International Journal of Artificial Intelligence and Computer Electronics
Volume: 02, Issue: 03, July-September 2026, pp:44-53
P-ISSN 3117-8529 | E-ISSN 3117-8537

A Secure Multi-Layer Framework For File Upload Validation, Encryption, And Controlled Access

Mohammed Abubakar Siddiq[1], Dr. Md Zainlabuddin[2]
[1]PG Scholar, Department Of Computer Science & Engineering ISL Engineering College, Hyderabad, India.
[2]Associate Professor; Department Of Computer Science & Engineering ISL Engineering College, Hyderabad, India
Corresponding Author Email: abusmas38@gmail.com

Article Received 22-05-2026, Revised 29-06-2026, Accepted 11-08-2026

Abstract
File upload functionality is widely used in modern applications and communication platforms, including Internet of Things (IoT) devices, smart homes, and smart city environments, to support efficient data exchange and content sharing. However, insecure file upload mechanisms can create unrestricted file upload (UFU) vulnerabilities that threaten the confidentiality, integrity, and availability of communication systems. This study examines the major security risks, underlying causes, and potential consequences associated with UFU vulnerabilities. Such weaknesses may be exploited without extensive user interaction or privileged access, potentially leading to unauthorized file execution, system compromise, service disruption, and reduced network reliability. The study highlights the importance of implementing robust file validation, access control, secure storage, and monitoring mechanisms to strengthen upload security. The findings demonstrate the need for continued research and improved defensive approaches to protect communication systems and maintain reliable service delivery in emerging smart environments.

Keywords: *Unrestricted File Upload (UFU), File Upload Security, Cybersecurity, Internet of Things (IoT), Smart Home, Smart City, Communication Systems, Secure File Management, Access Control, Data Security.*

1. Introduction

File uploading has become a routine function in web applications, cloud platforms, enterprise systems, and connected environments. Users and devices frequently transfer documents, images, videos, configuration files, and other digital resources through upload interfaces. Although this functionality improves information exchange and service usability, it also introduces an important security boundary because files supplied by users cannot always be assumed to be trustworthy. A poorly designed upload mechanism may allow an attacker to submit content that is executable, malformed, oversized, or otherwise inconsistent with the intended application requirements. Consequently, file upload security has become an important component of application and data protection.

The problem becomes more significant in IoT, smart-home, and smart-city environments, where a large number of heterogeneous devices and services continuously exchange information. Such systems may process data through web interfaces, cloud services, edge platforms, and communication networks. A weakness in any upload endpoint can therefore provide an entry point into a larger application ecosystem. Research on network and application security has demonstrated that vulnerabilities in supporting infrastructure can have consequences beyond the individual application in which they originate [26].

One of the major concerns associated with upload functionality is the **unrestricted file upload (UFU) vulnerability**. This vulnerability occurs when an application accepts files without sufficiently verifying their type, content, size, name, or destination. The uploaded content may subsequently be stored in a location where it can be executed or interpreted by the server. The OWASP security guidance identifies unrestricted file upload as a significant application-security concern because inadequate validation can allow an attacker to introduce potentially harmful content into the application environment [12]. MITRE similarly classifies unrestricted upload of dangerous file types under CWE-434, emphasizing the importance of restricting uploaded content according to the application's intended requirements [22].

File validation therefore represents the first defensive layer of a secure upload architecture. Simple extension checking alone is insufficient because an attacker may manipulate filenames, MIME information, or file contents. Studies of upload vulnerabilities have demonstrated the need for more comprehensive validation and testing mechanisms [24]. Research into automated vulnerability discovery has also shown that upload-related weaknesses can remain hidden when

applications rely on incomplete validation strategies [20].

Another concern is the possibility of specially constructed files that contain multiple interpretations or formats. Polyglot files are particularly relevant because a single file may satisfy the structural requirements of more than one file format and potentially bypass simplistic validation mechanisms. Magazinius et al. investigated such format-crossing techniques and demonstrated the security implications of polyglot files [14]. Similarly, research into dangerous document processing has shown that apparently legitimate file formats can contain security-sensitive structures that require careful handling [23].

Secure storage and confidentiality constitute additional requirements after a file has passed validation. Even when malicious uploads are prevented, sensitive legitimate files may still be exposed if they are stored in plaintext or accessed without adequate authorization. Encryption can reduce the consequences of unauthorized access by ensuring that stored content cannot be directly interpreted without the appropriate cryptographic protection. Controlled access mechanisms are therefore required alongside validation and encryption rather than being treated as separate security functions.

The security problem is particularly important for systems involving mobile and distributed users. Smartphone applications and wireless environments may expose data through multiple communication paths, increasing the importance of secure file handling and access practices [7]. Network-level security mechanisms also contribute to the protection of information as it moves between clients, servers, and supporting services [11]. Furthermore, modern cloud and serverless infrastructures introduce additional processing components that must be considered when designing secure upload workflows [19].

Based on these challenges, this study proposes a **Secure Multi-Layer Framework for File Upload Validation, Encryption, and Controlled Access**. The framework combines input validation, file-type and content verification, secure storage, encryption, authorization, and monitoring into a unified protection process. Instead of depending on a single security mechanism, the proposed approach establishes multiple defensive stages so that failure at one layer can be compensated for by subsequent controls. The objective is to reduce the likelihood of malicious file acceptance, protect stored information, restrict unauthorized access, and improve the overall security of file exchange in connected application environments.

2. Literature Review

N. Uddin and M. Jabr investigated file upload security and validation in the context of Software as a Service cloud environments. Their work highlights the importance of carefully validating uploaded files because cloud-hosted applications can expose file-processing functionality to remote users. The study provides an important foundation for developing secure upload mechanisms in distributed application environments [24].

The Open Web Application Security Project (OWASP) identifies unrestricted file upload as a significant web application security weakness. Its guidance emphasizes that applications should not blindly trust files supplied by users and should implement appropriate restrictions and validation before accepting and storing uploaded content. This work provides practical security guidance for designing the validation layer of the proposed framework [12].

MITRE Corporation documents unrestricted upload of files with dangerous types under CWE-434. The weakness classification highlights the security consequences of accepting file types that an application does not actually require. This provides a useful basis for establishing file-type restrictions and implementing controlled upload policies within secure application architectures [22].

T.-J. Lee et al. developed Fuse, a technique for discovering file-upload vulnerabilities through penetration testing. Their research demonstrates that upload vulnerabilities can be identified by systematically examining the behavior of upload functionality and testing the application's response to potentially unsafe inputs. The findings support the use of security testing as an important component of a multi-layer upload protection framework [8].

Y. Chen et al. proposed Uradar for discovering unrestricted file-upload vulnerabilities through adaptive dynamic testing. Their approach emphasizes the importance of dynamically examining application behavior rather than relying solely on static inspection. This research is relevant to the proposed framework because effective protection requires understanding how uploaded content is processed after it enters the application [20].

J. Huang et al. investigated Ufuzzer, a lightweight method for detecting PHP-based unrestricted file-upload vulnerabilities through static and fuzzing co-analysis. Their work combines different analysis perspectives to identify weaknesses that may not be apparent through a single testing technique. This supports the broader principle that upload security should involve multiple complementary defensive and analytical mechanisms [2].

H. Oz et al. examined the security of file uploads in Node.js applications and highlighted the risks associated with insufficient handling of uploaded

content in modern server-side environments. Their study demonstrates that the underlying application framework does not automatically eliminate upload-related risks; instead, developers must explicitly establish appropriate validation and processing controls [16].

M. Müller et al. investigated security and privacy issues associated with processing dangerous paths and Portable Document Format files. Their work demonstrates that security problems may continue beyond the initial upload stage and can emerge during file parsing or subsequent processing. This finding is important for the proposed framework because validation should be complemented by safe storage and controlled processing mechanisms [23].

M. Zimmermann et al. studied security threats within the npm ecosystem and demonstrated how dependencies and software components can contribute to application security risks. Their findings are relevant to file-upload systems because uploaded content is often processed through external libraries and application components. Therefore, the security of the complete processing chain must be considered rather than concentrating only on the upload interface [4].

X. Li and Y. Xue surveyed server-side approaches for securing web applications and discussed multiple mechanisms for protecting application functionality against different classes of attacks. Their work reinforces the need for layered security controls rather than depending on a single defensive mechanism. This principle directly supports the multi-layer architecture proposed in this study [6].

M. D. Zainlabuddin and N. Sharma investigated security enhancement in data propagation for wireless networks. Their research emphasizes protecting information as it moves through communication environments. This perspective is relevant to secure file management because uploaded information may pass through several network components before reaching its protected storage location [11].

Mohammed Abdul Bari, Shahanawaj Ahamad, and Mohammed Rahmat Ali examined smartphone security and protection practices. Their work highlights the importance of secure user-side practices in environments where sensitive information is accessed and exchanged through mobile devices. Such considerations are relevant to the proposed framework because secure upload protection must account for both application-side controls and the environments from which files are submitted [7].

Overall, existing research establishes that unrestricted file upload vulnerabilities cannot be adequately addressed through filename or extension checking alone. The literature points toward the combined use of **strict validation, dynamic testing, secure processing, encryption, access control, and continuous monitoring**. However, many approaches focus on individual stages of the upload lifecycle. The proposed framework addresses this limitation by integrating these protective mechanisms into a unified multi-layer architecture in which uploaded files are validated before acceptance, protected through encryption during storage, and made available only through controlled authorization mechanisms.

3. Methodology

The proposed methodology focuses on securing the file upload process through effective validation and controlled storage. A user selects a file through the application interface, after which preliminary checks may be performed on the client side. Since client-side controls can be bypassed, the primary security mechanisms are implemented on the server. The server examines the uploaded file according to defined security policies and either rejects invalid submissions or processes approved files for storage. The methodology recognizes the growing use of reusable upload libraries, particularly in web development environments such as Node.js, and emphasizes the need to secure both application-level upload logic and third-party components.

The proposed system uses a modular architecture consisting of User, Client, Admin, Server, and Database components to provide secure file upload, storage, and controlled access. Files submitted by users pass through multiple server-side validation stages before being accepted for processing. Approved files are encrypted using RSA, while SHA-256 hash values are generated to support integrity verification. The server manages secure storage, access requests, and communication between system components. Authorized clients can search for available files and request access, while the administrator manages registrations and supervises upload and access activities. The database maintains file metadata, encrypted information, integrity values, and related key-management details. Logging and monitoring functions support auditing and security analysis, while access-control mechanisms restrict retrieval to authorized entities. The modular design improves scalability and fault isolation and provides a foundation for future extensions such as cloud integration and enhanced cryptographic protection.

International Journal of Artificial Intelligence and Computer Electronics
Volume: 02, Issue: 03, July-September 2026, pp:44-53
P-ISSN 3117-8529 | E-ISSN 3117-8537

System Architecture

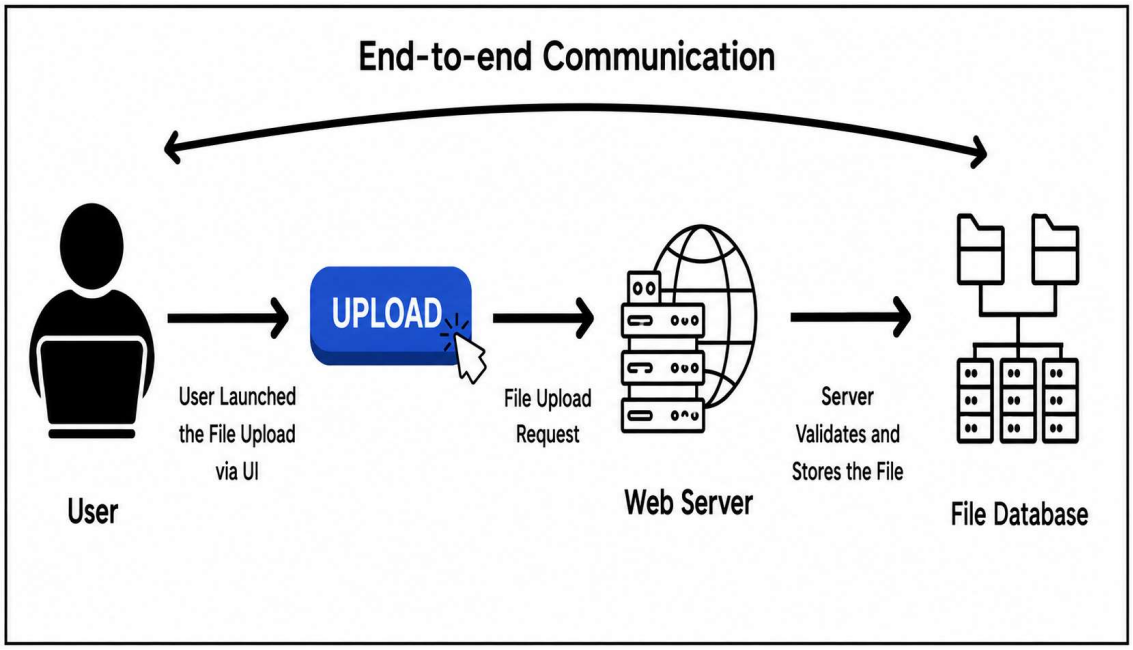

Figure 1: System Architecture

4. Implementation

The implementation phase represents the conversion of the proposed system design into an operational software application. It involves developing the source code required to implement the administrative, client, user, and server functionalities. The system is implemented using Java Servlet technology, JSP, HTML, and CSS, with database connectivity used to manage user credentials, registration information, authentication details, and access permissions. The implementation focuses on providing secure authentication, controlled access, and an interactive interface for the different system modules.

Admin Backend Implementation:

The administrator login functionality is implemented through a Java Servlet that processes the administrator's email address and password submitted through the login interface. The servlet receives the authentication parameters, validates them against the configured administrator credentials, and redirects the administrator to the administrative home page when authentication is successful. If the supplied credentials are invalid, the request is redirected back to the administrator login page. The servlet supports both GET and POST requests, allowing the login process to be handled through standard HTTP communication. This module provides the initial authentication mechanism for accessing administrative functions.

Client Login Implementation:

The client authentication module is implemented using a Java Servlet integrated with the application database. The login process incorporates OTP-based verification as an additional authentication layer. When the client submits the login information, the system retrieves the OTP, associated email address, and generation time from the active session. The submitted email address is first compared with the email address associated with the generated OTP. The system also verifies whether the OTP remains within its defined validity period of ten minutes. Expired or missing OTP information results in an appropriate error message and requires the client to initiate the authentication process again.

After successful OTP validation, the entered credentials are checked against the client registration records stored in the database. A prepared SQL statement is used to retrieve the corresponding record only when the account has an approved registration status. If the credentials are valid, the temporary OTP information is removed from the session, the client's email address is stored in the session, and the client is redirected to the client home page. Invalid credentials, incorrect OTP

International Journal of Artificial Intelligence and Computer Electronics
Volume: 02, Issue: 03, July-September 2026, pp:44-53
P-ISSN 3117-8529 | E-ISSN 3117-8537

values, invalid OTP formats, or database-related errors are handled through suitable messages and redirection to the login interface. This implementation provides an additional authentication layer and ensures that only approved clients can access protected system resources.

Login Interface Implementation:

The login interface is developed using JSP, HTML, and CSS to provide an interactive and user-friendly authentication environment. The interface presents the major modules of the system, including User, Client, Admin, and Server, with each module providing access to its corresponding functionality. The user module supports registration and file-related operations, while the client module enables authorized users to search for files, request decryption keys, and obtain approved files. The administrator module is responsible for managing registrations and monitoring users and clients, whereas the server module handles file verification and data management.

The login page provides fields for entering the user's email address and password and includes an OTP-generation mechanism. After an OTP is generated, an additional field is displayed for entering the received verification code. The interface also provides appropriate messages for authentication errors and includes a registration option for new users. CSS is used to improve the visual appearance through responsive layouts, styled cards, gradients, animations, buttons, and form components. The module-based design makes navigation easier while maintaining a consistent presentation throughout the application.

5. Results And Discussion

Snapshots

The proposed system is implemented as a web-based application using Core Java technologies. Server-side communication and processing are managed through Java networking and web components, while the user interface is developed using JSP, HTML, and Cascading Style Sheets. The application integrates multiple modules to provide controlled file uploading, verification, storage, access management, and monitoring.

Various Snapshots

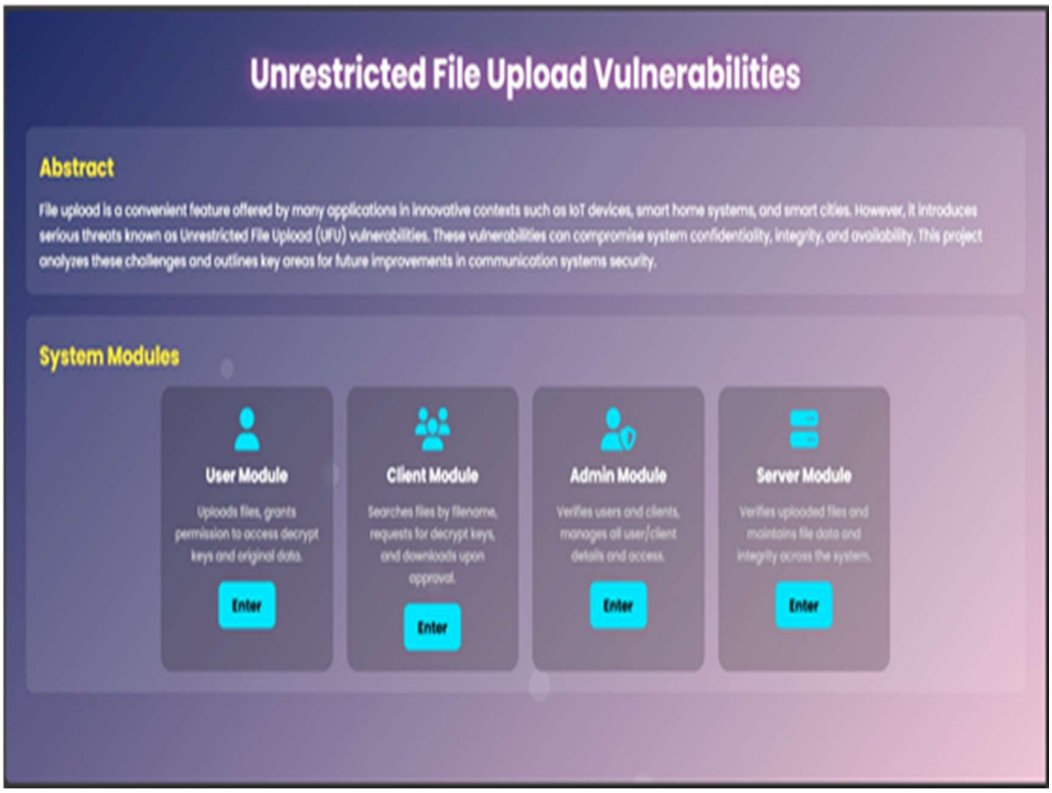

Figure 2

The system consists of four major modules: User, Client, Admin, and Server. Each module performs specific operations within the secure file management framework. The User module is

International Journal of Artificial Intelligence and Computer Electronics
Volume: 02, Issue: 03, July-September 2026, pp:44-53
P-ISSN 3117-8529 | E-ISSN 3117-8537

responsible for file uploading and access-related activities. The Client module allows authorized users to search for files and submit requests for access. The Admin module manages user and client verification, while the Server module handles file validation, storage, and related security operations. Together, these modules establish a controlled environment for secure file handling.

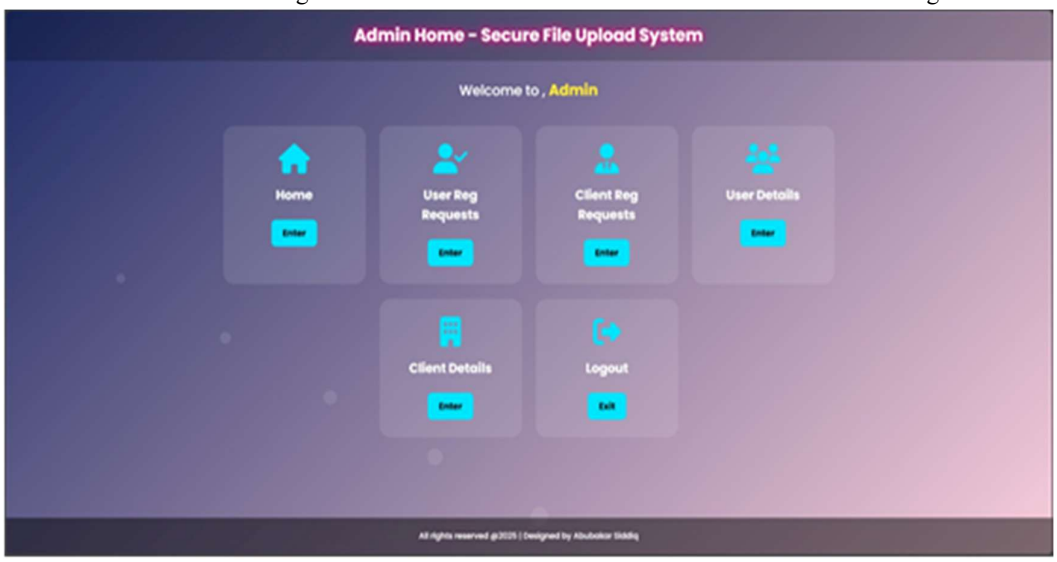

Figure 3

The Admin Home interface provides centralized access to administrative functions. Through this dashboard, the administrator can review user registration requests, client registration requests, user information, and client information. The administrator can approve and manage authorized entities and securely terminate the session through the logout facility.

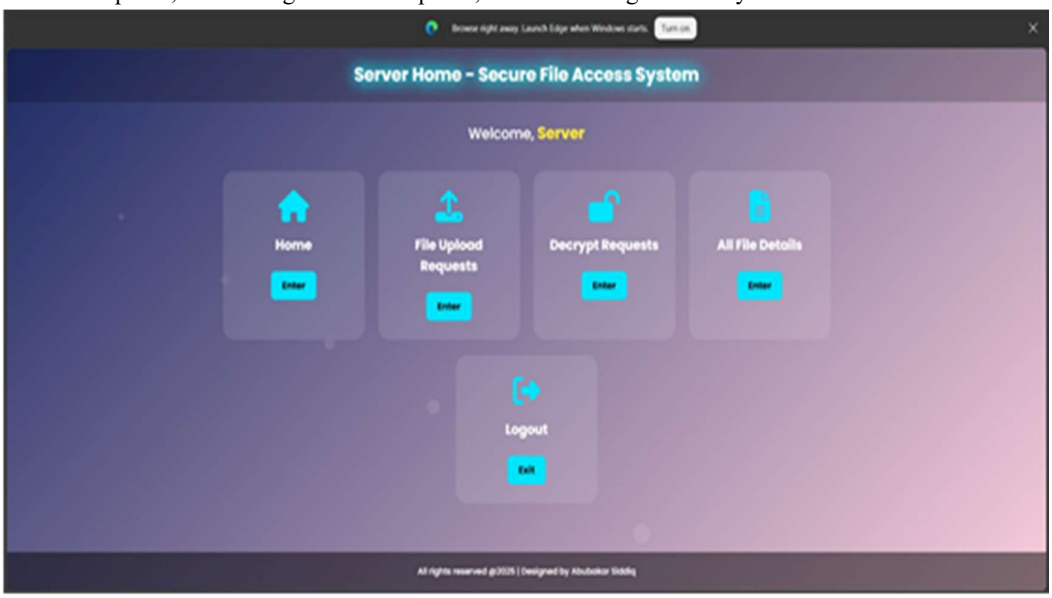

Figure 4

The Server Home interface provides access to important server-side functions. It enables the management of file upload requests, decryption or access requests, and file-related information. The server component processes requests and maintains controlled access to protected files stored within the system.

International Journal of Artificial Intelligence and Computer Electronics
Volume: 02, Issue: 03, July-September 2026, pp:44-53
P-ISSN 3117-8529 | E-ISSN 3117-8537

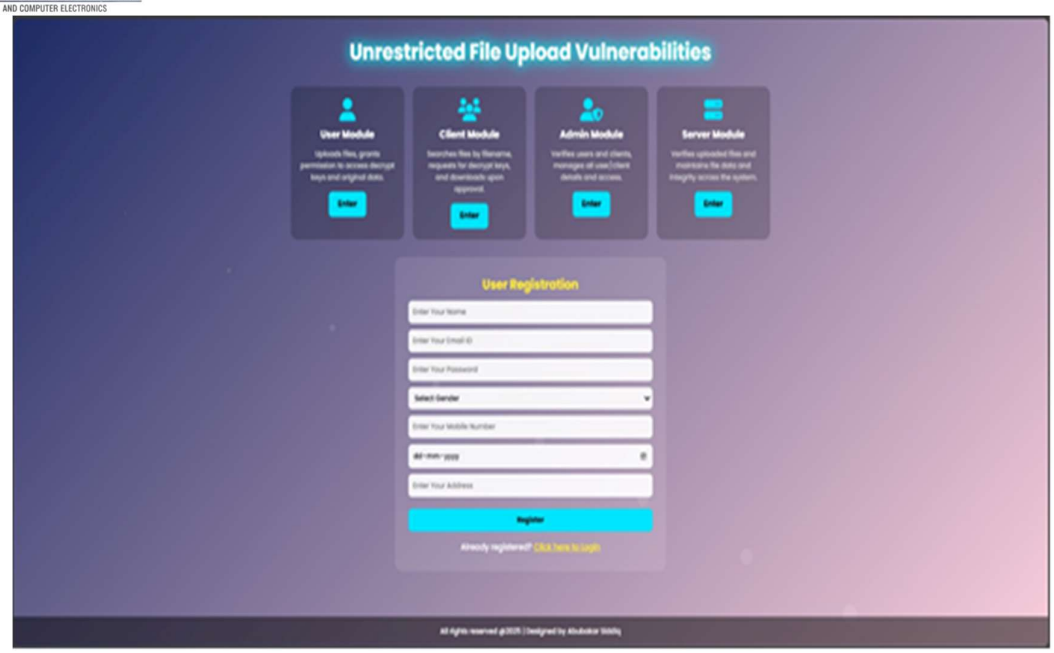

Figure 5

The User Registration interface allows new users to create an account by submitting required details such as name, email address, password, gender, mobile number, date of birth, and address. After registration, user information can be processed according to the system's approval mechanism. Existing users can access the login option to enter the application.

Figure 6

The User Home interface provides the main file management functions for authorized users. Users can upload files, monitor the status of submitted files, and initiate requests related to decryption or access keys. The logout function securely closes the active user session.

Software Testing

General

Software testing is performed to identify errors, weaknesses, and unexpected behavior in an application before deployment. It helps determine

whether the developed system operates according to its specified requirements and meets user expectations. Testing is applied to individual components as well as the complete integrated system to ensure that failures are identified and corrected at different stages of development. Different testing techniques are used because each technique examines a particular aspect of system quality.

Developing Methodologies

The testing methodology begins with the preparation of a structured test plan covering both general system functions and specific security-related features. The application is evaluated under relevant operating conditions to verify that the implemented modules satisfy the requirements defined during system development. Quality control procedures are used to identify defects, validate expected behavior, and improve the reliability of the application. The testing framework is organized according to the functions and interactions of individual system modules.

6. Conclusion

The project successfully presents a Secure Multi-Layer File Upload Validation System for improving the protection of web-based file handling operations. The proposed framework integrates User, Client, Admin, and Server modules to establish a controlled process for user registration, authentication, file uploading, validation, storage, access requests, and monitoring.

Multiple security mechanisms are incorporated to strengthen file protection. Server-side validation helps control uploaded content, RSA-based encryption protects sensitive file data, and SHA-256 hashing supports integrity verification by helping identify unauthorized modifications. Administrative approval and controlled access procedures further restrict file-related operations to authorized entities. The modular design improves the organization and maintainability of the application while allowing different responsibilities to be managed independently. Testing of the system supports the verification of individual functions and module interactions. Overall, the project demonstrates that combining validation, cryptographic protection, authentication, integrity checking, and controlled access can provide a more secure approach to file management and help reduce risks associated with unrestricted file upload vulnerabilities.

Future Enhancement

The proposed system can be extended with additional security and functionality to address future requirements and evolving cyber threats. Hybrid encryption mechanisms may be introduced to improve the efficiency of protecting large files, while role-based access control can provide more detailed permission management for different categories of users.

Future versions may also incorporate real-time malware scanning to detect suspicious files before storage. Multi-factor authentication can further strengthen user and client verification. Cloud-based storage integration could improve scalability and support distributed file management.

Automated auditing and reporting features can provide administrators with improved visibility into upload and access activities. Tamper-resistant logging mechanisms may also be incorporated to strengthen accountability. AI-based anomaly detection could assist in identifying unusual upload behavior or suspicious access patterns.

Additional enhancements may include support for large files, batch uploads, intelligent file searching, mobile compatibility, cross-platform access, and stronger protection for data transmitted between system components. These improvements can help make the system more scalable, efficient, user-friendly, and resilient against emerging security challenges.

References

[1]. Anjani Haritha Sannidhanam. (2024). Guardrails and Safety Mechanisms for LLM-Powered Enterprise Applications. *International Journal of Engineering Science & Humanities, 14*(3), 273–285.

[2]. J. Huang *et al*., “Ufuzzer: Lightweight detection of PHP-based unrestricted file upload vulnerabilities via static-fuzzing co-analysis,” *Association for Computing Machinery*, 2021.

[3]. V. K. B. Parasaram, V. T. Bathini, S. K. Nalluri, & A. H. Sannidhanam. (2026). A Sustainable AI-Enabled Predictive Maintenance Framework for Smart Industrial Systems Using Industrial IoT Data. *2026 Third International Conference on Innovations in Cybersecurity and Data*

Science (ICICDS), 440–447. doi:10.1109/ICICDS70526.2026.11604878

[4]. M. Zimmermann *et al*., "Small world with high risks: A study of security threats in the npm ecosystem," in *Proc. 28th USENIX Security Symp.*, 2019.

[5]. Sannidhanam, A. H. (2020). Comparative Analysis of Cloud Computing Architectures for Enterprise Applications. *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology, 12*(02), 169–180. doi:10.18090//samriddhi.v12i02.16

[6]. X. Li and Y. Xue, "A survey on server-side approaches to securing web applications," *ACM Comput. Surveys*, vol. 46, no. 4, 2014. doi: 10.1145/2541315.

[7]. Mohammed Abdul Bari, Shahanawaj Ahamad, Mohammed Rahmat Ali," *Smartphone Security and Protection Practices*", International Journal of Engineering and Applied Computer Science (IJEACS) ; ISBN: 9798799755577 Volume: 03, Issue: 01, December 2021 (International Journal,U K) Pages 1-6

[8]. T.-J. Lee *et al*., "Fuse: Finding file upload bugs via penetration testing," in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2020.

[9]. Performance Analysis of Wireless Sensor Networks for Smart Monitoring Applications. (2016). *International Journal of Humanities and Information Technology, 1*(04), 10–29. doi:10.21590/ijhit.01.04.04

[10]. Ijteba Sultana, **Dr. Mohd Abdul Bari** ,Dr. Sanjay," *Routing Performance Analysis of Infrastructure-less Wireless Networks with Intermediate Bottleneck Nodes*", International Journal of Intelligent Systems and Applications in Engineering, ISSN no: 2147-6799 IJISAE,Vol 12 issue 3, 2024, Nov 2023

[11]. M. D. Zainlabuddin and N. Sharma, "Security enhancement in data propagation for wireless network," *Rev. GEINTEC—Gestão, Inovação e Tecnologias*, vol. 11, no. 4, pp. 4110–4119, Aug. 2021.

[12]. Open Web Application Security Project (OWASP), "Unrestricted file upload," 2024. [Online]. Available: OWASP Unrestricted File Upload. Accessed: Apr. 20, 2024.

[13]. Sannidhanam, A. H. (2021). Real-Time Claims Processing Using Event-Driven Architectures. *International Journal of Technology, Management and Humanities, 7*(01), 36–50. doi:10.21590/07.01.02

[14]. J. Magazinius *et al*., "Polyglots: Crossing origins by crossing formats," in *Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS)*, 2013, pp. 753–764. doi: 10.1145/2508859.2516685.

[15]. M. S. Uddin and M. D. Zainlabuddin, "Secure video processing and watermark embedding using Shamir secret rule," *Int. J. Artif. Intell. Comput. Electron.*, vol. 2, no. 1, pp. 25–31, Mar. 2026.

[16]. H. Oz *et al*., "(In)Security of file uploads in Node.js," in *Proc. ACM Web Conf.*, 2024.

[17]. Sannidhanam, A. H. (2026). LLM-Driven Workflow Optimization: Intelligent Routing in Asynchronous Distributed Systems. *International Journal of AI and Machine Learning, 1*(2), 23–33.

[18]. Forum of Incident Response and Security Teams (FIRST), "CVSS v3.0 specification document," 2015. [Online]. Available: FIRST CVSS v3.0 Specification. Accessed: Aug. 3, 2024.

[19]. Dr. Mohammed Abdul Bari,Arul Raj Natraj Rajgopal, Dr.P. Swetha ," *Analysing AWSDevOps CI/CD Serverless Pipeline Lambda Function's Throughput in Relation to Other Solution*", International Journal of Intelligent Systems and Applications in Engineering , JISAE, ISSN:2147-6799, Nov 2023, 12(4s), 519–526

[20]. Y. Chen *et al*., "Uradar: Discovering unrestricted file upload vulnerabilities via adaptive dynamic testing," *IEEE Trans. Inf. Forensics Security*, 2024.

[21]. Anjani Haritha Sannidhanam. (2023). Zero-Downtime AI Model Updates

in Real-Time Inference Systems. *International Journal of Engineering Science & Humanities, 13*(2), 70–78.

[22]. MITRE Corporation, "CWE-434: Unrestricted upload of file with dangerous type," 2023. [Online]. Available: MITRE CWE-434. Accessed: Aug. 3, 2024.

[23]. M. Müller *et al*., "Processing dangerous paths—On security and privacy of the Portable Document Format," in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2021.

[24]. N. Uddin and M. Jabr, "File upload security and validation in context of software as a service cloud model," in *Proc. 6th Int. Conf. IT Convergence and Security (ICITCS)*, 2016, pp. 1–5.

[25]. Performance Analysis of Wireless Sensor Networks for Smart Monitoring Applications. (2016). *International Journal of Humanities and Information Technology, 1*(04), 10–29. doi:10.21590/ijhit.01.04.04

[26]. A. M. Abdelrahman *et al*., "Software-defined networking security for private data center networks and clouds: Vulnerabilities, attacks, countermeasures, and solutions," *Int. J. Commun. Syst.*, vol. 34, no. 4, p. e4706, 2021. doi: 10.1002/dac.4706.

[27]. Sannidhanam, A. H. (2020). Comparative Analysis of Cloud Computing Architectures for Enterprise Applications. *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology, 12*(02), 169–180. doi:10.18090//samriddhi.v12i02.16

[28]. J. Huang *et al*., "Ufuzzer: Lightweight detection of PHP-based unrestricted file upload vulnerabilities via static-fuzzing co-analysis," *Association for Computing Machinery*, 2021.

About Author:

Abubakar Siddiq is currently pursuing a Master of Technology in Computer Science and Engineering at ISL Engineering College, affiliated with Osmania University, Hyderabad, India. He completed his Bachelor of Engineering in Information Technology from ISL Engineering College, affiliated with Osmania University, from 2018 to 2022.