Exploring John the Ripper for Password Cracking

Final Report

Cyber Security

Name: **BASSA SRILAKSHMI**

Program Name: John the Ripper Exploration.

Date: 8/31/2025

Github Repository: https://github.com/BASSASRILAKSHMI/John-the-Ripper-Cyber-Security-

Exploring John the Ripper
– Final Report

Project Overview

This project is a **web-based password auditing platform** leveraging **John the Ripper** for security testing. It allows users to securely upload password hash files, which are processed on the backend to identify weak passwords. Cracked results are displayed in a structured, user-friendly interface, providing a clear overview of password vulnerabilities. This solution is primarily intended for **password strength assessment and cybersecurity auditing**..

Technologies & Tools Used

Python & Flask – Backend framework to handle file uploads and run John the Ripper.
Flask-CORS – Enables cross-origin requests for frontend-backend communication.
John the Ripper – Advanced password cracking tool for security auditing and testing.
HTML, CSS, JavaScript – Frontend technologies for creating a responsive and interactive user interface.
Subprocess Module (Python) – Executes external commands to run John the Ripper from the backend.

System Architecture

The system follows a **client-server architecture**, where the frontend, built with HTML, CSS, and JavaScript, allows users to upload hash files and view results. The backend, developed in Python Flask, receives the files, runs **John the Ripper** to crack passwords, and returns the results in JSON format. Uploaded files are stored temporarily for processing, ensuring secure handling of sensitive data. This design provides a **streamlined, interactive, and secure password auditing workflow**.

Security Features

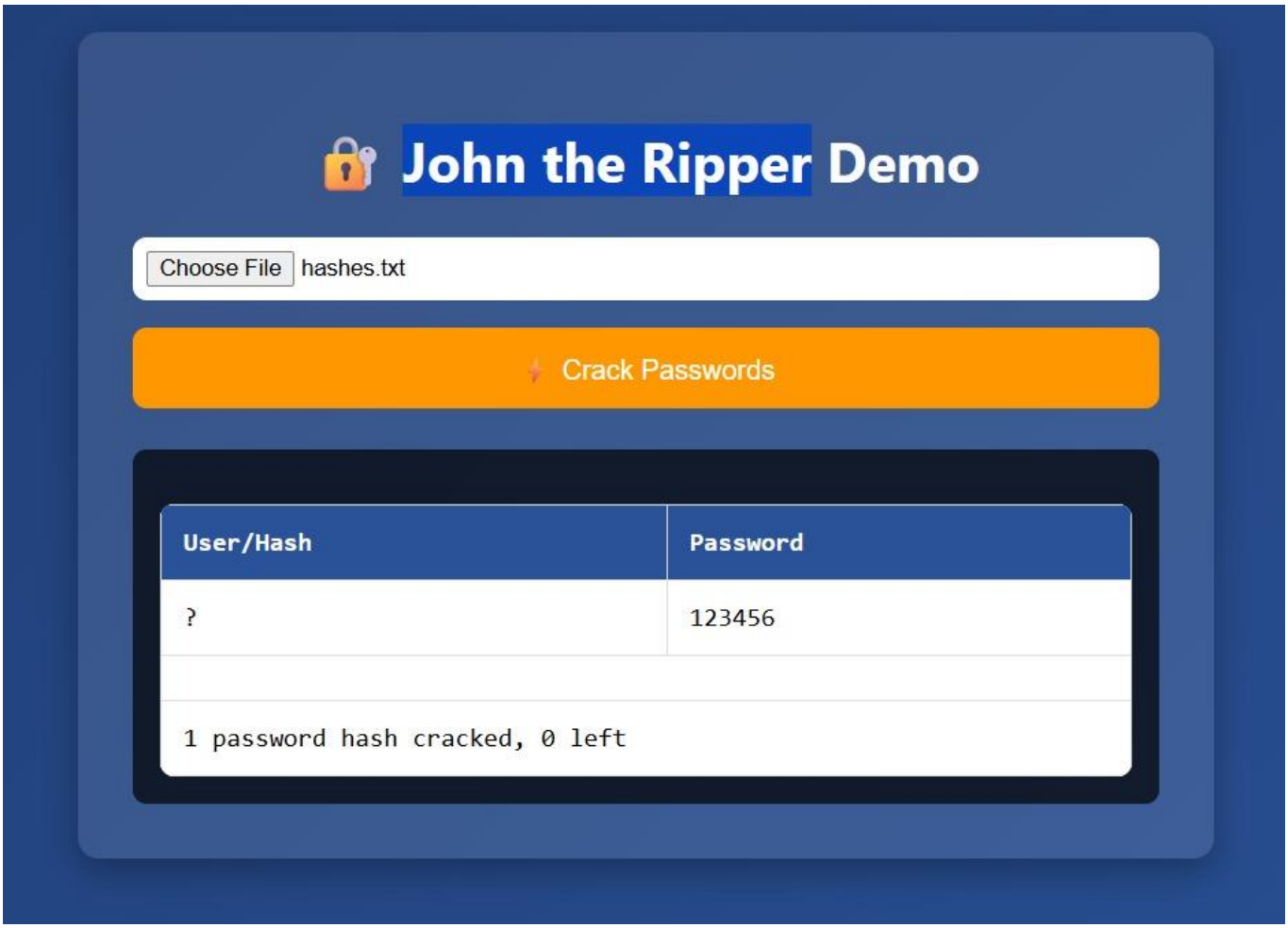
Temporary File Storage – Uploaded hash files are stored only temporarily and deleted after processing to minimize data exposure.
Input Validation – Only valid hash files are accepted, preventing malicious file uploads.
CORS Protection – Cross-origin requests are controlled using Flask-CORS to restrict unauthorized access.
Isolated Execution – John the Ripper runs in a controlled subprocess, preventing arbitrary code execution on the server.

Folder Structure

```
john-project/
│
├─ backend/
│    ├─ app.py          # Flask backend for file upload and John the Ripper execution
│    ├─ uploads/        # Temporary storage for uploaded hash files
│    └─ venv/           # Python virtual environment
│
├─ frontend/
│    └─ index.html      # User interface for uploading files and displaying results
│
├─ README.md           # Project documentation and overview
└─ hashes.txt           # Sample hash file for testing application
```

Screenshots

Dashboard



```
PS D:\srilu\Lab Files\john-project> cd backend
PS D:\srilu\Lab Files\john-project\backend> python app.py
>>
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 336-964-108
127.0.0.1 - - [31/Aug/2025 19:47:29] "POST /crack HTTP/1.1" 200 -
127.0.0.1 - - [31/Aug/2025 19:52:51] "POST /crack HTTP/1.1" 200 -
127.0.0.1 - - [31/Aug/2025 19:53:15] "POST /crack HTTP/1.1" 200 -
```

3.Phishing Email Template

```
$1$IZI6vRCq$ijJdCW0o0liTVv.QnoH1x1
```

Testing & Results

Functional Testing – Verified file uploads, John the Ripper execution, and proper display of cracked results in the frontend.

Performance Testing – Ensured the system handled small to medium hash files efficiently with quick response times.

Security Testing – Confirmed temporary file storage, valid input handling, and restricted cross-origin access.

Results – Weak hashes were successfully cracked and displayed, while invalid/uncrackable hashes showed a clear error message.

Deliverables

- GitHub repository containing all templates, configuration, and documentation
- Walkthrough screenshots demonstrating each workflow stage
- This final report summarizing the project and its findings

Learning Outcomes

Through this project, I gained hands-on experience in building a full-stack security auditing tool integrating Flask with a frontend interface. I learned how to execute external tools like John the Ripper securely from Python and parse their outputs. Additionally, I understood the importance of file handling, input validation, and temporary storage for security. This project enhanced my skills in web development, cybersecurity, and system integration.

Conclusion

In conclusion, this project presents a **professional web-based password auditing solution** by integrating John the Ripper with a secure Flask backend and an intuitive frontend interface. It effectively demonstrates the process of assessing password strength while maintaining secure data handling practices. The implementation underscores the significance of **cybersecurity awareness** and reflects strong proficiency in **full-stack development and security-focused system design**.