Information Security Project Proposal

Threat Guard

Session: 2023 – 2027

Submitted by:

Mian Saad Tahir 2023-CS-62

Muhammad Talha 2023-CS-70

Supervised by:

Dr. Ayesha Altaf

Course:

CSC-201L Information Security Lab

Department of Computer Science

University of Engineering and Technology

Lahore Pakistan

Table of Contents

Table of Figures

Project Title and Introduction:

Threat Guard:

Threat Guard is an advanced cybersecurity tool designed to detect and mitigate online threats. It provides a comprehensive threat analysis and protection system, combining phishing detection, malware scanning, DDoS prevention, and data breach monitoring into a single, user-friendly platform.

Importance:

With the rise in cyber threats such as phishing attacks, malware distribution, and credential leaks, organizations and individuals are at constant risk of data theft and financial loss. Threat Guard helps mitigate these risks by:

- Detecting malicious URLs used for phishing attacks.
- Scanning uploaded files for malware.
- Preventing DDoS attacks by blocking suspicious traffic.
- Checking if emails and passwords have been compromised in past data breaches.

Users of Threat Guard:

- Organizations looking to secure their networks.
- Individuals concerned about their cybersecurity.
- Cybersecurity analysts for threat intelligence.
- Developers integrating security measures in their applications.

Problem Statement:

Cybercriminals are constantly evolving their tactics, making it difficult for individuals and organizations to stay protected from various online threats. Phishing attacks, malware-infected files, DDoS attacks, and data breaches are among the most prevalent security risks. Traditional security solutions often address these issues separately, leaving gaps in protection.

Threat Guard is designed to provide a comprehensive cybersecurity solution by integrating four essential security features into a single platform:

- **Phishing Detection** – Identifies and flags malicious URLs that attempt to steal sensitive information.
- **File Scanning** – Analyzes uploaded files for malware threats before they can cause harm.
- **DDoS Protection** – Implements rate limiting and IP blocking to prevent denial-of-service attacks.
- **Pawned Credentials Checker** – Verifies if an email or password has been exposed in data breaches.

Objectives and Scope:

Objectives:

- Provide a unified platform for detecting multiple cybersecurity threats.
- Identify phishing URLs using keyword analysis and VirusTotal API.
- Scan files for malware using VirusTotal API and known malicious hashes.
- Implement rate limiting and IP blocking to mitigate DDoS attacks.
- Check if emails and passwords have been compromised in data breaches.

Scope:

- Designed for individuals, businesses, and cybersecurity professionals.
- Uses APIs, database storage, and real-time analysis for accuracy.
- Focuses on detection but does not actively prevent or remove threats.
- Limited to web-based scanning without modifying user systems.

Features and Functionalities:

1. Phishing Detection:

Feature: Detects potentially harmful phishing URLs.
Components:

- phishing_checker.html → UI for URL input and displaying results.
- utils.py → Contains check_url_virustotal() function for checking URLs via VirusTotal API.
- PhishingURL table in phishing_urls.db stores detected URLs and their status.

Functionality:

- Users enter a URL, and the system analyzes it using keywords, HTTPS validation, and VirusTotal API.
- If phishing is detected, it stores the URL in the database and marks it as Phishing.
- Prevents duplicate entries.

2. File Scanning:

Feature: Detects malicious files using VirusTotal API.
Components:

- file_scan.html → UI for uploading files and displaying scan results.
- utils.py → Contains check_file_virustotal() function for scanning files.
- List of known malicious hashes in malicious_hashes set for local detection.

Functionality:

- Users upload a file, and its SHA-256 hash is checked against VirusTotal API and a predefined malicious hash list.
- If a match is found, a warning is displayed.

3. DDoS Protection:

Feature: Implements rate-limiting and IP blocking to mitigate DDoS attacks.
Components:

- app.py → Flask Limiter is used for rate-limiting requests (e.g., 5 per minute).
- blocked_ips.db → Stores blocked IPs when excessive requests are detected.
- admin.html → Admin panel to view and unblock IPs.
- too_many_requests.html → UI shown when a user is rate-limited.

Functionality:

- If an IP exceeds the request limit, it is automatically blocked and stored in blocked_ips.db.
- The admin panel allows manual unblocking of IPs.

4. Pawned Email & Password Checker:

Feature: Checks if an email or password has been compromised in data breaches.
Components:

- pawned_checker.html → UI for entering email/password and displaying results.
- pawned_check.py → Contains check_password_pawned() function using HIBP API.
- pawned_data.db → Stores previously checked passwords/emails (optional).

Functionality:

- Users enter an email or password, and it is checked against the Have I Been Pwned (HIBP) API.
- If compromised, it shows how many times the email/password has been leaked.

5. User Interface & Navigation:

Components:

- index.html → Homepage with links to all tools.
- home.html → Main dashboard.
- Buttons linking to Phishing Checker, File Scanner, DDoS Test, and Pawned Checker.

6. Database Management:

Components:

- phishing_urls.db → Stores detected phishing URLs.
- blocked_ips.db → Stores blocked IPs for DDoS protection.
- pawned_data.db → (Optional) Stores previous pawned email/password checks.

Functionality:

- Uses SQLAlchemy ORM to interact with SQLite databases.
- Prevents duplicate entries and maintains historical records.

Technology Stack:

Threat Guard is built using modern technologies to ensure efficiency, scalability, and security.

Backend:

- Flask → Lightweight web framework for handling requests and responses.
- SQLAlchemy → ORM for managing SQLite databases.
- Flask-Limiter → Implements rate-limiting for DDoS protection.
- VirusTotal API → Used for phishing URL and file scanning.
- Have I Been Pwned (HIBP) API → Checks for compromised emails and passwords.

Frontend:

- HTML, CSS → Basic UI for interacting with the system.
- Jinja2 → Flask templating engine for dynamic content rendering.

Database:

- SQLite → Stores phishing URLs, blocked IPs, and pawned data.

System Architecture:

Threat Guard follows a modular architecture with a clear separation of concerns:

Client-Side (UI):

Users interact with a web-based UI to check URLs, scan files, test DDoS protection, and verify pawned credentials.

Backend (Flask Server):

- Handles user requests, processes data, and communicates with APIs (VirusTotal, HIBP).
- Implements rate-limiting and database operations for phishing detection, file scanning, and pawned checks.

Database Layer:

- Stores detected phishing URLs, blocked IPs, and compromised credentials.
- Uses SQLAlchemy ORM to manage SQLite databases efficiently.

Third-Party API Integration:

- VirusTotal API → Scans URLs and files for malicious threats.
- HIBP API → Checks passwords and emails against breached data sources.

User Interface Designs:

Figure 1-Main Interactive Screen

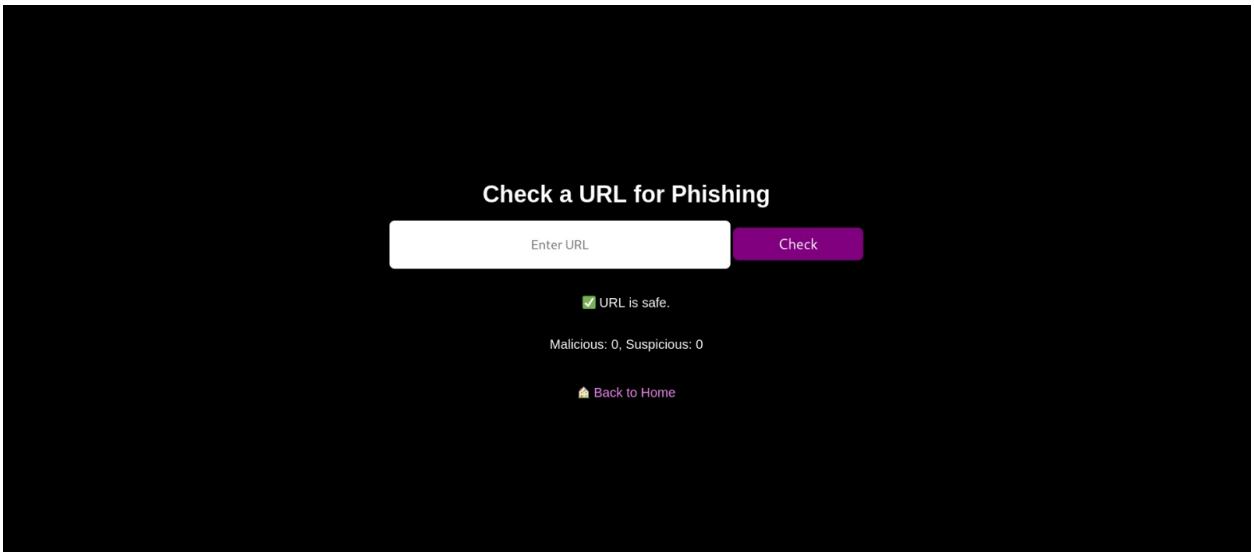


Figure 2-Phishing URL Checker

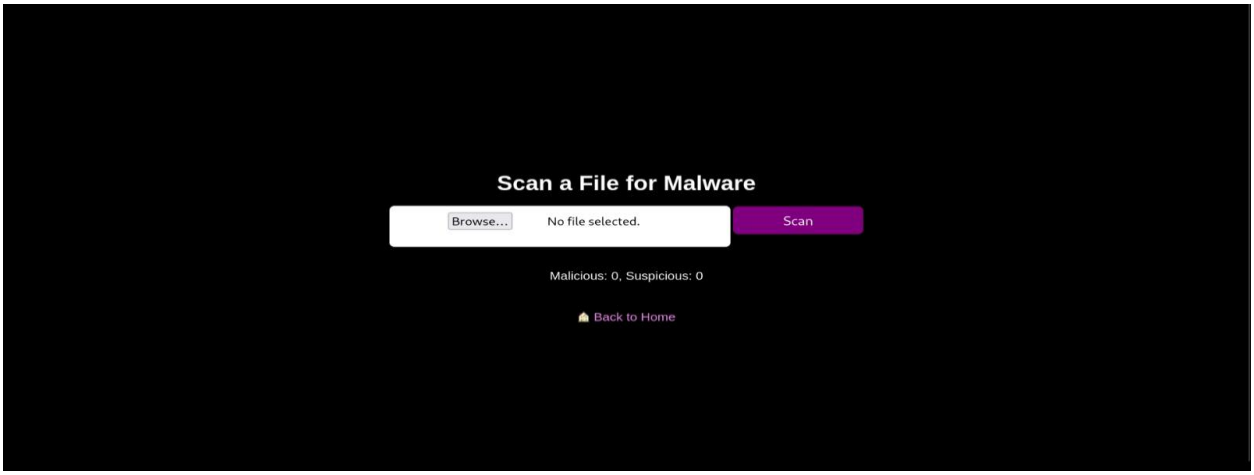
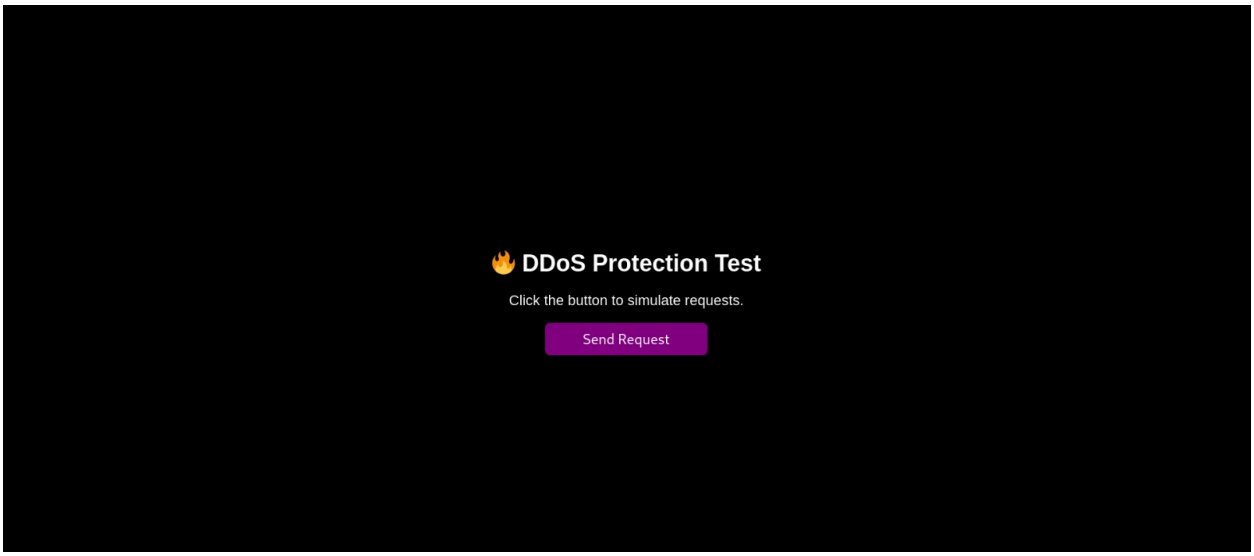


Figure 3-File Scanning



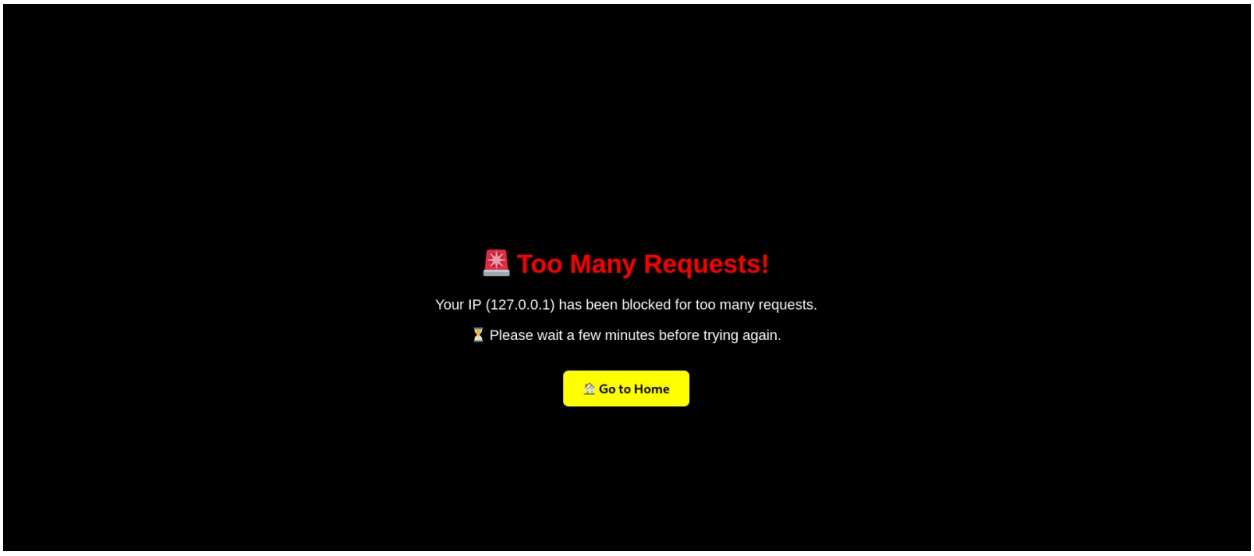


Figure 4-DDoS Protection

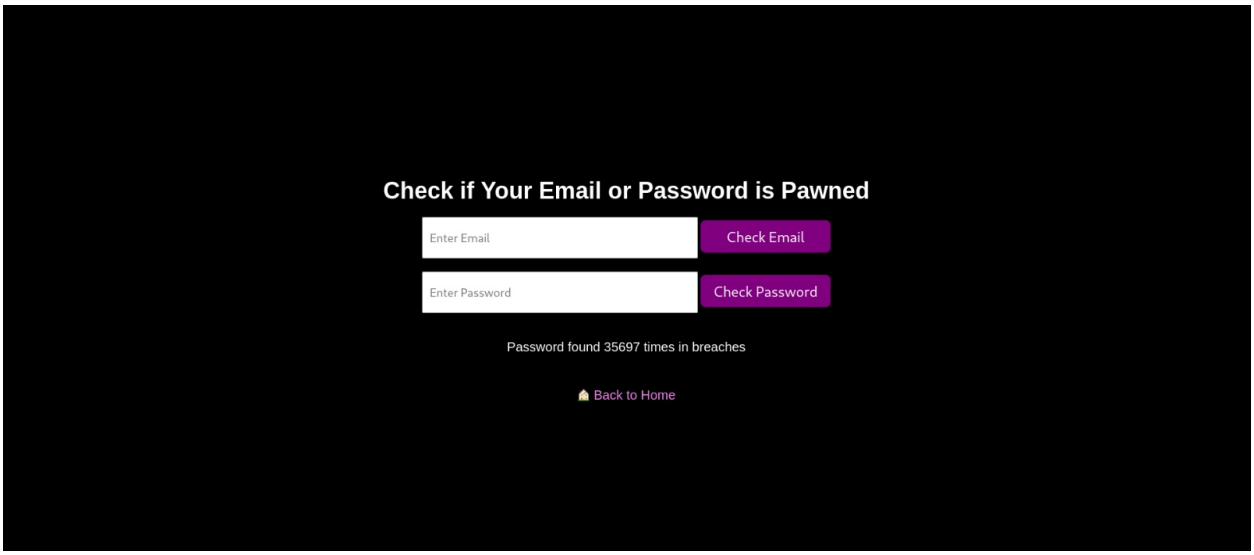


Figure 5-Pawned Checker

Implementation Details (Code):

1.Phishing URL Detection:

- Users enter a URL, which is checked for phishing characteristics.
- Local validation: Checks keywords and HTTPS usage.
- VirusTotal API: Verifies if the URL is flagged as malicious.
- Database storage: New URLs are stored with their status.

Code Snippet (Phishing URL Check in app.py)

```
@app.route("/phishing_checker", methods=["GET", "POST"])
@limiter.limit("5 per minute")
def phishing_checker():
    result, vt_result = None, None

    if request.method == "POST":
        url = request.form.get("url")
        if url:
            result = check_url(url)
            vt_result = check_url_virustotal(url)

            existing_entry = PhishingURL.query.filter_by(url=url).first()
            if not existing_entry:
                status = "Phishing" if "⚠️" in result else "Safe"
```

```
            new_entry = PhishingURL(url=url, status=status)
            db.session.add(new_entry)
            db.session.commit()
        else:
            flash("This URL is already in the database.", "info")

    return render_template("phishing_checker.html", result=result, vt_result=vt_result)
```

2.File Scanner:

- Users upload a file, which is scanned using:
- SHA-256 hash comparison (against known malicious hashes).
- VirusTotal API (for a detailed scan).

Code Snippet (File Check in utils.py)

```
def check_file_hash(file):
    file_hash = hashlib.sha256(file.read()).hexdigest()
    if file_hash in malicious_hashes:
        return "⚠️ Warning: Malicious file detected!"
    return "☑ File is safe."
```

3.DDoS Protection:

- Implements rate-limiting (5 requests per minute).
- Automatically blocks frequent attackers by logging their IPs.

Code Snippet (Blocking IPs in app.py)

```
@app.errorhandler(429)
def too_many_requests(e):
    ip = get_remote_address()
    if not BlockedIP.query.filter_by(ip_address=ip).first():
        db.session.add(BlockedIP(ip_address=ip))
        db.session.commit()
    return render_template("too_many_requests.html", ip=ip), 429
```

4.Pawned Credentials Checker:

- Uses the "Have I Been Pwned" (HIBP) API to check if an email or password has been breached.
- Stores results in the database for tracking.

Code Snippet (Pawned Check in pawned_check.py)

```
import requests


def check_pawned_email(email):
    api_url = f"https://haveibeenpwned.com/api/v3/breachedaccount/{email}"
    headers = {"hibp-api-key": "your_api_key"}


    response = requests.get(api_url, headers=headers)
    if response.status_code == 200:
        return "⚠️ Warning: Email has been found in data breaches!"
    return "☑ Email is safe."
```

5.Admin Panel:

- Provides an interface to view and unblock blocked IPs.
- Uses Flask to display stored data from the database.

Code Snippet (Admin Panel in app.py)

```
@app.route("/admin")
def admin():
    return render_template("admin.html", blocked_ips=BlockedIP.query.all())


@app.route("/unblock/<ip>")
def unblock(ip):
    blocked_ip = BlockedIP.query.filter_by(ip_address=ip).first()
    if blocked_ip:
        db.session.delete(blocked_ip)
        db.session.commit()
    return redirect(url_for("admin"))
```

Database View:

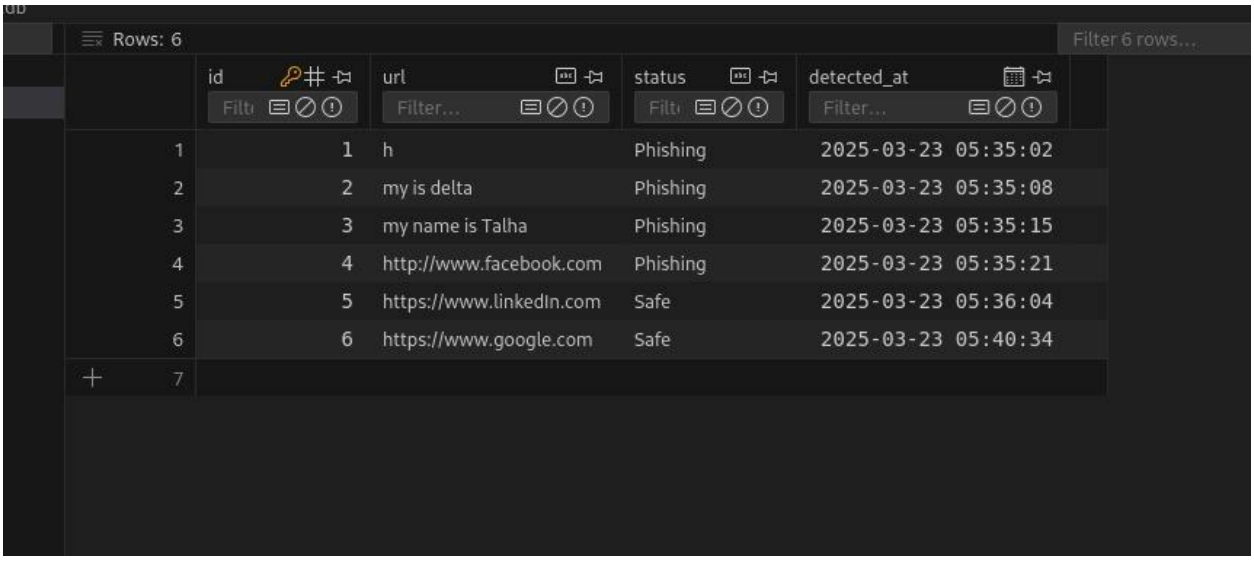


Figure 6-Phishing Urls

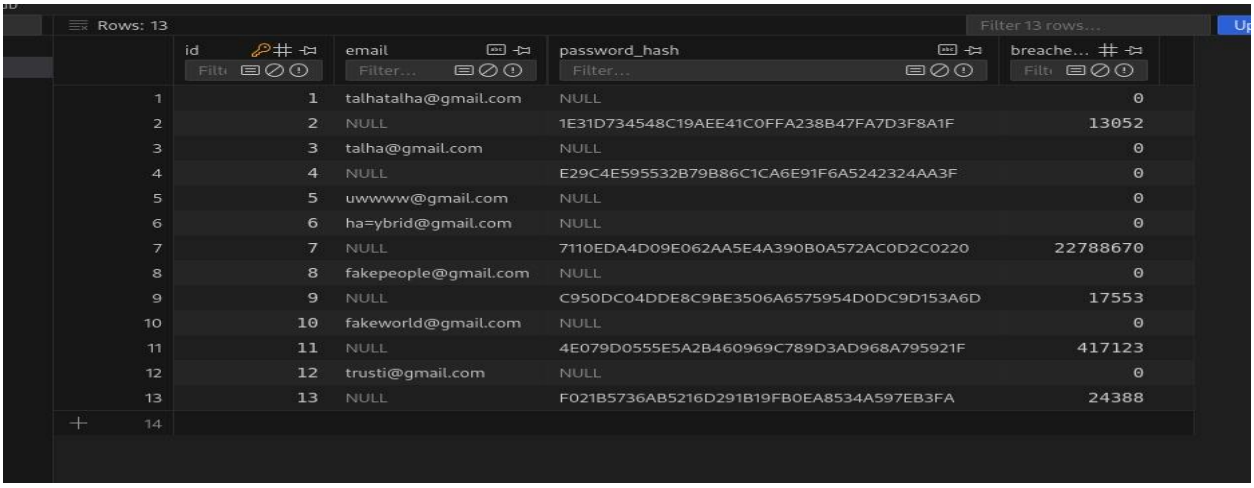


Figure 7-Pawned Emails and Passwords

Figure 8-Blocked Ips

Weaknesses:

- Limited Phishing Detection – Relies on keyword matching and VirusTotal API, which may miss advanced phishing techniques.
- File Scanner Limitation – Only checks file hashes; does not analyze file behavior or embedded malware.
- DDoS Protection Scope – Only blocks based on rate-limiting; does not implement advanced AI-based traffic analysis.
- Dependency on External APIs – Pawned check and VirusTotal rely on third-party APIs, which may have rate limits or downtime.
- No User Authentication – Lacks an admin authentication system to restrict access to sensitive features.

Future Enhancements:

- AI-Powered Phishing Detection – Implement machine learning to detect phishing patterns more accurately.
- Behavior-Based File Scanning – Use sandboxing techniques to analyze file behavior instead of just hash comparison.
- Advanced DDoS Mitigation – Integrate with Cloudflare or other security services for real-time attack prevention.
- Offline Pawned Data Checks – Store and update breach databases locally to reduce API dependence.
- User Authentication & Role Management – Add a login system for admin control and restricted access to features.