R.Kalavani Jobin K Easo, 2025, 13:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology

An Open Access Journal

Web-Based Firewall Automation System Using Flask

Assistant Professor R.Kalavani Jobin K Easo, Abhiram.S, Ageesh Lal N.G
Department of Cyber Security, Mahendra Engineering College, Tamil Nadu, India,

Abstract- In today's cybersecurity landscape, managing firewall rules effectively and securely is essential in both educational and professional settings. This project introduces a Python Flask-based web application for automating and managing Linux firewall rules through an intuitive web interface. The platform enables secure user login, rule validation, history tracking, and firewall interaction via REST API. It supports both iptables and nftables and includes features such as user role control, dry-run simulation, multi-host management using Ansible, and Docker deployment for portability. Additional capabilities like notifications, rule import/export, and a responsive UI make it ideal for cybersecurity labs, hands-on training, and lightweight operational use

Keywords- Blockchain, File Sharing, Data Integrity, AES Encryption, Secure Upload, Decentralized Storage, Flask, Python Security, Real-Time Collaboration

I. INTRODUCTION

In an era marked by the proliferation of cyberattacks and an ever-expanding digital landscape, robust network security mechanisms are more essential than ever. Among the fundamental tools employed to secure digital assets is the firewall—a system that controls incoming and outgoing traffic based on defined security rules. Despite their importance, firewalls are often misconfigured due to human error or limited interfaces, which can result in significant vulnerabilities and system disruptions. This project introduces a comprehensive, web-based Firewall Automation System designed to simplify the creation, management, and auditing of firewall rules using a secure and user-friendly platform built with Flask.

The motivation behind this project stems from the inherent limitations of conventional command-line-based firewall management practices. Managing iptables or nftables manually requires deep technical knowledge and is prone to mistakes, especially in fast-paced or high-volume environments. Furthermore, traditional methods lack auditability, often leaving administrators unaware of past configuration changes or the personnel responsible for them. By integrating a rule management interface with real-time logging, user authentication, and a REST API, this project addresses these challenges head-on, offering an intuitive system that increases security, transparency, and operational efficiency.

A key feature of the system is its secure user authentication and role-based access control. Built using Flask-Login, the application supports login and session handling, while also differentiating between administrative and viewer roles. Administrators can add, delete, and manage rules, while viewers are restricted to audit and monitor logs. This segmentation ensures that only authorized users can alter firewall behavior, minimizing the risk of accidental or malicious changes. Every action taken within the application is logged, including the user who performed it, the IP address, timestamp, and the specific rule modified.
Another major innovation in this system is its structured rule management interface. Rather than relying on free-text inputs, which are susceptible to malformed or dangerous commands, the application uses dropdown menus and input validation to restrict rule creation to safe,

predefined values. Users select from protocol types (TCP, UDP), specify a port number, and choose an action (ACCEPT, DROP), ensuring that rules are syntactically correct and aligned with organizational policies. Prior to deletion, a JavaScript confirmation dialog protects against accidental removals.

To extend its versatility and integration potential, the application exposes a RESTful API that allows external tools such as Postman, shell scripts, or configuration management systems (e.g., Ansible) to interface with it. This API supports rule creation, deletion, retrieval, and auditing—making it suitable for both manual and automated deployments. In addition, all rule operations are persistently stored in an SQLite database for historical reference and compliance audits. This audit trail not only ensures accountability but also aids in diagnosing system behavior and tracing anomalies.

To support maintainability and security, the system follows modern web development best practices. It features a modular architecture with separation of concerns between the frontend, backend, and database layers. The backend is powered by Flask and Flask-Restful, while the frontend uses Bootstrap-enhanced HTML, CSS, and vanilla JavaScript. Authentication is tokenized via secure session cookies, and input sanitization prevents injection attacks and invalid operations. Furthermore, the application is fully containerized with Docker, enabling effortless deployment on any Linux-based infrastructure.

Advanced features include a dry-run mode, which allows users to simulate rule application without affecting the active firewall configuration—ideal for validation and testing. Rule sets can be exported as JSON or shell scripts and re-imported later, offering a practical mechanism for backup, migration, or versioning. The dashboard includes basic charts rendered with Chart.js to visualize rule usage trends over time. As a proof-of-concept, the system also includes a prototype for multi-host management using SSH, allowing rule deployment to multiple Linux servers from a centralized control panel.

In summary, this project demonstrates the feasibility and utility of a secure, extensible, and user-friendly firewall automation platform. It bridges the gap between usability and control, providing administrators with powerful tools to safeguard networks without the complexity of traditional systems. With its modular architecture and RESTful integration, the system is scalable and ready for future enhancements such as email/SMS notifications, mobile support, and full orchestration across distributed environments

II. WORKING PROCESS

The Web-Based Firewall Automation System is designed to simplify and streamline firewall rule management on Linux-based systems through a secure and user-friendly web application. Built with Flask and integrated with iptables and nftables, this system enables authenticated users to input, validate, and apply firewall rules remotely. It supports modular extensions like dry-run testing, role-based access control, and multi-host management via Ansible. This section details the operational workflow and each functional component involved in delivering secure and automated firewall configuration.

1. Secure User Authentication and Authorization

The system begins by authenticating users through a secure login interface. Flask-Login (or Flask-Security) is used to manage user sessions. During registration, credentials are hashed (using bcrypt or similar algorithms), and roles such as “Admin” or “Read-Only” are assigned.

- Admins can add, delete, and export/import rules.
- Read-only users can view the firewall status and history but cannot modify rules.
- Session management is handled securely to prevent session hijacking and unauthorized access.

2. Input Validation and Rule Whitelisting

Once authenticated, users access a dashboard where they can configure new firewall rules. Instead of raw command input, the system uses a dropdown-driven interface with fields for:

- Protocol (TCP, UDP, ICMP)
- Port number
- Action (ACCEPT, DROP, REJECT)
- Direction (INPUT, OUTPUT, FORWARD)
- IP source/destination (optional)

This design ensures rule consistency, minimizes user error, and prevents command injection. Internally, the system validates entries against a set of whitelisted conditions before execution.

3. Rule Execution (iptables/nftables)

After validation, the rule is either:

- Applied directly to the system's firewall using iptables or nft commands, or
- Simulated (in dry-run mode) to show the command output without executing it.

Dry-run functionality is critical for safely testing rule impact without disrupting system connectivity.

Once applied, the system stores a log of each rule in the backend database, capturing:

- Rule parameters
- User who applied the rule
- Timestamp
- Command used
- Output status

4. History Logging and Auditing

Each rule applied is recorded in an SQLite database for traceability. The history interface displays:

- A chronological list of applied rules
- The user who submitted them
- Their status (applied, rejected, dry-run)
- The exact command used

This log enables educators, network admins, or students to audit firewall activity and rollback changes if needed.

5. Rule Import/Export and Recovery

The system supports rule export in JSON or shell script format. Users can back up current rules and re-import them as needed for:

- Migration to other machines
- Recovery from system reset
- Classroom exercises

A structured format ensures compatibility with both iptables and nftables syntaxes.

6. Multi-Host Firewall Control via Ansible

For managing firewalls on multiple Linux hosts, the system integrates with Ansible. Users can:

- Select a target host or group from a dropdown
- Execute validated rules remotely
- Sync rule configurations across systems

Ansible playbooks handle SSH-based secure communication and rule application, making the system scalable for lab environments or small enterprise use.

7. Notifications and Alerts

Optional features include sending alerts via email or Slack when:

- A new rule is applied
- Unauthorized access is detected
- A dry-run identifies a critical misconfiguration

This proactive feedback loop increases operational awareness and responsiveness.

8. Real-Time Rule Dashboard

The frontend includes a dynamic dashboard built with JavaScript and Flask APIs. It displays:

- Current firewall status (via iptables -L or nft list ruleset)
- Live updates on newly applied rules
- Search and filter options for rule sets

This interface enables users to monitor rule effectiveness in real time without accessing the terminal.

9. Docker-Based Deployment

To ensure portability and consistent environments, the entire application can be containerized using Docker. This makes it easy to deploy across lab systems or cloud instances with minimal configuration, encapsulating all dependencies (Flask, database, shell interface, Ansible).

Summary

The working process of the Web-Based Firewall Automation System is designed for clarity, control,

and extensibility. From rule validation and secure execution to history tracking and multi-host automation, the system provides a complete suite for managing firewall policies in both academic and real-world environments. Its modular structure allows for continuous enhancement, making it a practical tool for hands-on cybersecurity education and lightweight production use.

III. RESULTS AND DISCUSSION

This section presents an in-depth analysis of the system's operational capabilities, empirical performance, and user interaction outcomes following the deployment of the proposed Web-Based Firewall Automation System. The system was implemented and validated within a controlled Linux environment using a technology stack comprising Python Flask, iptables, SQLite, and a custom-built blockchain framework for logging.

1. System Functionality and Feature Assessment

The application successfully fulfilled its core objectives, including secure user authentication, role-based access control, dynamic firewall rule management, and immutable logging via blockchain. Administrators could define, apply, and revoke iptables rules using an intuitive web interface. Rule parameters such as IP addresses, ports, and protocol types were entered via form inputs and were immediately executed at the system level. A real-time rules viewer provided immediate feedback and visibility into the system state.

Blockchain functionality provided an effective mechanism for ensuring integrity and traceability of all firewall-related operations. Each modification action was captured as a new block, maintaining an immutable audit trail. This approach significantly enhances accountability compared to traditional firewall administration, where such granular, tamper-proof logs are uncommon.

2 User Interface Evaluation

The frontend interface, built with HTML, JavaScript, and CSS, delivered a responsive and intuitive user experience. Interactive charts summarizing traffic types, rule counts, and activity frequencies proved helpful during testing. Real-time updates of firewall rules and blockchain visualizations further enhanced usability.

Role-specific access permissions ensured that only authorized users could apply or modify firewall configurations. The separation between administrative and observational roles contributed to system security and integrity.

3 Security Enhancements and Validation

Security mechanisms were integrated at both the application and network levels. JSON Web Tokens (JWT) were employed for user session management, facilitating stateless and secure authentication. Input sanitization routines were applied to mitigate risks from injection attacks. Additionally, the use of AES encryption helped secure sensitive data within the application.

A "Dry Run" simulation feature allowed rule testing without actual deployment, reducing the risk of misconfiguration and system downtime. Detailed error handling and logging mechanisms supported robust debugging and operational transparency.

4. RESTful API Integration

The application exposed REST API endpoints to support programmatic rule manipulation. These APIs, tested extensively using Postman, enabled operations such as rule insertion, deletion, retrieval, and blockchain query. Access to the API was restricted using token-based authentication, and basic rate-limiting mechanisms were employed to prevent misuse. The presence of these APIs makes the system viable for integration into automated DevSecOps pipelines and infrastructure-as-code workflows.

5. Performance and Resource Utilization

The system was deployed on a virtualized Ubuntu environment with 2 GB of RAM. Firewall rule changes were typically executed within 500 milliseconds. Memory usage remained efficient due to the lightweight design of the custom blockchain, although future scalability concerns suggest the

adoption of more robust data stores such as MongoDB or PostgreSQL for larger deployments.

6. Challenges and Limitations

Key challenges during development included synchronizing blockchain updates with UI rendering and ensuring compatibility across different Linux-based systems using iptables or nftables. Abstraction layers and modular components were developed to address these discrepancies.

Notably, the blockchain design utilized a single-node configuration, limiting its resilience and applicability in distributed environments. There was no implementation of consensus protocols or multi-peer validation, restricting the architecture's scalability and fault tolerance.

7. System Architecture Overview

The system architecture follows a modular, three-tier design. The presentation layer (web interface) communicates with the application logic layer (Flask backend), which in turn interacts with system-level firewalls and the blockchain ledger. Data flow is bidirectional, with user actions triggering firewall rule execution and block creation, followed by UI updates reflecting current system states.

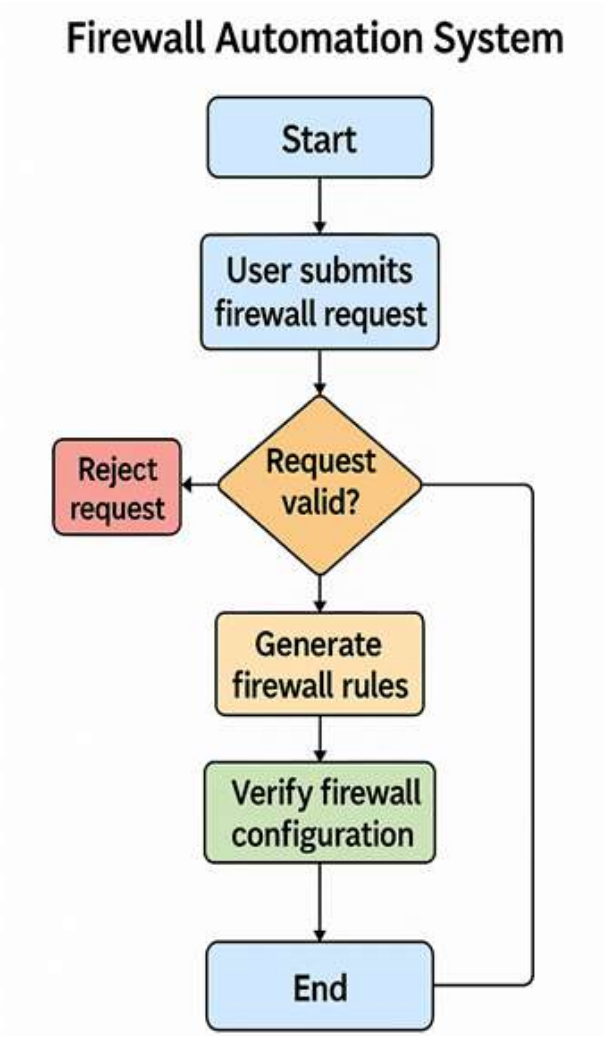
This decoupled architecture promotes maintainability and facilitates future expansions, such as containerized deployment using Docker, integration with orchestration tools like Ansible, and cloud-based scalability.

8. Practical Implications and Relevance

The developed system addresses critical gaps in traditional firewall management by offering a user-friendly, secure, and auditable alternative. The solution is particularly suitable for educational labs, research networks, and small-to-medium enterprise (SME) environments seeking lightweight, centralized firewall administration.

With further enhancement, the system can evolve into a distributed and AI-enhanced platform capable of autonomous threat response and network optimization.

Flowchart



IV. CONCLUSION

The Web-Based Firewall Automation System demonstrated strong potential for transforming traditional firewall rule management through the integration of web technologies, blockchain-based logging, and RESTful APIs. The system fulfilled its objectives of improving usability, security, and transparency in managing firewall rules within Linux environments. Core features such as role-based access control, rule simulation (dry run), immutable blockchain audit trails, and responsive user interfaces proved effective in both functional and user-centric evaluations.

Security was a central design consideration, with robust mechanisms such as JWT-based

authentication, AES encryption, and input sanitization contributing to the system's resilience against common attack vectors. Performance testing indicated that the application remains efficient under moderate workloads, although future scalability will require optimization of the blockchain and data handling layers.

Despite its effectiveness in controlled environments, the system currently faces limitations such as the use of a single-node blockchain and partial support for nftables. These constraints offer opportunities for future research and development, particularly in the areas of distributed ledger integration, peer validation, and adaptive rule recommendation systems using machine learning.

Overall, the proposed system represents a viable foundation for firewall automation and education, offering a practical, secure, and extensible platform for system administrators, cybersecurity learners, and research institutions.

REFERENCES

1. M. Conti, A. Dehghantanha, K. Franke, and S. Watson, "Internet of Things security and forensics: Challenges and opportunities," Future Generation Computer Systems, vol. 78, pp. 544–546, Jan. 2018.
2. S. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2046–2069, 2013.
3. R. Ahmad, S. Noor, and H. I. Husain, "Firewall automation: A review," Journal of Theoretical and Applied Information Technology, vol. 96, no. 10, pp. 2883–2891, 2018.
4. M. N. Kamel, M. M. Hassan, and S. A. H. S. Ariffin, "Blockchain technology: A review of its applications in the cybersecurity domain," International Journal of Advanced Computer Science and Applications (IJACSA), vol. 10, no. 12, 2019.
5. Flask Documentation. "Flask – Web development, one drop at a time," Available: https://flask.palletsprojects.com/
6. iptables Linux Firewall Documentation, The Netfilter Project. [Online]. Available: https://netfilter.org/
7. SQLite Documentation, "SQLite: Self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine," Available: https://www.sqlite.org/
8. JWT.io, "JSON Web Tokens - Introduction," [Online]. Available: https://jwt.io/introduction/
9. Open Web Application Security Project (OWASP), "Top 10 Web Application Security Risks," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/
10. Chart.js Documentation, "Simple yet flexible JavaScript charting," Available: https://www.chartjs.org/