International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Design and Implementation of a Secure File Sharing Web Application with Expiry and Malware Detection

Omkar Shinde

ADYPU`

ABSTRACT:

In the digital era, secure and controlled file sharing is an indispensable necessity across enterprises, academic institutions, and individual users. Traditional file-sharing platforms often lack built-in mechanisms for link expiration, password protection, and malware detection. This paper presents a Secure File Sharing Web Application developed using open-source technologies such as Node.js, Express.js, and ClamAV antivirus engine. The system enables users to upload files, set download expiry durations, protect downloads via passwords, and alert recipients about malicious files. The application demonstrates scalability, efficiency, and user-centric design, providing a viable alternative to generalized platforms. The research includes architectural design, system development, user evaluation, and performance analysis.

Keywords: File sharing, Web security, Link expiration, ClamAV, Node.js, Malware detection, Password protection

Introduction

1.1 Background As digital file exchanges grow in volume and sensitivity, security concerns surrounding uncontrolled access, data leaks, and malware propagation have escalated. File-sharing platforms must now accommodate dynamic control mechanisms that ensure confidentiality and trustworthiness of shared data.

1.2 Motivation Most mainstream file-sharing systems offer either minimal security or rely on user discretion. Users have limited means to restrict access or guarantee malware-free transfers. The need for a reliable, lightweight tool that includes security-first capabilities inspired the creation of this system.

1.3 Research Problem Existing systems lack sufficient, built-in mechanisms for access limitation and malware detection. Manual workarounds pose usability and reliability risks. How can an open-source web application offer secure file sharing with real-time malware detection, link expiry, and password-based access?

1.4 Objectives

- Develop a secure file-sharing platform
- Integrate link expiration and access protection
- Implement antivirus scanning with ClamAV
- Design a minimal and intuitive UI
- Evaluate performance and security outcomes

1.5 Methodology The research follows a waterfall-based approach for clear phase-wise development: requirement analysis, design, implementation, testing, and evaluation. Data was collected from user tests and benchmarked using standard web application performance metrics.

Literature Review

2.1 Existing File Sharing Solutions Dropbox, Google Drive, and WeTransfer dominate cloud-based file transfer. However, these platforms often lack direct malware detection and fully customizable sharing control.

2.2 Shortcomings in Security and Control Studies reveal that over 30% of data leaks originate from publicly shared links without proper restrictions. There is limited support for link expiration and password protection, especially in free tiers. According to a survey by Cybersecurity Ventures (2023), 61% of users expressed concern over link persistence beyond intended timeframes.

2.3 Use of ClamAV and Similar Antivirus Tools ClamAV is an open-source antivirus engine suitable for server-side integration. It supports real-time scanning and signature updates, making it ideal for on-the-fly malware analysis during file transfer. It is commonly integrated with content management systems and custom backend applications, where latency and security must coexist.

2.4 Open-source Web Development Frameworks Node.js and Express.js offer non-blocking, event-driven development for responsive web systems. These technologies support modular, secure coding and easy middleware integrations. Their open-source nature ensures wide compatibility and an active community for troubleshooting and optimization.
2.5 User Experience in Secure Web Applications Modern UX research emphasizes usability, feedback loops, and minimal steps in user workflows. Applications should offer security features without intimidating non-technical users. In this context, hiding complexity behind a simple interface is essential. Progressive Disclosure is often used in such systems.

System Architecture and Design

3.1 System Overview The system allows file uploads with password options and expiry settings. The backend scans files and returns a downloadable link only if the file is clean and accessible. A JSON-based metadata store is used to track file parameters including expiry, hash, and scan status.

3.2 Component Architecture

- Frontend: HTML, CSS, JavaScript (Vanilla)
- Backend: Node.js with Express.js
- File Handler: Multer
- Scanner: ClamAV Daemon + clamdjs client
- Metadata: Lightweight JSON store
- Security: SHA-256 for password hashing

3.3 Flow Diagrams Use Case: Upload → Scan → Save Metadata → Generate Link → Share → Access → Validate (Password/Expiry/Scan) → Download Sequence: User → Upload → Server (File Save + Metadata Save) → ClamAV → Metadata Update → Server Response → Share

3.4 Technology Stack

- Node.js v18.0+
- Express.js Framework
- ClamAV (latest signatures updated via freshclam)
- HTML5, CSS3 for frontend
- JSON for internal data persistence
- Optional: Docker for deployment

Development Methodology

4.1 Requirements Specification Functional Requirements:

- Upload File with metadata
- Generate expiring links
- Password-protect download access
- Real-time malware scanning
- Warn users on infected file download attempt

Non-Functional Requirements:

- Responsive within 3 seconds
- Cross-browser compatibility
- Accessible UI for low-literacy users
- Platform independence

4.2 Feature Implementation File upload and scanning are tightly coupled. Files are not served unless scanned clean. Passwords are stored as SHA-256 hashes. Expiry is enforced using a timestamp comparison logic.
4.3 Integration of ClamAV clamdjs is configured to connect with the ClamAV daemon on the local machine. File buffers are piped directly to the scanner and results are asynchronously stored in metadata. If infected, the user receives a warning with no download option.
4.4 Link Expiry Logic Each file receives a timestamp during upload. The download route checks if the current UNIX time exceeds the saved expiry. Expired links are rejected with a 403 Forbidden message.
4.5 Password Protection Mechanism Upon link access, the user is prompted for a password (if set). The password is hashed and compared against the stored hash. A mismatch denies access. This avoids storing plaintext credentials.

Testing and Evaluation

5.1 Unit Testing Modules for upload, hash generation, scan initiation, and expiry validation were tested using Mocha/Chai.
5.2 Integration Testing Complete file lifecycle tested with clean and dummy-malicious files. Tested behavior under invalid access conditions.
5.3 Security Testing Simulated brute-force attacks, expired links, and access without password. System responded correctly to all edge cases.
5.4 Pilot User Evaluation A test group of 15 users interacted with the app. 93% were able to share and retrieve files without assistance. Overall system rating was 4.7/5.

5.5 Usability Analysis Feedback indicated high satisfaction with simplicity. Suggested improvements included mobile layout optimization and email-based link notifications.

Results and Discussion

6.1 Performance Metrics

- Upload latency (under 2.5s for files <10MB)
- Scan accuracy: EICAR tests blocked 100%
- Expiry enforcement: Verified across 5 timing thresholds

6.2 User Feedback Users appreciated password prompt and malware alerts. Some requested drag-and-drop support and session-based upload history.

6.3 Functionality Comparison Compared to Dropbox and Google Drive:

- ***Pros: Security by default, no account required***
- Cons: No large file handling or cloud sync
- Unique: ClamAV integration and no login

6.4 Threat Mitigation Tested scenarios included:

- Infected PDF files (simulated)
- Shared links with wrong passwords
- Expired links beyond set duration All scenarios were successfully blocked.

Limitations and Future Work

7.1 Technical Constraints

- Local storage only
- Dependency on manually updated ClamAV signatures
- Basic UI; lacks bulk upload

7.2 Scalability Concerns The system does not support concurrent users >100 efficiently without DB optimization or CDN support.

7.3 Feature Suggestions

- Add login system
- File preview option
- Integration with cloud storage (AWS/GCP)

7.4 Enhancement Roadmap

- Add upload limits and size filters
- Generate audit logs
- Use Redis cache for temporary link expiry
- OTP-based link sharing for one-time access

Conclusion

The Secure File Sharing Web Application delivers a reliable platform for secure data exchange with real-time scanning and privacy control. It stands out with its minimal design, extensible backend, and direct malware protection workflow. This system demonstrates a practical and educational approach to implementing security-first web applications in real-world scenarios.

REFERENCES:

[1] Node.js Documentation – https://nodejs.org
[2] Express.js Framework – https://expressjs.com
[3] ClamAV Antivirus Engine – https://www.clamav.net
[4] GitHub Developer Best Practices – https://github.com
[5] OWASP Security Guidelines – https://owasp.org/www-project-top-ten/
[6] Cybersecurity Ventures 2023 Report – https://cybersecurityventures.com
[7] Stack Overflow Developer Survey 2023 – https://survey.stackoverflow.co/2023/