**1. Introduction**

Phishing attacks pose a significant cybersecurity threat by tricking users into revealing sensitive information such as credentials and financial details. This tool aims to prevent phishing by maintaining a central repository of Indicators of Compromise (IOCs), blocking malicious domains/IPs, and providing real-time alerts to users.

**2. Features**

**2.1 Core Functionalities**

* **IOC Repository:** Maintain a central list of malicious domains and IP addresses.
* **Real-Time Blocking:** Prevent connections to phishing domains at the browser or network level.
* **Threat Intelligence Feeds:** Auto-update the blacklist with external threat feeds.
* **AI-Based Analysis:** Detect lookalike domains and malicious content.
* **User Alerts:** Notify users when they attempt to access a phishing website.
* **SIEM Integration:** Send logs to security tools (Splunk, ELK, etc.).

**2.2 Additional Features**

* **Crowdsourced Reporting:** Allow users to submit suspicious domains/IPs.
* **Dashboard & Analytics:** View phishing attempts, blocked sites, and trends.
* **Mobile App Integration:** Extend protection to Android/iOS devices.

**3. System Architecture**

**3.1 Components**

* **Backend:** API to store and fetch IOCs.
* **Database:** Central repository (PostgreSQL, MongoDB, or SQLite).
* **Frontend:** UI for managing blocked domains/IPs.
* **Browser Extension:** Blocks phishing domains in real-time.
* **AI Engine:** Detects phishing patterns.

**3.2 Technologies Used**

* **Backend:** Django, Flask, or FastAPI
* **Database:** PostgreSQL / MongoDB
* **Frontend:** React / Vue.js
* **Browser Extension:** JavaScript (Chrome API)
* **Threat Intelligence APIs:** OpenPhish, PhishTank
* **Machine Learning:** TensorFlow / Scikit-Learn

**4. Step-by-Step Implementation**

**4.1 Setting Up the Backend**

1. **Initialize Django or Flask**
2. **Define the IOC Model:** Store domains, IPs, timestamps, and threat sources.
3. **Create API Endpoints:**
   * Add new IOCs
   * Retrieve IOCs
   * Delete IOCs
4. **Secure the API:** Use JWT authentication.
5. **Deploy on AWS/GCP/Azure.**

**4.2 Building the Browser Extension**

1. **Create a Manifest File (manifest.json)**
2. **Develop background.js to Block Requests**
3. **Store and Sync Blocked Domains/IPs**
4. **Create popup.html to Show Blocked Requests**
5. **Handle User-Added Domains**
6. **Test on Chrome and Firefox**

**4.3 AI-Based Phishing Detection**

1. **Train an ML Model:**
   * Collect phishing and legitimate domains.
   * Use NLP to analyze website text.
   * Implement Levenshtein distance to detect similar domains.
2. **Deploy AI as a REST API.**
3. **Integrate with the Browser Extension.**

**4.4 Real-Time Threat Intelligence Integration**

1. **Fetch Updates from APIs:** Abuse.ch, OpenPhish.
2. **Validate and Store New IOCs.**
3. **Schedule Automatic Updates.**

**4.5 User Alerts and Reporting**

1. **Display Alerts on Phishing Attempts.**
2. **Allow Users to Report New Threats.**
3. **Log and Review Reports in Dashboard.**

**5. Deployment & Maintenance**

**5.1 Deploying Components**

* **Backend:** AWS EC2, Heroku, or DigitalOcean
* **Database:** Cloud-managed PostgreSQL
* **Frontend:** Vercel / Netlify
* **Browser Extension:** Chrome Web Store

**5.2 Ongoing Maintenance**

* Regularly update IOCs.
* Improve AI detection accuracy.
* Monitor false positives and adjust blocking logic.
* Respond to user-submitted threats.

**6. Future Enhancements**

* **DNS Sinkhole Implementation.**
* **SIEM and Firewall Integrations.**
* **Mobile App Support.**
* **Auto-Generated Threat Reports.**

**7. Conclusion**

This phishing prevention tool will significantly reduce phishing risks by blocking malicious websites in real-time and providing users with a secure browsing experience. Continuous updates, AI enhancements, and user feedback will further improve the effectiveness of the tool.