A PROJECT REPORT

ON

DATA LEAKAGE PREVENTION SYSTEM (DLPS)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

ABSTRACT

In the modern digital landscape, data security is paramount. The **Data Leakage Prevention System (DLPS)** is a comprehensive enterprise-grade web application designed to detect and prevent potential data breaches within an organization. The system monitors data transfer channels, specifically file uploads, to identify sensitive information such as Personally Identifiable Information (PII) and confidential keywords before it leaves the secure environment. This project implements a robust Role-Based Access Control (RBAC) model, ensuring a strict separation between User and Administrator privileges. Key features include a real-time scanning engine utilizing Regular Expressions (Regex) and pattern matching, an interactive dashboard for analytics, dynamic policy management, and an immutable audit trail for forensic analysis. The system is built using PHP for server-side logic, MySQL for the database, and a responsive frontend using HTML5, CSS3, and JavaScript, ensuring a secure and user-friendly experience for all stakeholders.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Overview

The Data Leakage Prevention System (DLPS) is a specialized security tool developed to address the growing threat of internal data theft and accidental data exposure. By implementing a gateway that scans all outgoing files, the system ensures compliance with organizational data security policies.

1.2 Problem Statement

Traditional security measures often focus on external threats (firewalls, antivirus). However, a significant percentage of data breaches originate internally—either through malicious intent or employee negligence. Organizations lack affordable, easy-to-configure tools to monitor and block sensitive data transmission in real-time.

1.3 Objectives

- To develop a web-based platform for secure file scanning.
- To implement strict Role-Based Access Control (User vs Admin).
- To detect sensitive patterns like Credit Card numbers, SSNs, and confidential keywords.
- To maintain a secure, immutable log of all attempts for auditing.

1.4 Scope of the Project

The project covers the development of a web portal where users can upload files for verification. Administrators can define what constitutes "sensitive data" by creating dynamic policies. The scope extends to reporting and analytics but excludes network-level packet sniffing (currently focused on application-level file upload scanning).

CHAPTER 2: SYSTEM ANALYSIS

2.1 Existing System

Currently, many organizations rely on manual auditing or trust-based policies, which are prone to human error. Commercial DLP solutions exist but are often prohibitively expensive and complex to deploy for smaller setups.

2.2 Proposed System

The proposed DLPS is a lightweight, effective, and user-friendly web application. It automates the detection process, provides instant feedback to users, and alerts administrators. It centralizes policy management, making it easy to update security rules across the organization instantly.

2.3 Feasibility Study

- **Technical Feasibility**: The system manages file I/O and text processing, which is well-supported by PHP. MySQL handles the relational data efficiently.
- **Operational Feasibility**: The UI is designed to be intuitive, requiring minimal training.
- **Economic Feasibility**: Built on open-source technologies (XAMPP stack), keeping costs at zero.

CHAPTER 3: SYSTEM REQUIREMENT SPECIFICATION

3.1 Hardware Requirements

- Processor: Intel Core i3 or higher
- RAM: 4 GB minimum
- Hard Disk: 10 GB free space

3.2 Software Requirements

- Operating System: Windows / Linux
- Server: Apache Web Server (via XAMPP)
- Database: MySQL
- Language: PHP 8.0+
- Browser: Chrome, Firefox, or Edge

3.3 Functional Requirements

- **Authentication**: Secure login for Users and Admins.
- **File Upload**: Support for text-based files.
- **Scanning**: Regex-based pattern matching.
- **Reporting**: Dashboard visualization of safe vs. blocked files.
- **Policy Management**: CRUD operations for DLP policies.

3.4 Non-Functional Requirements

- **Security**: Passwords hashed using Bcrypt. Session hijacking protection.
- **Performance**: Scan results generated in under 1 second for standard files.
- **Reliability**: 99.9% uptime for the local server environment.

CHAPTER 4: SYSTEM DESIGN

4.1 System Architecture

The system follows the **Model-View-Controller (MVC)** architectural pattern principles (adapted for core PHP).

- **Presentation Layer**: HTML/CSS/JS Frontend.
- **Logic Layer**: PHP scripts handling scanning and routing.
- **Data Layer**: MySQL storage for logs and user data.

4.2 Data Flow

1. User logs in -> Uploads File.
2. Server receives file -> Reads content.
3. Content passed to Scanning Engine -> Checked against `dlp_policies`.
4. Result (Safe/Risk) -> Stored in Database (`file_scans`).
5. Response sent to User Dashboard -> Admin Audit Log updated.

4.3 Database Design (Schema)

- `users` : Stores user credentials and profile info.
- `admins` : Specialized table for high-privilege accounts.
- `file_scans` : Records of every file processed (filename, risk level, hash).
- `dlp_policies` : Rules defining regex patterns and severity.
- `audit_logs` : Chronological record of system activities.

4.4 Module Description

- **Admin Module**: Dashboard, User Management, Policy Management, Reports.
- **User Module**: Dashboard, Upload Interface, History, Profile.
- **Auth Module**: Session handling, Registration, Login, Logout.

CHAPTER 5: IMPLEMENTATION

5.1 Technology Stack

- **Languages**: PHP, JavaScript (ES6), SQL.
- **Libraries**: Chart.js (Visualization), FontAwesome (Icons).
- **Styling**: Custom CSS with Dark Theme aesthetics.

5.2 Key Algorithms (Scanning Logic)

The core engine iterates through active policies. For each policy, it applies the associated Regular Expression to the file content.

```
foreach ($policies as $policy) {
    if (preg_match($policy['pattern'], $fileContent)) {
        $riskLevel = $policy['severity'];
        // Flag file as risk
    }
}
```

5.3 Security Implementation

- **SQL Injection**: Prevented using PDO Prepared Statements.
- **XSS**: Input sanitization on all output fields.
- **Access Control**: Session-based checks `$_SESSION['role']` on every page load.

CHAPTER 6: TESTING

6.1 Testing Methodologies

- **Unit Testing**: Testing individual functions (e.g., login validation).
- **Integration Testing**: Verifying the flow from Upload -> Scan -> Database -> Dashboard.
- **Security Testing**: Attempting unauthorized access to Admin pages (successfully blocked).

6.2 Test Cases

ID	Test Description	Expected Result	Actual Result	Status
TC1	Admin Login	Access Dashboard	Accessed	Pass
TC2	User Access Admin Page	Access Denied	Redirected to Login	Pass
TC3	Upload SSN File	Block/Critical Alert	Blocked	Pass
TC4	Upload Safe Text	Safe Status	Safe Status	Pass

CHAPTER 7: CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

The DLPS project successfully demonstrates a functional security tool capable of mitigating data leakage risks. It meets all primary objectives, providing a secure, user-friendly, and effective solution for file monitoring.

7.2 Future Enhancements

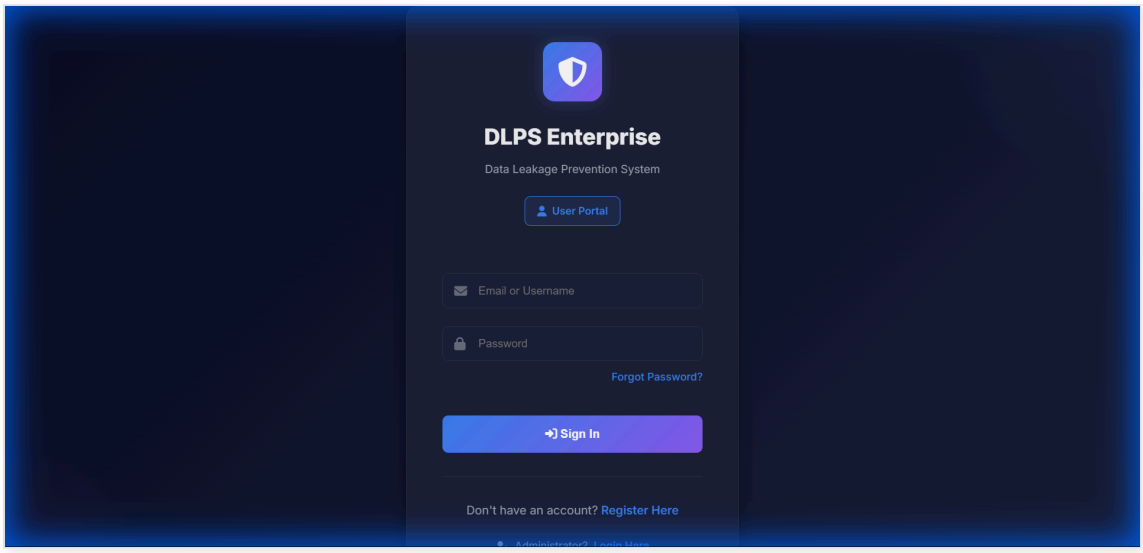
- Support for PDF and Image OCR scanning.
- Integration with email servers for outbound attachment scanning.
- AI/ML-based anomaly detection for unknown threat patterns.

CHAPTER 8: PROJECT SCREENSHOTS

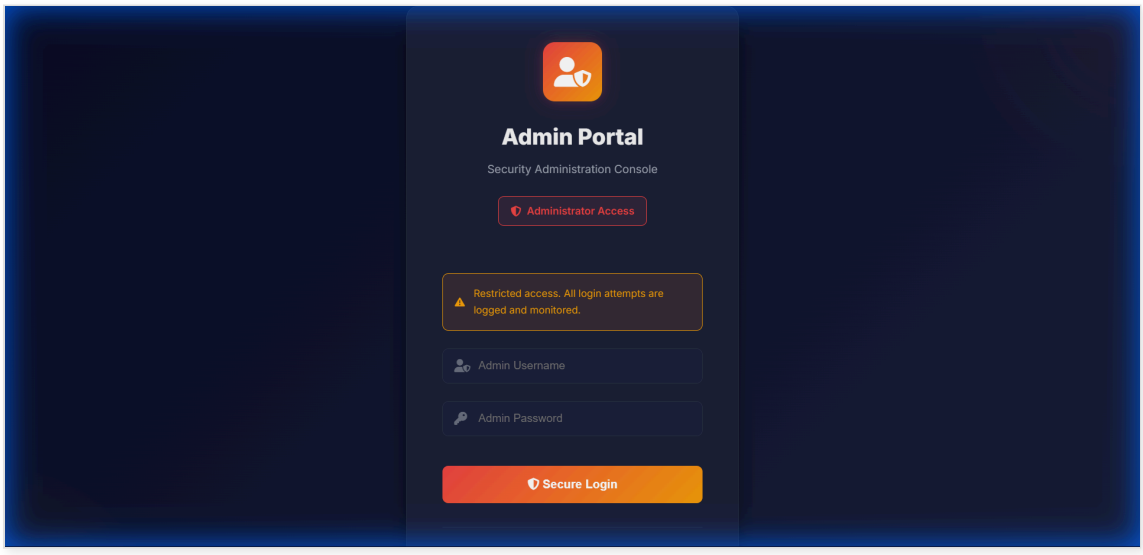
8.1 Authentication

Login Page

To maintain security, the user and admin login pages are separate.



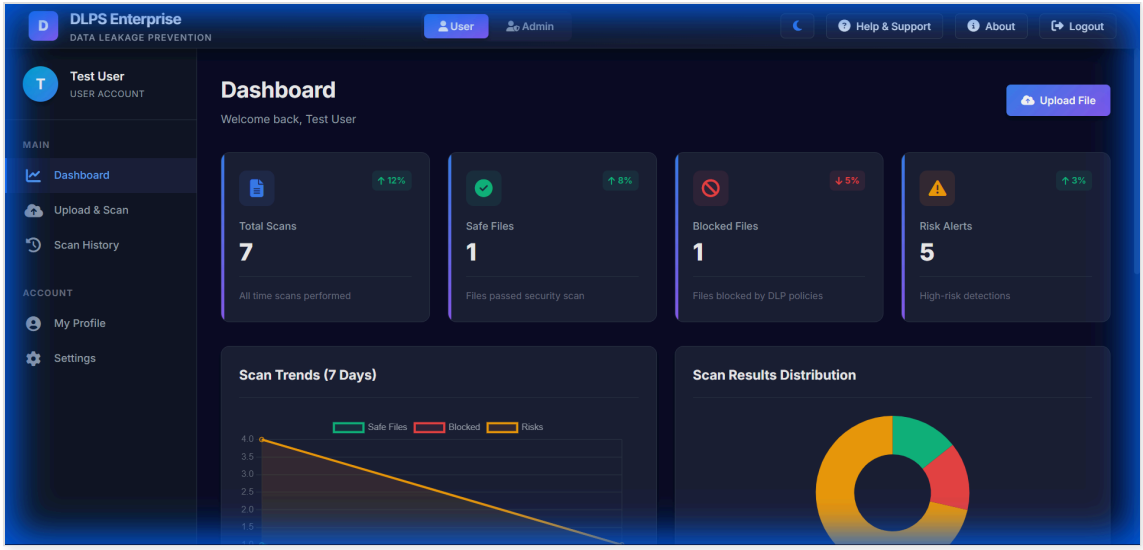
Admin Login



8.2 User Portal

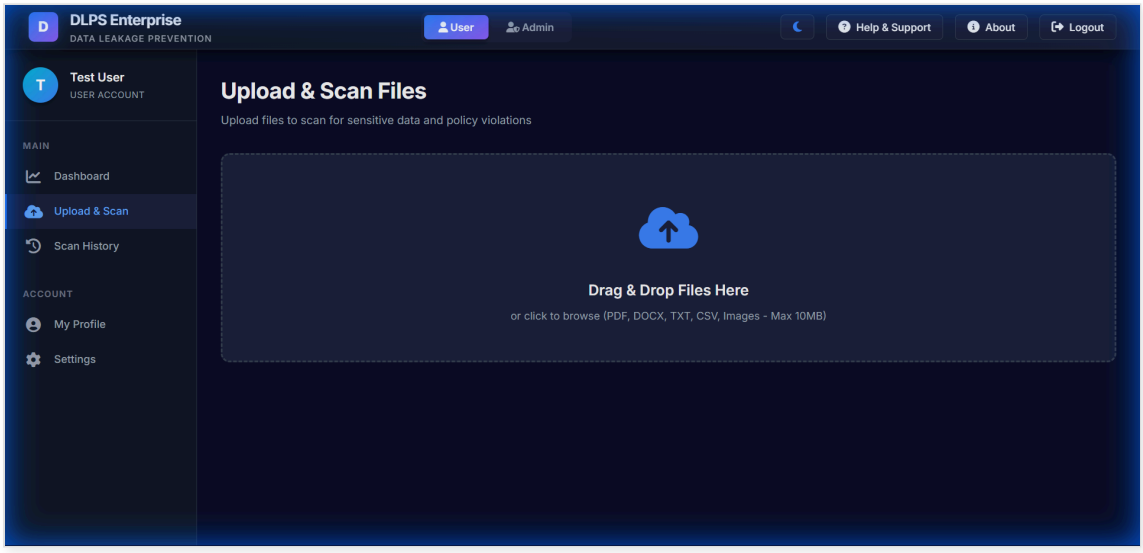
User Dashboard

The user dashboard provides a quick overview of recent activities and scan status.



File Upload Interface

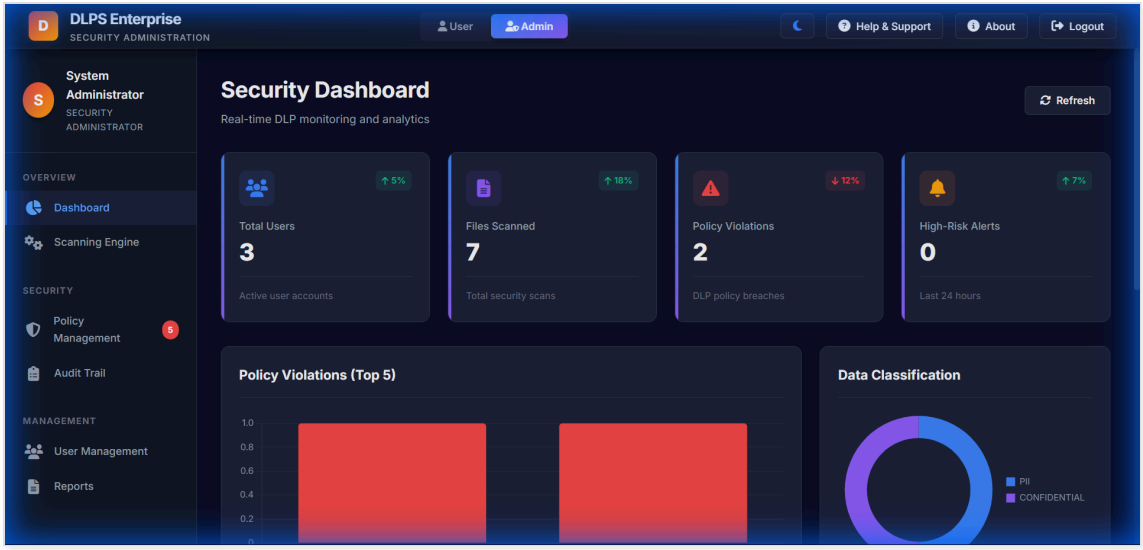
Users can drag and drop files for real-time DLP scanning.



8.3 Admin Portal

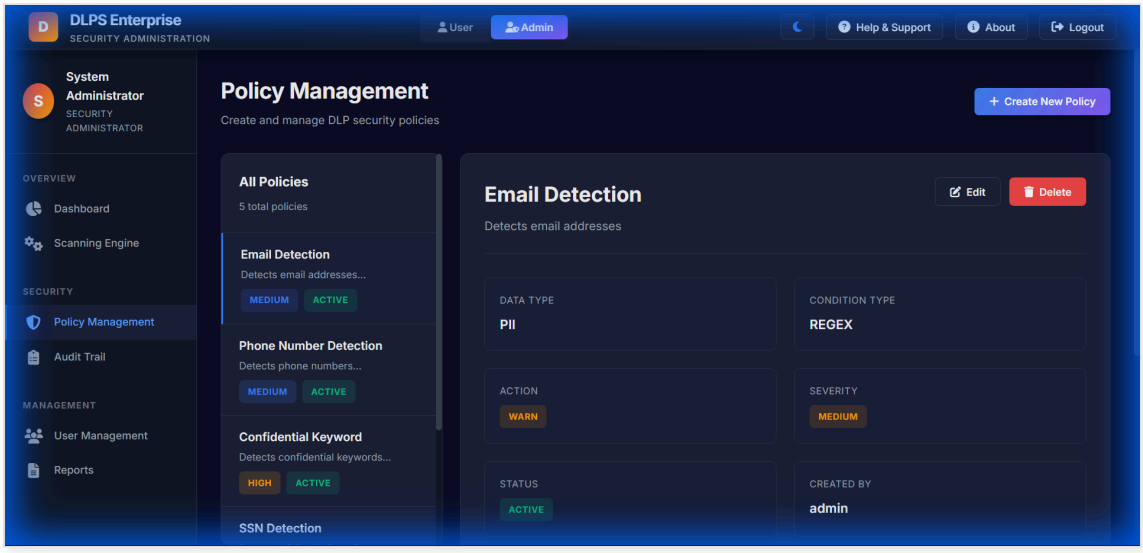
Admin Dashboard

A SOC-style dashboard showing scanning trends, risk distribution, and system health.



Policy Management

Admins can verify and configure regex patterns for data detection.



Audit Trails

Immutable logs of all system activities for forensic analysis.