Software Requirements Specification (SRS)

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to outline the requirements for a Data Leakage Prevention (DLP) web application that enables users to upload files, check for data breaches, and manage file access securely. Admins have extended privileges to manage users and monitor system activities.

## 1.2 Scope

This DLP system is a web-based platform that:  
- Prevents sensitive file leaks.  
- Checks for email and phone leaks using HIBP API (RapidAPI).  
- Supports role-based access control.  
- Enables auditing and reporting.  
- Allows file uploads with antivirus scans and ownership mapping.

## 1.3 Intended Audience

- Developers building or maintaining the system.  
- System Administrators deploying and configuring the system.  
- Security Analysts auditing file usage and leak status.  
- End Users uploading and checking personal file breaches.

# 2. Overall Description

## 2.1 Product Perspective

This system is a standalone Flask web app running on a local or cloud server. It uses a SQLite database and REST APIs to check for data leaks.

## 2.2 User Classes and Characteristics

- Admin: Full control over users, files, roles, logs. Can impersonate users and export reports.  
- User: Can upload/view/delete their own files. Can check personal email/phone for leaks.

## 2.3 Assumptions and Dependencies

- The system uses the HaveIBeenPwned API via RapidAPI.  
- Internet connectivity is required for breach checking and email services.  
- File storage is local unless cloud storage is integrated later.

# 3. Functional Requirements

## 3.x User Authentication

* F1: User can register with email and password.
* F2: Login with hashed password check.
* F3: Password reset via tokenized email link.
* F4: Email verification during signup.
* F5: Two-factor authentication (optional phase).

## 3.x Role-Based Access Control

* F6: Role field (Admin/User) determines privileges.
* F7: Only admins can update user roles or delete users.

## 3.x File Upload and Management

* F8: Authenticated users can upload files.
* F9: Files are tagged to uploaded\_by user.
* F10: File type is detected by extension.
* F11: File size limit enforced.
* F12: Antivirus scan on upload (ClamAV or similar).
* F13: Admins can delete any file; users only their own.

## 3.x Leak Check Module

* F14: Check for email leaks using HIBP API.
* F15: Check for phone number leaks via HIBP (RapidAPI).
* F16: Results displayed with date, breach source, etc.
* F17: Optional email alert if leak is found.

## 3.x Admin Dashboard

* F18: View users and files in tabular format.
* F19: Search and sort by username, file name, etc.
* F20: View audit logs (login, upload, delete).
* F21: Generate reports (CSV).
* F22: Impersonate user view for debugging.

# 4. Non-Functional Requirements

## 4.x Performance

* Should support 100+ concurrent users with low latency.
* File upload limit: 50 MB (configurable).

## 4.x Usability

* Mobile responsive UI.
* Dark theme with accessible fonts and icons.

## 4.x Reliability

* Session timeout after inactivity.
* Graceful error handling.

## 4.x Security

* Passwords stored as bcrypt hashes.
* HTTPS recommended for deployment.
* XSS, CSRF, and SQL injection protections enabled.

## 4.x Portability

* Works on any modern browser.
* Cross-platform (Linux/Windows hosting).

# 5. External Interface Requirements

## 5.1 User Interfaces

/register: Signup form.  
/login: Login form.  
/dashboard: File upload, leak check.  
/admin: Admin control panel.

## 5.2 Hardware Interfaces

None specific; depends on server hosting.

## 5.3 Software Interfaces

Flask (backend), SQLite (DB), Jinja2 (templates),  
HIBP API via RapidAPI,  
SMTP service for password reset/verification

# 6. Future Enhancements

* Cloud storage integration (e.g., S3 or Firebase).
* AI-based anomaly detection in file upload behavior.
* Visual analytics using Plotly or Chart.js.
* External OAuth (Google/Microsoft) login.
* File content redaction for sensitive data.