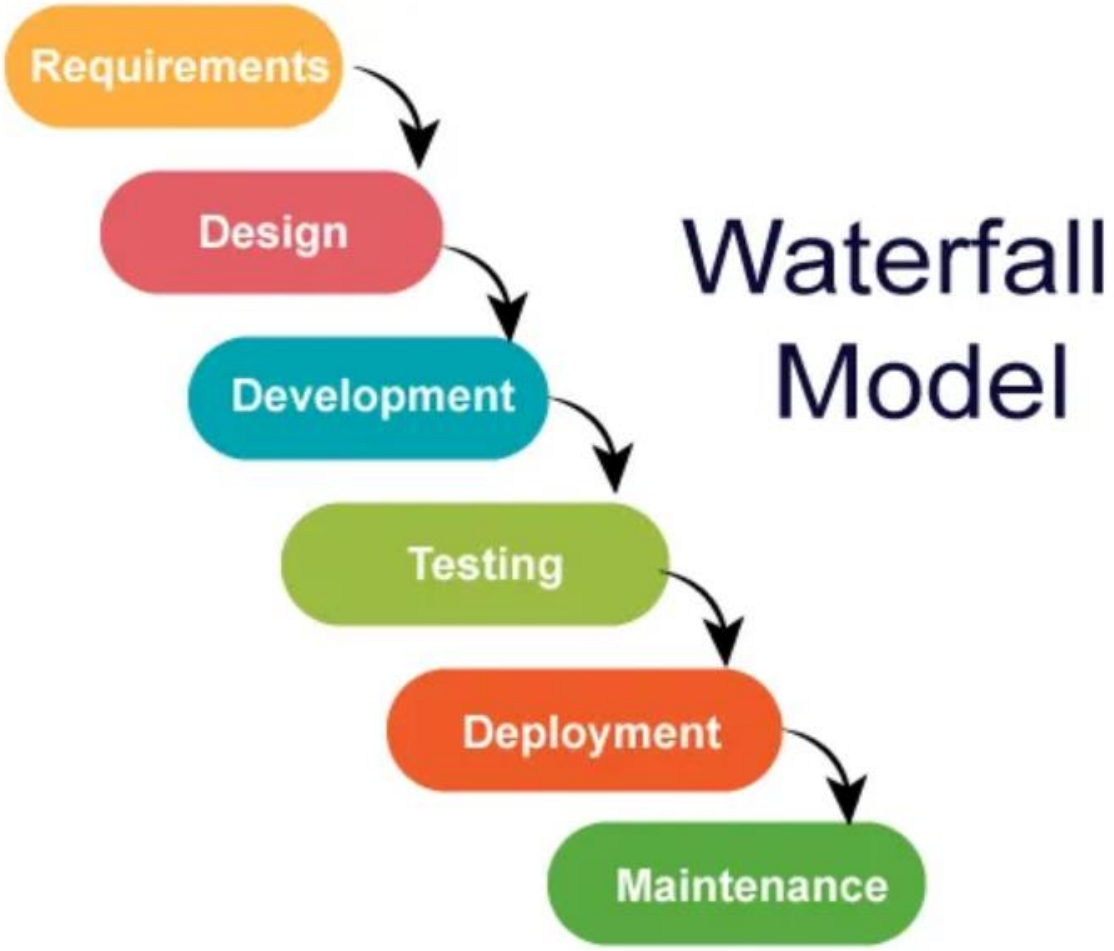
- **Objective**: Test the integrated system to find and fix defects.
- **Activities**:
    - Unit testing: Verify each module independently.
    - Integration testing: Ensure modules interact correctly.
    - System testing: Test the full malware scanner on various network scenarios.
    - Use test cases for known malware and normal traffic.
- **Output**: Test Reports, Bug Reports, and Verified Software.

5. Deployment

- **Objective**: Deploy the malware scanner in a target environment.
- **Activities**:
    - Install the system on a real or simulated network.
    - Monitor performance and detect any deployment issues.
- **Output**: Deployed System.

6. Maintenance

- **Objective**: Provide ongoing support and updates.
- **Activities**:
    - Fix post-deployment bugs.
    - Update malware definitions and ML models.
    - Add features as needed based on user feedback.
- **Output**: Maintenance Reports, Updated Software Versions.


Requirements
Design
Development
Testing
Deployment
Maintenance
Waterfall Model

Software Requirement Specification(SRS)

1. Introduction

1.1 Purpose

The **Network Malware Scanner** is a security-based application designed to detect and analyze malware within a device over the network. It aims to enhance network security by identifying malicious traffic and providing actionable insights to reduce threats.

1.2 Intended Audience

The intended audience for this document includes:

- **Developers** – For implementation guidance.
- **Security Analysts** – To understand the security scope and functionality.
- **Project Managers** – To oversee project requirements and development progress.
- **End Users** – To people who actually uses the software.

1.3 Scope

The **Network Malware Scanner** is a security-focused application that will:

- Monitor network-connected devices to detect malware threats.
- Use signature-based detection for known malware threats.
- Employ behavioral analysis to identify anomalies and zero-day attacks.
- Provide a comprehensive dashboard displaying scan results and threat reports.
- Allow integration with external threat intelligence feeds for up-to-date malware detection.

By delivering these functionalities, the application aims to enhance cybersecurity and prevent malware intrusions within a network environment.

1.4 Definitions

- **Malware**: Malicious software such as viruses, worms, Trojans, ransomware, and spyware.
- **Signature-based detection**: A method of detecting known malware by comparing against a database of malware signatures.
- **Behavioral analysis**: A technique that detects malware based on unusual behavior patterns.

- **Threat intelligence feeds**: Real-time information sources that provide data on emerging threats.

1.5 References

- IEEE SRS Standard Guidelines
- OWASP Security Best Practices
- NIST Cybersecurity Framework

2. Overall Description

2.1 User Interfaces

The system will provide the following user interfaces:

- **Dashboard**: Users can perform scans, view reports, and configure settings.
- **Admin panel**: For managing users, logs, and system configurations.
- **API access**: For integration with external security tools.

2.2 System Interfaces

The application will integrate with:

- **Network monitoring tools** for real-time scanning.
- **Threat intelligence APIs** for updated malware signatures.
- **Cloud storage** for log and report management.

2.3 Constraints, Assumptions, and Dependencies

- The system must ensure real-time detection with minimal network latency.
- Data privacy must be maintained by encrypting scanned data.
- The application must be compatible with major operating systems and browsers.
- Requires continuous updates for malware signature databases.
- Assumes network access permissions are granted for full scanning.

2.4 User Characteristics

The users of this application include:

- **System Administrators** – Manage scans and threat responses.
- **Cybersecurity Analysts** – Investigate detected threats.
- **General Users** – Perform network scans to check for malware.

3. System Features and Requirements

3.1 Functional Requirements

- **Network scanning** – Detects all the network requests on devices and scans them for malware.
- **Malware detection** – Identifies threats using signature-based and behavioral techniques.
- **Threat reporting** – Generates reports with severity levels and mitigation steps.
- **User authentication** – Ensures secure access to the application.
- **Log management** – Stores and tracks previous scans and threats.

3.2 Use Cases

Use Case 1: Network Malware Scan

Actors: User, Application software
Preconditions: User is authenticated and has network access.
Steps:

1. User initiates a scan.
2. Software scans devices over the network.
3. Malware is detected and analyzed.
4. A report is generated and displayed.

Use Case 2: Threat Report Generation

Actors: Application software, Security Analyst
Steps:

1. Application software detects malware.
2. Generates a threat report.
3. Security Analyst reviews and takes action.

3.3 External Interface Requirements

- **REST API** for third-party security tool integration.
- **Database connection** for storing scan logs.
- **User authentication system** (OAuth, LDAP, or custom authentication).

3.4 Logical Database Requirement

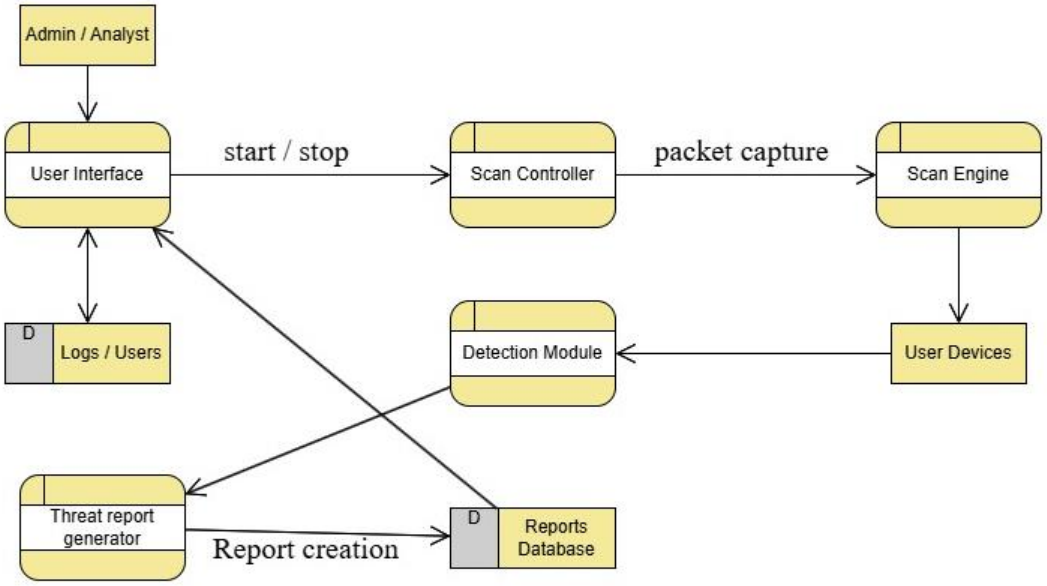
The system will maintain:

- **User Data**: Authentication credentials and access permissions.
- **Scan Logs**: History of scanned devices and detected threats.
- **Threat Database**: Malware signatures and behavioral patterns.

3.5 Nonfunctional Requirements

- **Performance**: Real-time scanning with minimal impact on network performance.
- **Security**: Encrypted communications and role-based access control.
- **Scalability**: Ability to scan networks with multiple devices efficiently.
- **Availability**: 99.9% uptime with failover support.

Data Flow Diagram:



Project Management:

Function Type	**Function Type Count**	**Complexity**	**Functional Point**
Input Function (User details, System Details and permissions)	2	3	6
Internal UI (Dashboard , Admin Panel, Security Updates)	3	5	15
API Calls	3	4	12
Scanning all type of requests and responses	1	5	5
System Scan	1	5	5
Database Connection and store generated reports	1	3	3
Login Authentication System	1	4	4
			50 (Total FP)

Efforts (For Manpower)

=> Total FP * Productivity Factor

50 * 2.5 = 125.0 , where 2.5 is the standard Productivity Factor

SCHEDULE TIME LINE CHART

Required Phases	Max Required Time
Phase 1:Requirement Analysis	3-4 weeks
Phase 2:Design	2-3 weeks
Phase 3:Implementation and Unit Testing	5-6 weeks
Phase 4:Integration and System Testing	2-3 weeks
Phase 5:Deployment	1-2 weeks
Phase 6:Maintenance	1 week

Risk Table

List of Risk	Category	Probability	Impact
Resources Availability	Catastrophic	75-100%	Project Time Delays, Data Storage Issues
API end points are not working properly	Marginal	25-50%	Security Related Updates Issues
Overloads on server	Marginal	25-50%	User Authenticating Issues
Deployment Delays	Marginal	25-50%	Penalty to institutes
Technology Issues	Negligible	0-25%	Project Time Delays

ARCHITECTURAL DESIGN :

LAYERED ARCHITECTURE:

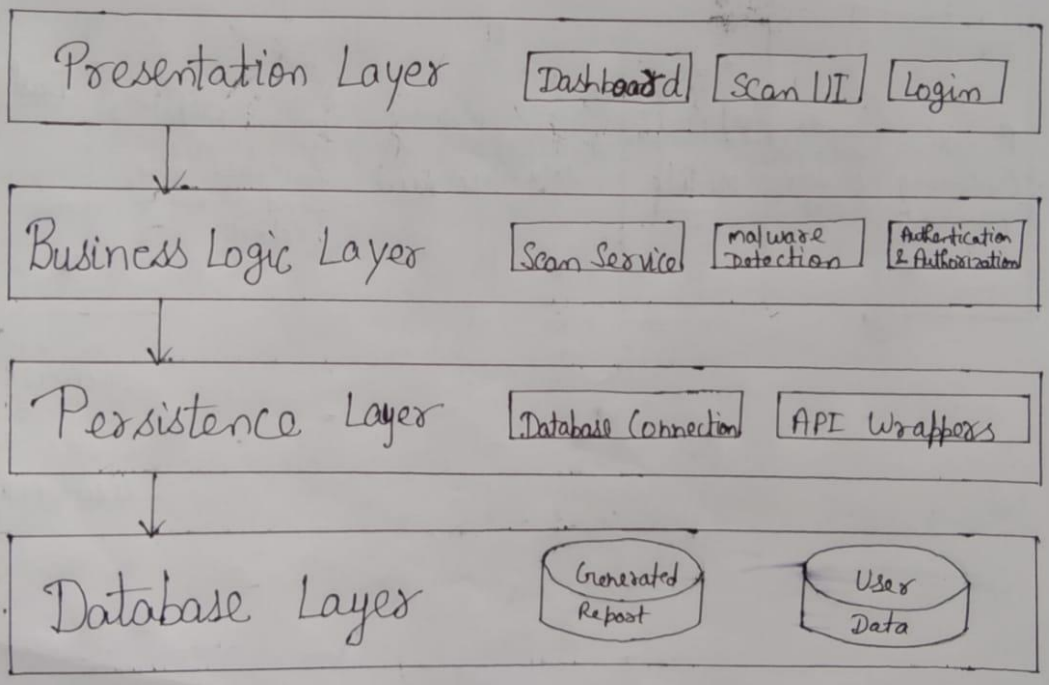
1. Presentation Layer (UI) (Used for show):
    - Dashboard
    - Admin Panel
    - Scan Initiation UI
    - Login Page

2. Business Logic Layer (Application Layer) (Used for Decide):
    - Scan Service
    - Malware Detection Component
    - Threat Report Generation
    - Authentication & Authorization module

3. Persistence Layer (Used for Fetch or save):
    - Database interaction (User data, Scan logs, threats)
    - API wrappers for threat intelligence feeds
    - Cloud Storage connection handlers

4. Database Layer (Storage):
    - File Storage for logs and reports
    - User data

Architectural Design



Data Design:

User Data :

Field	Type	Description
Username	String	Username
Email	String	Unique Email
Password	String	Unique Password

System Configuration:

Field	Type	Description
Device ID	UUID / String	Unique ID of the device
IP Address	String (IPv4/6)	IP Address of the device
Hostname	String	Hostname of the device
MAC Address	String	MAC Address
Operating System	String	OS name/version

Malware Signature

Field	Type	Description
SignatureID	UUID	Unique ID for the malware signature
Name	String	Malware name
Description	Text	What the malware does
Pattern	String/Text	Pattern to match in traffic
SeverityLevel	Enum (Negligible/Marginal /Critical/Catastrophic)	Threat level

Scanned Result :

Field	Type	Description
ScanID	UUID	Unique ID for the scan
Detected	Boolean	Malware found? (Yes/No)
Detection Time	Timestamp	When malware was detected
Report	File	Generated Report

Component Level Design :

1. Packet Capture Component

- Uses libraries like libpcap, WinPcap, or Pcap4J.
- Can sniff live traffic or open pcap files.
- Sends raw packets to the Preprocessing Module.

2. Preprocessing Module

- Extracts headers and payloads (IP/TCP/UDP layers).
- Filters out irrelevant traffic (like trusted IPs).

3. Signature Database

- Contains known malware fingerprints.
- Organized into:
  - Static Signatures (known malware byte patterns)
  - Behavioral Signatures (e.g., "scans multiple ports rapidly")
- Stored in an efficient format (maybe **SQLite**, or **NoSQL** like MongoDB).

4. Malware Detection Engine:

- The core "brain" of the scanner.
- Compares packet data against the Signature Database.
- Two modes:
  - Static Matching (signature pattern matching)
  - Heuristic/Anomaly Detection (e.g., port scan detection, protocol misuse)

Techniques:

- Regex pattern matching
- Payload fingerprinting
- Machine Learning module (optional, for anomaly detection)

5. Alert Manager

- Triggered when malware is detected.
- Assigns severity based on Signature Database info.
- Sends alerts to:
  - Dashboard
  - Email

5. Logging & Reporting Module

- Logs all captures, scans, detections, and errors.
- Generates summary report.
- Supports formats like CSV, JSON, PDF.

6. Admin Dashboard

- **UI for:**
  - User Profile details
  - Viewing real-time scan status
  - Viewing alerts
- **Database for:**

  Store User details

Example Tech Stack:

- Frontend: React Native,
- Backend: Flask, Fast API, Node.js

7. Security related Updates

Regularly provides the security related updates to the user using different API.

Component Level Design for "N/W Malware Scanner"