Software Requirements Specification (SRS)

Project: Cybersecurity Dashboard (Web Application)

Version: 1.0

Date: 2025-08-24

Prepared by: ChatGPT

---

1. Purpose

This SRS describes the functional and non-functional requirements for a professional, user-friendly, and educational **Cybersecurity Dashboard** web application. The system provides multiple security tools (hashing, encryption, brute force password cracking, scanning, vulnerability testing) packaged inside modular, card-style UI components with a neon-green, dark-themed design and a Matrix-style animated background.

The document is intended for stakeholders, developers, QA engineers, and designers involved in the project.

---

2. Scope

The Cybersecurity Dashboard will be a full-stack web application implemented using Flask (Python) for the backend and HTML/CSS/JavaScript for the frontend. It will provide:

- Practical modules for file hashing, encryption, brute force cracking, hash comparison, malware/URL scanning, and web vulnerability scanning.
- Educational content and documentation for each module.
- User authentication and session management with persistent storage.
- Aesthetic, modern UI with Matrix rain background animation that remains fixed while users scroll.

The system will be designed for local deployment and eventual hosting on a cloud VM or PaaS.

---

3. Definitions, Acronyms & Abbreviations

- **SRS** — Software Requirements Specification
- **API** — Application Programming Interface
- **UI** — User Interface
- **DB** — Database
- **AES** — Advanced Encryption Standard

- **SQLi** — SQL Injection
- **XSS** — Cross-Site Scripting
- **CSRF** — Cross-Site Request Forgery

---

4. Stakeholders

- **Project Owner / You**: Defines features and approves releases.
- **Developer(s)**: Implement frontend, backend, and integrations.
- **Designer(s)**: Create UI assets, wireframes, and visuals.
- **QA Engineer(s)**: Test functionality and security.
- **End Users**: Learners, security enthusiasts, and admins.

---

5. System Overview

High-level components:

- **Frontend**: Responsive HTML templates (Flask Jinja2), CSS (dark theme, neon green accents), JS (matrix.js, module interactions).
- **Backend**: Flask app (`app.py`) with modular blueprints for each feature.
- **Database**: SQLite (development) with option to migrate to MySQL/Postgres.
- **Uploads storage**: `uploads/` (server-side), with quotas and scanning before processing.

File Structure (canonical):

```
project/
├── app.py
├── requirements.txt
├── templates/
│   ├── index.html
│   ├── login.html
│   ├── signup.html
│   ├── about.html
│   ├── modules/
│   │   ├── hash.html
│   │   └── encrypt.html
│   └── ...
├── static/
│   ├── css/
│   │   └── style.css
│   └── js/
│       └── matrix.js
├── uploads/
├── database/
```

```
│   └── users.db
└── README.md
```

---

6. Functional Requirements

Authentication & User Management

FR-AUTH-01: User Registration — The application shall allow users to sign up with a username, email, and password.

FR-AUTH-02: Login/Logout — Users shall be able to log in and out. Sessions shall persist until logout or inactivity timeout.

FR-AUTH-03: Password Storage — Passwords shall be stored using a strong hashing algorithm (e.g., bcrypt via `werkzeug.security.generate_password_hash`).

FR-AUTH-04: User Roles — Two roles: `user` (default) and `admin` (optional). Admins can view logs and manage content.

FR-AUTH-05: Account Recovery — Optionally support password reset via email (future enhancement).

Dashboard & Navigation

FR-DASH-01: Navbar — A working navbar linking to Home, Modules, Education, About, Contact, and User Profile pages.

FR-DASH-02: Footer — Includes social media links and contact info.

FR-DASH-03: Each module is presented as a card (separate div) with title, short description, upload inputs, action buttons, and result area.

Module: Hash Calculator

FR-HASH-01: The system shall compute MD5, SHA-1, SHA-256, and SHA-512 for uploaded files and pasted text.

FR-HASH-02: The system shall allow users to download/save computed hashes (optional CSV or JSON export).

FR-HASH-03: UI shall display computed hashes in a copyable format.

Module: File Encrypt/Decrypt

FR-ENC-01: The system shall encrypt uploaded files using AES (e.g., AES-256-GCM) with a user-provided password/key.

FR-ENC-02: The system shall decrypt previously encrypted files when provided with the correct key.

FR-ENC-03: Encrypted files shall be downloadable and securely stored temporarily during processing.

FR-ENC-04: Keys/passwords shall never be stored in plaintext on the server.

Module: Password Cracker (Brute Force Unlocker)

FR-BRUTE-01: The system shall attempt brute-force password recovery for supported file types: PDF, ZIP, DOCX, PPTX, XLSX.

FR-BRUTE-02: The system shall accept user-supplied character sets, min/max lengths, and optional wordlists.

FR-BRUTE-03: The system shall run brute force jobs in background threads or tasks with progress reporting and the ability to stop/cancel.

FR-BRUTE-04: For safety and legality: the UI shall display a clear disclaimer and require the user to confirm they have legal authorization to test the file.

FR-BRUTE-05: The system shall log brute force attempts per-user (job id, start/end time, status) for auditing.

FR-BRUTE-06: Rate-limits and resource caps must be enforced to prevent abuse and DoS on the host machine.

Module: Hash Comparison

FR-COMP-01: Allow two file uploads or file+hash input to compare hashes and show match/mismatch.

FR-COMP-02: Provide a brief explanation of what a matching/different hash indicates.

Module: URL & Malware Scanner

FR-MAL-01: The system shall accept a URL for quick checks (e.g., via API lookups) and provide safety verdicts (safe/unsafe/unknown).

FR-MAL-02: The system shall scan uploaded files against a signature database (simple local signatures or integrate with freeware engines) and return the findings.

FR-MAL-03: For external lookups (e.g., VirusTotal), the system shall allow optional integration using API keys (user-provided or admin-configured).

FR-MAL-04: The system shall quarantine suspicious files and present results to the user.

Module: Vulnerability Scanner

FR-VULN-01: Allow users to input a target website or URL for non-destructive vulnerability checks (passive scans for headers, misconfigurations).

FR-VULN-02: Provide optional active checks (XSS/some SQLi tests) only after repeated explicit confirmation, and rate-limited to prevent misuse.

FR-VULN-03: The results shall include severity levels and remediation suggestions.

Education & Documentation

FR-EDU-01: Each module shall link to a short educational explanation that describes the concept, how the tool works, risks, and safe usage.

FR-EDU-02: Provide an About page covering project goals, legal/ethical guidance, and contact info.

Admin & Logging

FR-LOG-01: Maintain server-side logs for user actions (uploads, brute force jobs, scans) for auditing.

FR-LOG-02: Admin dashboard to view logs (admin-only endpoint).

---

7. Non-Functional Requirements

Security

- **NFR-SEC-01:** CSRF protection on all state-changing forms.
- **NFR-SEC-02:** Input validation and sanitization for file names, URLs, and text inputs.
- **NFR-SEC-03:** Limit file upload size (e.g., 50 MB) and allowed MIME types.
- **NFR-SEC-04:** Store passwords hashed (bcrypt) and use HTTPS in production.
- **NFR-SEC-05:** Quarantine and delete uploaded files after a configurable retention period.

Performance

- **NFR-PERF-01:** Matrix animation should not block main thread — use `requestAnimationFrame` and optimized drawing.
- **NFR-PERF-02:** Brute-force tasks should be executed in background worker threads/processes; UI should poll for progress using AJAX or WebSockets.
- **NFR-PERF-03:** Page assets should be minified and served with caching headers.

Usability

- **NFR-UX-01:** Dark theme with consistent neon green accents and clear typography.
- **NFR-UX-02:** All modules accessible from navbar; cards should be keyboard-focusable and screen-reader friendly.
- **NFR-UX-03:** Provide clear error messages and tooltips.

Compatibility

- **NFR-COMP-01:** Works in modern browsers (Chrome, Firefox, Edge). Mobile-responsive layouts.

Maintainability

- **NFR-MNT-01:** Code organized into blueprint-style modules; clear separation of concerns.
- **NFR-MNT-02:** Provide README with setup, environment variables, and deployment instructions.

Legal & Ethical

- **NFR-LEGAL-01:** Display explicit disclaimers for brute-force and vulnerability scanning features; require user confirmation.
- **NFR-LEGAL-02:** Retain minimal data and comply with privacy best practices.

---

8. UI / UX Specifications

Theme & Aesthetics

- **Colors:** Background: near-black (`#0b0b0b`), Neon Accent: `#00ff66` (or accessible alternative), muted greys for text.
- **Cards:** Rounded corners (border-radius: 12px), soft shadows (2xl style), padding `p-4` / `p-6`.
- **Buttons:** Modern, with neon green hover glow (box-shadow on hover), transitions for hover and active states.
- **Upload Inputs:** Styled custom file inputs with drag-and-drop support.
- **Matrix Background:** Canvas full-screen fixed at top of `<body>` with `z-index: -1`.

Layout

- Desktop: 3-column responsive grid for module cards; collapse to 1 column on mobile.
- Navbar sticky at top with subtle transparency.
- Footer fixed to bottom of content (not viewport) with social icons.

Accessibility

- Use ARIA labels on interactive elements.
- Ensure contrast ratio meets WCAG AA for primary text.
- Keyboard navigable controls.

Behavior & Animation

- Matrix effect: smooth falling 0/1 animated via `matrix.js` using `requestAnimationFrame`.
- Scripts loaded at bottom of `index.html` for performance.
- Background remains fixed when scrolling (use `position: fixed` for canvas).

---

9. matrix.js Specification

The `matrix.js` file must:

- Create a `<canvas>` that covers `innerWidth` x `innerHeight`.
- Use an array of column `y` positions and draw `0` or `1` characters in neon green.
- Control frame rate using `requestAnimationFrame` and a `delta` time mechanism to allow slower/faster animation speed.
- Listen for `resize` events and adjust canvas size and columns.
- Ensure drawing is efficient (clear only required area or use translucent fill to create trails).

API / Config options (in-code):

```
const matrixConfig = {
  fontSize: 18,
  speed: 50, // pixels per second or integrated into delta
  charSet: ['0','1'],
  color: '#00ff66',
  trailOpacity: 0.05
}
```

The canvas element shall be inserted at the top of `<body>` and have `z-index: -1` so clicks go through to the UI.

---

10. Data Model

Important tables (SQLite example):

users

Column	Type	Notes
id	INTEGER	Primary Key, autoincrement
username	TEXT	Unique
email	TEXT	Unique

Column	Type	Notes
password_hash	TEXT	bcrypt hash
role	TEXT	'user' or 'admin'
created_at	DATETIME	

uploads

Column	Type	Notes
id	INTEGER	PK
user_id	INTEGER	FK -> users.id
filename	TEXT	Original filename
path	TEXT	Server path
size	INTEGER	bytes
status	TEXT	'uploaded', 'scanned', 'deleted'
uploaded_at	DATETIME	

jobs (brute force / scans)

Column	Type	Notes
id	INTEGER	PK
user_id	INTEGER	FK
job_type	TEXT	'brute_force','scan'
target	TEXT	path or url
params	TEXT	JSON params
status	TEXT	'pending','running','done','failed'
result	TEXT	JSON result
started_at	DATETIME	
ended_at	DATETIME	

scan_results

Column	Type	Notes
id	INT	PK

Column	Type	Notes
job_id	INT	FK
findings	TEXT	JSON

---

11. API Endpoints (suggested)

- `GET /` — Home page (dashboard)
- `GET /login`, `POST /login` — Auth
- `GET /signup`, `POST /signup`
- `POST /upload` — Upload file
- `POST /hash` — Return file/text hashes
- `POST /encrypt` — Encrypt file
- `POST /decrypt` — Decrypt file
- `POST /brute/start` — Start brute force job
- `POST /brute/stop` — Stop job
- `GET /jobs/<id>` — Get job status
- `POST /scan/url` — URL scan
- `POST /scan/file` — File malware scan
- `GET /admin/logs` — Admin logs

All endpoints that modify server state must require CSRF tokens and user session verification.

---

12. Security Considerations

- Validate and sanitize all user inputs.
- Limit file upload size and whitelist MIME types.
- Scan uploads before processing in sensitive modules.
- Brute-force and vulnerability scans require user authorization and explicit confirmation.
- Protect admin endpoints with role checks.
- Use HTTPS in production and secure cookies.

---

13. Testing Plan

- **Unit tests** for hash generation, encryption/decryption, and parsing.
- **Integration tests** for end-to-end flows: upload → scan/encrypt → download.
- **Security tests**: test CSRF, XSS, input validation, file upload handling.
- **Performance tests**: matrix animation CPU usage, concurrent brute-force task handling.
- **User acceptance tests (UAT)**: verify UI, navigation, and educational content.

---

14. Deployment & Operations

- **Local dev:** Use virtualenv, `requirements.txt`, and SQLite.
- **Production:** Use Gunicorn + Nginx (or a PaaS). Serve static files via CDN or Nginx.
- **Environment variables:** SECRET_KEY, DATABASE_URL, API keys for external services.
- **Backups:** Schedule database backups and regular purge of `uploads/`.
- **Monitoring:** Use logs and optionally integrate external error tracking.

---

15. Milestones & Timeline (suggested)

1. **Week 1:** Project scaffolding, authentication, DB setup, basic UI skeleton.
2. **Week 2:** Hash Calculator, Hash Comparison modules, and educational content.
3. **Week 3:** File Encrypt/Decrypt module + UI polish.
4. **Week 4:** Password Cracker core (safe local implementation) with job management.
5. **Week 5:** URL & Malware Scanner (local signatures) + vulnerability scanner (passive checks).
6. **Week 6:** Admin features, logging, and final QA.
7. **Week 7:** Deployment, documentation, and UAT.

(Adjust based on team size and priorities.)

---

16. Acceptance Criteria

- All modules run without critical errors on the supported platform.
- User authentication works and user data stored securely.
- Matrix background animates smoothly while UI remains fully interactive and scrollable.
- Brute-force module includes disclaimer and enforces rate-limits; job management works (start/stop/progress).
- Educational content present and accessible from each module.

---

17. Appendix: Legal & Ethical Notice

This application includes tools that can be used for offensive security (password cracking, vulnerability scanning). The application **must not** be used for illegal activity. Users are responsible for ensuring they have explicit authorization to test or attempt password recovery on any file or target. The system shall present clear disclaimers and require the user to acknowledge them before using such functionality.

End of SRS v1.0