Anjuman -I- Islam's

M. H. Saboo Siddik College of Engineering, Byculla, Mumbai-400008.

Department of Internet of Things (IoTCSBCT)

University of Mumbai

A Special Book Report on

"PhishEye - An OSINT-Based Real-Time Scam and Phishing Website Detection Framework"

Submitted in the Partial Fulfillment of the Requirements for

Open-Source Intelligence in the Fourth Year of CSE IoT.

Supervisor

Er. Pragati Pal

2025 - 2026

Anjuman -I- Islam's

M. H. Saboo Siddik College of Engineering

University of Mumbai

Department of Internet of Things (IoT)

CERTIFICATE

This is to Certify that the Report Entitled '**PhishEye - An OSINT-Based Real-Time Scam and Phishing Website Detection Framework'** is Submitted By

Name	Roll No.	Signature
Khan Humayun Majid	232860	
Ansari Zoha Najmul Kalam	232856	
Ansari Adnan Mohd Irfan	232865	

The bonafide students of **M. H. Saboo Siddik College of Engineering**, Mumbai, and they have successfully prepared and presented a book report on '**PhishEye - An OSINT-Based Real-Time Scam and Phishing Website Detection Framework'** in the partial fulfillment of the requirements for the **BE** course in **CSE IOT** at the **University of Mumbai** for the subject **Open-Source Intelligence** during the academic year **2025-2026.**

Subject in charge:	**Head of the Department:**	**Date:**
Er. Pragati Pal	Er. Samana Jafri	August, 2025

MINI PROJECT APPROVAL

This Mini-Project entitled **PhishEye - An OSINT-Based Real-Time Scam and Phishing Website Detection Framework** by **Khan Humayun Majid, Ansari Zoha Najmul Kalam, Ansari Adnan Mohd Irfan** is approved for the degree of **"Bachelor of Engineering"** in **"Computer Science Engineering (IOTCSBCT)".**

EXAMINERS

_____

INTERNAL EXAMINER

Date: ____/____/_____

Place: _____

PhishEye - An OSINT-Based Real-Time Scam and Phishing Website Detection Framework

Prepared by

Name	Roll No.	Signature
Khan Humayun Majid	232860	
Ansari Zoha Najmul Kalam	232856	
Ansari Adnan Mohd Irfan	232865	

Anjuman-I-Islam's

M.H. Saboo Siddik College of Engineering

University of Mumbai

Department of Internet of Things (IoTCSBCT)

August 2025

ACKNOWLEDGEMENT

It is with immense gratitude that we take this opportunity to acknowledge and express our heartfelt thanks to all those who contributed to the success of this project. First and foremost, we extend our sincere appreciation to our Subject Incharge, Er. Pragati Pal, for her invaluable guidance and deep expertise. Her unwavering support and availability whenever needed played a really crucial role in helping us complete this project successfully. We are also deeply thankful to the staff of the CSE IoT Department, whose kindness, encouragement, and support were instrumental throughout the project's duration. Their collective assistance made this journey smoother and more productive. Finally, we would like to express our profound gratitude to the Almighty, whose blessings have continuously guided and inspired us through this challenging endeavor.

ABSTRACT

PhishEye is a sci-fi–inspired **URL security scanner** designed to detect phishing and scam websites through real-time risk assessment. Developed using **React, TypeScript, Tailwind CSS, and Arwes**, with **GSAP animations** for a cinematic interface, it provides both security and an engaging user experience. The system evaluates URLs using OSINT-based parameters such as domain age, DNS records, SSL/TLS configuration, redirect chains, reputation lists (URLHaus, Google Safe Browsing, VirusTotal), and brand look-alike detection. These signals are combined through a modular **100-point heuristic scoring engine**, producing **LOW, MEDIUM, or HIGH** risk classifications.

The results are displayed through an animated **risk meter, evidence cards** with supporting details, **a redirect graph**, and a **verification timeline**. Additional features include **scan history, report export (PDF/JSON)**, and **settings for integrating API keys**, making the tool suitable for both individual users and security professionals. With its **privacy-first architecture**, PhishEye operates client-side by default using deterministic mock responses, while allowing users to unlock advanced detection with their own API keys.

Initial testing confirms that PhishEye offers **fast**, **interpretable**, and **user-friendly threat analysis**. The framework demonstrates strong potential for cybersecurity awareness, incident triage, and educational use. Future enhancements will include **machine learning–driven phishing detection**, a **browser extension for proactive alerts**, **bulk scanning support**, and **integration with broader threat intelligence platforms**, establishing PhishEye as an open, extensible, and practical solution for combating phishing and online scams.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

Abbreviation	Full Form
OSINT	Open-Source Intelligence
DFD	Data Flow Diagram
UI	User Interface
API	Application Programming Interface
DNS	Domain Name System
SSL	Secure Sockets Layer
TLS	Transport Layer Security
RDAP	Registration Data Access Protocol
ML	Machine Learning
JSON	JavaScript Object Notation
PDF	Portable Document Format
IDE	Integrated Development Environment
HSTS	HTTP Strict Transport Security
CSP	Content Security Policy
CT	Certificate Transparency (Logs)
DoH	DNS over HTTPS
VT	VirusTotal

PWA	Progressive Web Application
RBAC	Role-Based Access Control
CSV	Comma-Separated Values
DB	Database
JWT	JSON Web Token
UI/UX	User Interface / User Experience

LIST OF FIGURES

Chapter 1: Introduction

1.1 Background

Phishing and scam websites have become one of the most pervasive threats on the internet, targeting individuals, corporations, and even government institutions. These malicious sites mimic trusted platforms to deceive users into revealing sensitive information such as passwords, banking credentials, and personal data. According to recent cybersecurity reports, phishing attacks account for more than 90% of data breaches worldwide. Traditional detection methods often rely on blacklists or centralized security services, which cannot keep pace with the rapidly evolving nature of phishing websites.

In this context, **Open-Source Intelligence (OSINT)** has emerged as a powerful approach to detect, track, and analyze malicious URLs using publicly available information such as domain records, SSL/TLS certificates, DNS data, and community-driven reputation services.

1.2 Motivation

The motivation behind this project stems from the increasing need for **real-time, user-friendly phishing detection tools**. Existing systems such as VirusTotal or Google Safe Browsing are powerful but either require paid access for large-scale use or lack interactive and educational features.

For students, researchers, and security analysts, a lightweight but **visually rich OSINT-based scanner** can serve both as a protective tool and as an educational platform to understand how phishing detection works. Our project — **PhishEye: An OSINT-Based Real-Time Scam and Phishing Website Detection Framework** — is designed to address this gap by combining **OSINT techniques** with a **cinematic sci-fi interface** for a more intuitive and engaging experience.

1.3 Problem Statement

The primary problem this project addresses is:

“How can we design an open-source, OSINT-powered web application that can analyze a given URL in real-time, assess its legitimacy using multiple security parameters, and present the results in an engaging, user-friendly manner?”

To address the dynamic nature of phishing and scam websites, PhishEye couples a TypeScript-based React client with an Express.js (TypeScript) API that delegates OSINT collection to Python services. This full-stack design enables real-time, multi-source lookups, risk scoring, and transparent evidence presentation, while remaining privacy-preserving for end users.

1.4 Scope of the System

1.4.1 Problem Definition

Phishing websites are growing at an unprecedented rate, making blacklist-only approaches insufficient. There is a need for a **scalable framework** that integrates multiple open data sources (WHOIS, SSL records, reputation feeds) into a **single platform**.

1.4.2 Proposed Work

The proposed system — **PhishEye** — accepts a URL, performs checks across OSINT sources, and applies a heuristic scoring algorithm. Results are then visualized using **risk meters, evidence cards, graphs, and timelines**.

1.4.3 Proposed Methodology

- Step 1: User inputs URL.
- Step 2: System queries OSINT sources (WHOIS, DNS, SSL, reputation lists).
- Step 3: Heuristic algorithm assigns scores.
- Step 4: UI renders **risk score, supporting evidence, and interactive visualizations**.

1: System Overview Diagram

1.5 Objective and Scope

1.5.1 Scope

This project delivers a full-stack web application with 11 functional pages: Dashboard, Analyzer, Report, History, Settings, Documentation, Privacy Policy, Terms of Service, Cookie Policy, and Disclaimer. The frontend is built with React + TypeScript (Vite), shadcn/ui, and Framer Motion for an accessible, responsive UX. The backend comprises an Express.js (TypeScript) API, which orchestrates Python services for OSINT lookups and applies a weighted risk-scoring algorithm. The MVP emphasizes privacy (local-only storage by default), explainability (evidence cards and raw data), and practical reporting (PDF export). No machine learning is included in v1.0.0; it is planned for future releases.

1.5.2 Objectives

1. Build a full-stack, privacy-preserving OSINT web application for real-time URL risk assessment.

2. Aggregate results from multiple OSINT sources (WHOIS/RDAP, Google Safe Browsing, VirusTotal, URLScan, CT logs, DoH DNS).

3. Provide transparent, explainable outputs with a weighted risk-scoring algorithm and evidence cards.

4. Support both single-URL and bulk analysis (up to 50 URLs) with progress indication.

5. Offer professional PDF exports, history management, and a monitoring UI.

6. Ship an accessible, responsive UI with dark/light themes and smooth animations.

7. Store analysis data locally in the browser with optional local encryption.

1.6 Organization of the Report

This report is structured into six chapters:

- **Chapter 1** introduces the background, motivation, problem statement, objectives, and scope.
- **Chapter 2** provides a literature survey of existing systems and related research.
- **Chapter 3** discusses the system architecture, algorithms, diagrams, and implementation.
- **Chapter 4** presents snapshots of the user interface and functionalities.
- **Chapter 5** outlines limitations and possible future enhancements.
- **Chapter 6** concludes the report with final remarks.

1.7 Advantages

- Real-time OSINT aggregation across multiple reputable sources
- Transparent, explainable scoring with evidence cards and raw JSON
- Bulk analysis of up to 50 URLs with progress tracking
- Professional PDF report export for audits and submissions
- Local-only storage by default; optional local encryption
- Accessible, responsive UI with dark/light themes and animations
- Monitoring page with live UI (simulated feeds in MVP)

- Comprehensive documentation and legal/compliance pages

1.8 Disadvantages

- Dependent on availability and rate limits of third-party OSINT services
- Monitoring feed uses simulated events in MVP (no continuous streaming ingest)
- No ML-based classification in v1.0.0 (planned)
- Certain advanced checks (e.g., deep content analysis, headless rendering at scale) are out of scope for the MVP

Chapter 2: Literature Survey

2.1 Survey of Existing Systems

Several tools and platforms currently exist for phishing and scam detection. These systems rely on blacklists, heuristic analysis, or reputation checks to identify malicious websites.

- **Google Safe Browsing**: Used widely in browsers like Chrome and Firefox, it maintains a constantly updated blacklist of known phishing and malware sites. While highly effective, it primarily works at the browser level and is not customizable by end-users.
- **VirusTotal**: An aggregation service that uses more than 70 antivirus engines and URL/domain reputation services. It is powerful for batch URL scanning but has limitations in free usage and requires an internet connection to cloud APIs.
- **PhishTank**: A community-driven platform where users submit suspected phishing URLs. It is useful for crowdsourced intelligence but limited by user activity and lacks interactive visualization.
- **URLHaus**: Maintained by abuse.ch, it provides free malware and phishing URL feeds. It is excellent for integrating into automated security tools but not designed as an interactive user interface for general users.

These systems are effective but either **lack engaging user interfaces, educational visualization features, or are locked behind premium APIs**.

System	Features	Limitations
Google Safe Browsing	• Real-time URL checks Malware & phishing detection API for developers • Multiple AV engines File & URL analysis	• Potential false positives • Scan limits for free tier Dependent on limitations
PhishTank	• Multiple AV engines File & URL analysis	• Open-source on engines API access
URLHaus	• Community-based Verified phishing reports	• Specifc to malwaire Less phishing coverage

2: Comparison Table Diagram

2.2 Previous Systems

Past projects and academic systems have also attempted phishing detection using OSINT-based or machine learning methods:

- **Heuristic-based Systems**: Earlier solutions focused on domain age, WHOIS data, SSL certificate checks, and URL length as indicators of phishing. While useful, they often lacked correlation between multiple signals.
- **Machine Learning–Based Systems**: Some research applied classifiers such as Random Forests, Decision Trees, and Neural Networks using features extracted from URLs. While effective in lab conditions, they often require large labeled datasets, which are not always practical.
- **Browser Extensions**: Many proof-of-concept extensions have been built to warn users about suspicious links. However, these are often not maintained or rely on limited data sources.

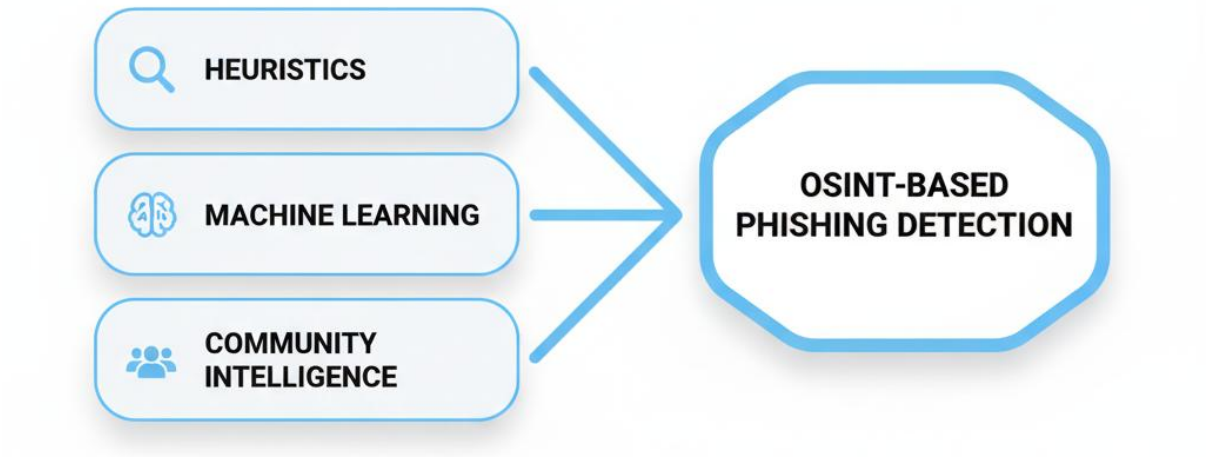
Our system builds on these ideas by combining **heuristics + OSINT feeds + cinematic UI visualization**, focusing on **real-time, user-friendly output**.

2.3 Research Papers

Several research works provide the foundation for PhishEye's methodology:

- **"A State-of-the-Art Review on Phishing Website Detection Techniques**
  - A comprehensive and recent review that categorizes phishing detection techniques—including heuristic, graph-based, LLM-based, and phishing kit-based methods. It provides a new taxonomy and highlights the strengths and limitations of each approach, helping to inform the selection of OSINT-driven heuristics for PhishEye.
  - DOI: 10.1109/ACCESS.2024.3514972
- **Systematization of Knowledge (SoK): A Systematic Review of Software-Based Web Phishing Detection"**, *IEEE Communications Surveys & Tutorials*, 2017
  - An authoritative survey of phishing detection systems, including heuristic methods, machine learning approaches, and rule-based systems. This paper provides the theoretical foundation for combining multiple detection signals into a unified scoring framework.
- **Phishing Webpage Detection: Unveiling the Threat Landscape and Investigating Detection Techniques"**, *IEEE Communications Surveys & Tutorials*, 2024
  - A recent survey that categorizes detection methods into URL-based, webpage-based, and hybrid approaches. It offers modern context for the types of heuristic checks (such as domain structure, certificate validity, and similarity detection) used in PhishEye.

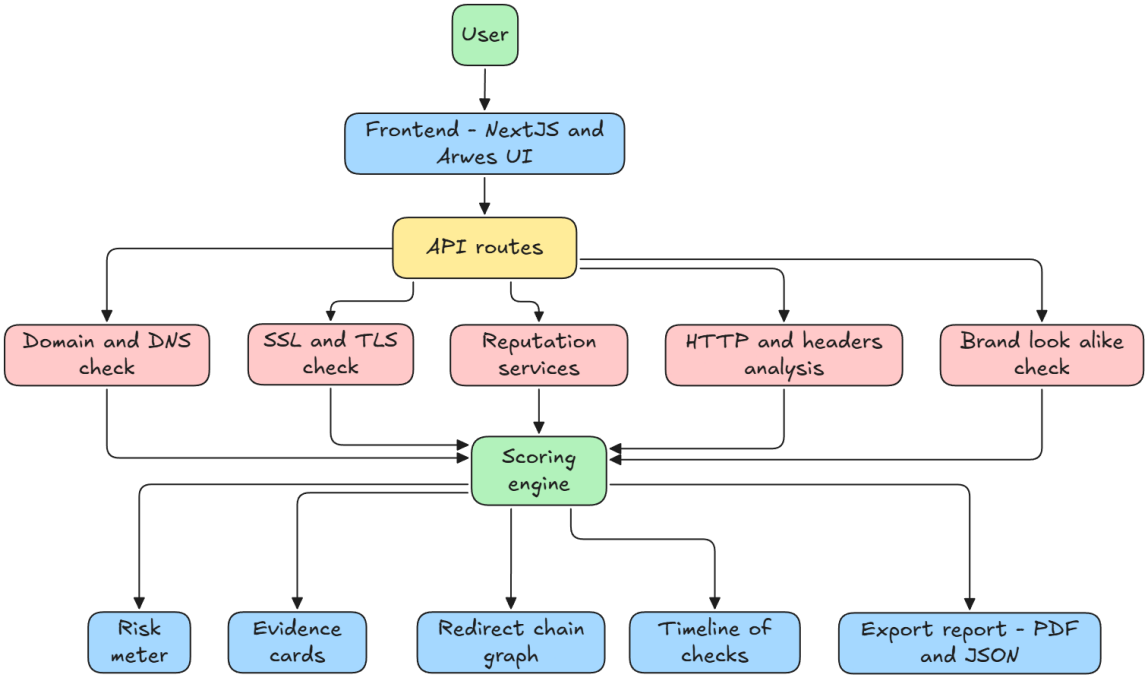
- **URL2Vec: URL Modeling with Character Embeddings for Fast and Accurate Phishing Website Detection"**, *IEEE ISPA/IUCC/BDCloud/SocialCom/SustainCom*, 2018
    - This paper introduces a novel technique using character-level embeddings of URLs for phishing detection, achieving both fast and accurate classification. Its approach reinforces the value of textual URL features in detection heuristics, which aligns with PhishEye's brand imitation detection.



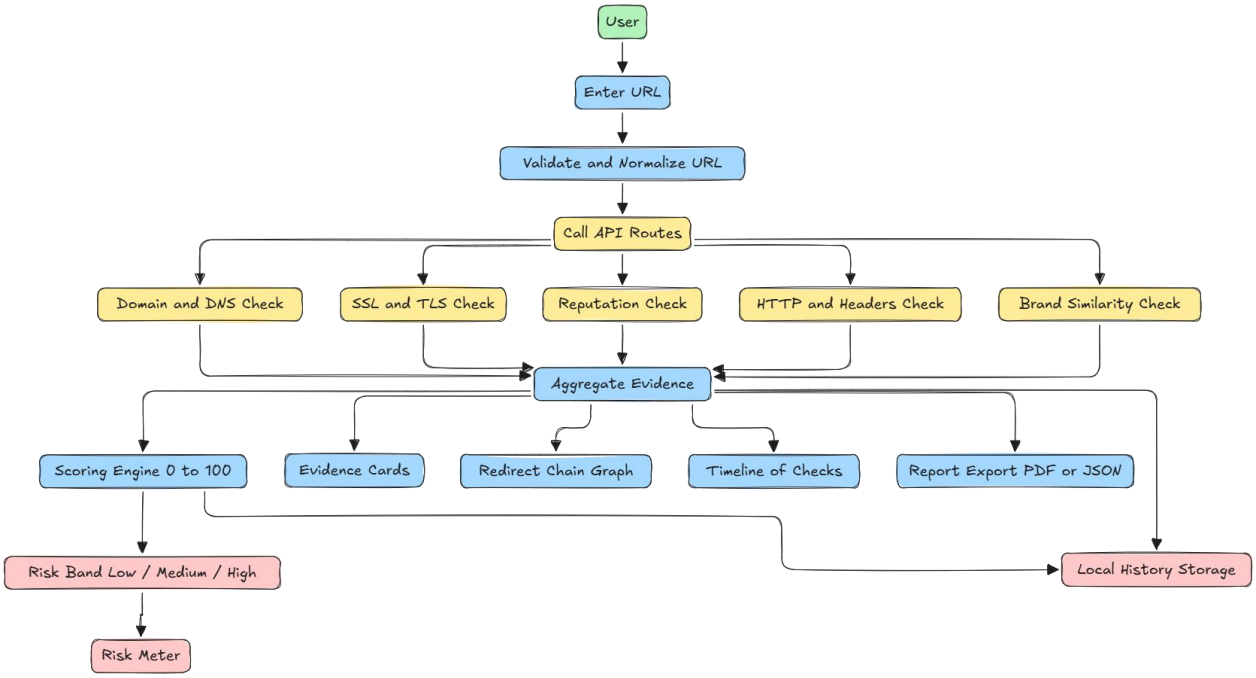
3: Research Map Diagram

Chapter 3: Literature Survey

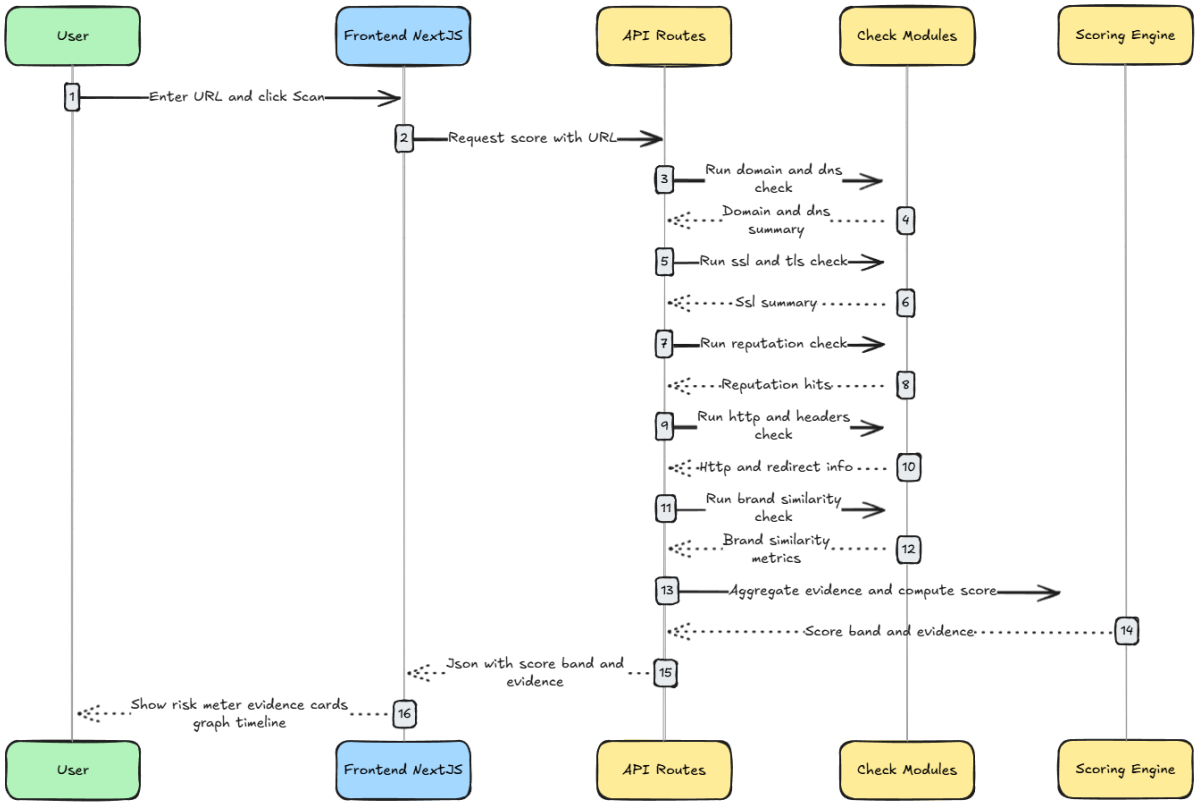
3.1 System Architecture



3.2 Data Flow Diagram (DFD – Level 1).



3.3 Use Case Diagram



3.4 Details of Hardware & Software

Hardware:

Any modern development machine capable of running Node.js 18+ and Python 3.x.

Software:

• Node.js 18+, npm or yarn

• React 18 + TypeScript (Vite)

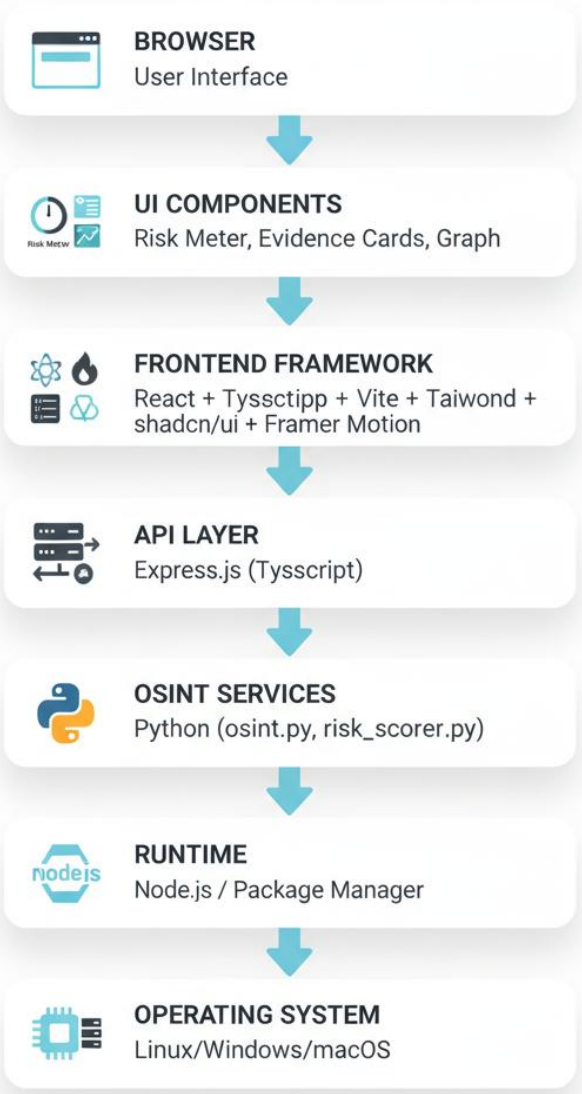
• Express.js (TypeScript)

• Python 3.x (osint.py, risk_scorer.py)

• shadcn/ui, Framer Motion, Zod, React Hook Form

• html2pdf.js (report export)

• (MVP) Browser LocalStorage (no server DB). External DB optional for future versions..



4: Environment Stack Figure

3.5 Coding

Types (foundation)

```
import { securityServiceManager } from './services';
import { mockScoreApi, mockReputationApi, mockDomainIntelApi } from
'./mockApi';
import { apiConfig } from './config';
import type {
  ScoreResponse,
  ReputationResponse,
  DomainIntelResponse,
  HistoryResponse,
  ScanSummary
} from '@/types/api';
import { extractDomain } from './url';
```

```
const API_BASE = '/api'; // Would be actual API base in Next.js
      throw error;
    }
  }

  async getReputation(url: string): Promise<ReputationResponse> {
    try {
      // Always try real services first, regardless of mock setting
      const availableServices = await
securityServiceManager.getAvailableServices();

      if (availableServices.length > 0) {
        console.log(`🔍 Using real services for reputation:
${availableServices.join(', ')}`);
        return await securityServiceManager.getReputation(url);
      }

      // Only use mock if no real services are available
      if (apiConfig.mockApis) {
        console.log('🔧 No real services available, using mock APIs for
reputation');
        return await mockReputationApi(url);
      }

      throw new Error('No API services available and mock APIs disabled');
    } catch (error) {
      console.error('Reputation API error:', error);

      // Fallback to mock API if real services fail and mock is enabled
      if (apiConfig.mockApis) {
        console.warn('⚠️ Real API failed, falling back to mock');
        try {
          return await mockReputationApi(url);
        } catch (mockError) {
          throw new Error('Failed to get reputation data from both real and
mock APIs');
        }
      }

      throw error;
    }
  }

  async getDomainIntel(host: string): Promise<DomainIntelResponse> {
    try {
      // Always try real services first, regardless of mock setting
      const availableServices = await
securityServiceManager.getAvailableServices();
```

```
      if (availableServices.length > 0) {
        console.log(`🔍 Using real services for domain intel:
${availableServices.join(', ')}`);
        return await securityServiceManager.getDomainIntel(host);
      }

      // Only use mock if no real services are available
      if (apiConfig.mockApis) {
        console.log('🔧 No real services available, using mock APIs for
domain intel');
        return await mockDomainIntelApi(host);
      }

      throw new Error('No API services available and mock APIs disabled');
    } catch (error) {
      console.error('Domain Intel API error:', error);

      // Fallback to mock API if real services fail and mock is enabled
      if (apiConfig.mockApis) {
        console.warn('⚠️ Real API failed, falling back to mock');
        try {
          return await mockDomainIntelApi(host);
        } catch (mockError) {
          throw new Error('Failed to get domain intelligence from both real
and mock APIs');
        }
      }

      throw error;
    }
  }
  }
}

// Export singleton instance
export const apiClient = new ApiClient();

// SWR fetchers
export const fetchers = {
  score: (url: string) => apiClient.getScore(url),
  reputation: (url: string) => apiClient.getReputation(url),
  domainIntel: (host: string) => apiClient.getDomainIntel(host),
  history: (cursor?: string) => apiClient.getHistory(cursor),
};

// API route handlers (these would be actual Next.js API routes)
export const apiHandlers = {
```

```
  // GET /api/score?url=...
  handleScore: async (url: string) => {
    const response = await mockScoreApi(url);
    return new Response(JSON.stringify(response), {
      headers: { 'Content-Type': 'application/json' }
    });
  },

  // GET /api/reputation?url=...
  handleReputation: async (url: string) => {
    const response = await mockReputationApi(url);
    return new Response(JSON.stringify(response), {
      headers: { 'Content-Type': 'application/json' }
    });
  },

  // POST /api/submit-scan
  handleSubmitScan: async (url: string) => {
    const response = {
      jobId: crypto.randomUUID(),
      status: 'queued',
      url
    };
    return new Response(JSON.stringify(response), {
      headers: { 'Content-Type': 'application/json' }
    });
  },

  // GET /api/history?cursor=...
  handleHistory: async (cursor?: string) => {
    const response: HistoryResponse = {
      scans: [],
      total: 0,
      cursor: undefined
    };
    return new Response(JSON.stringify(response), {
      headers: { 'Content-Type': 'application/json' }
    });
  }
};
```

3.5.2 Scoring (core logic)

```
import type { Evidence, RiskBand } from '@/types/api';

export interface ScoringWeights {
  domain: number;    // 0-35
  reputation: number; // 0-35
```

```
  http: number;      // 0-20
  brandLookAlike: number; // 0-10
}

export const DEFAULT_WEIGHTS: ScoringWeights = {
  domain: 35,
  reputation: 35,
  http: 20,
  brandLookAlike: 10,
};

export interface ScoringResult {
  score: number;
  band: RiskBand;
  breakdown: {
    domain: number;
    reputation: number;
    http: number;
    brandLookAlike: number;
    deductions: number;
  };
}

export function calculateRiskScore(
  evidence: Evidence[],
  weights: ScoringWeights = DEFAULT_WEIGHTS
): ScoringResult {
  let domainScore = 0;
  let reputationScore = 0;
  let httpScore = 0;
  let brandScore = 0;
  let deductions = 0;

  // Process each piece of evidence
  evidence.forEach((item) => {
    const weight = item.score / 100; // Normalize to 0-1

    switch (item.source.toLowerCase()) {
      case 'rdap':
      case 'dns':
      case 'whois':
        domainScore += weight * weights.domain;
        break;

      case 'urlhaus':
      case 'virustotal':
        reputationScore += weight * weights.reputation;
        break;
```

```
        case 'ssl':
        case 'http':
        case 'content':
          httpScore += weight * weights.http;
          break;

        case 'brand':
        case 'typosquatting':
          brandScore += weight * weights.brandLookAlike;
          break;
      }

      // Apply deductions for positive indicators
      if (item.status === 'success') {
        switch (item.source.toLowerCase()) {
          case 'ssl':
            if (item.title.includes('EV') || item.title.includes('OV')) {
              deductions += 5;
            }
            break;
          case 'rdap':
            if (item.title.includes('age') && parseInt(item.value) > 365) {
              deductions += 10;
            }
            break;
        }
      }
    });

    // Calculate total score
    const rawScore = Math.min(100, domainScore + reputationScore + httpScore +
brandScore);
    const finalScore = Math.max(0, rawScore - deductions);

    // Determine risk band
    let band: RiskBand;
    if (finalScore <= 29) {
      band = 'LOW';
    } else if (finalScore <= 59) {
      band = 'MEDIUM';
    } else {
      band = 'HIGH';
    }

    return {
      score: Math.round(finalScore),
      band,
```

```
    breakdown: {
      domain: Math.round(domainScore),
      reputation: Math.round(reputationScore),
      http: Math.round(httpScore),
      brandLookAlike: Math.round(brandScore),
      deductions: Math.round(deductions),
    },
  };
}

export function getRiskColor(band: RiskBand): string {
  switch (band) {
    case 'LOW':
      return 'hsl(var(--success))';
    case 'MEDIUM':
      return 'hsl(var(--warning))';
    case 'HIGH':
      return 'hsl(var(--danger))';
    default:
      return 'hsl(var(--neutral-500))';
  }
}

export function getRiskMessage(score: number, band: RiskBand): string {
  switch (band) {
    case 'LOW':
      return 'Low risk - Appears legitimate';
    case 'MEDIUM':
      return 'Medium risk - Exercise caution';
    case 'HIGH':
      return 'High risk - Likely malicious';
    default:
      return 'Unknown risk level';
  }
}
```

3.6 Source Code Repository

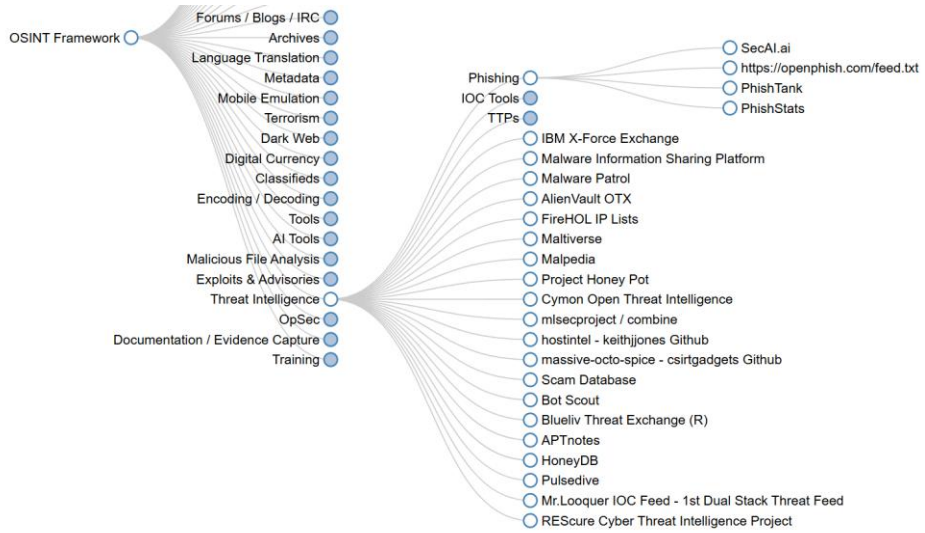
Primary Repo: PhishEye – Advanced URL Security Scanner

3.7 OSINT Frameworks Used

PhishEye integrates multiple **Open Source Intelligence (OSINT) APIs and frameworks** to perform real-time phishing and scam detection. These external sources provide reputation data, domain intelligence, SSL/TLS certificate verification, and automated scanning capabilities that power the system's risk scoring algorithm. Each integration is described below with a corresponding screenshot.

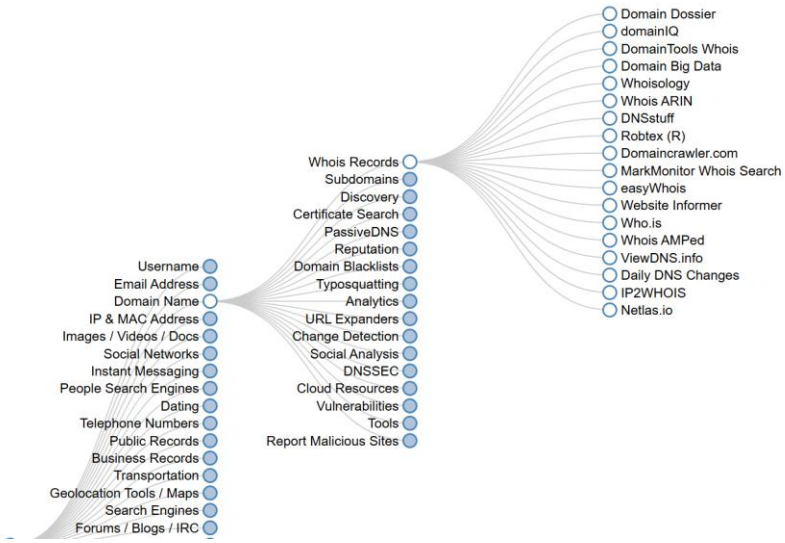
Threat Intelligence - Google Safe Browsing API (phishing)

Used to detect malicious and phishing websites by leveraging Google's continuously updated threat intelligence database. Provides verdicts on whether a URL is suspicious, malicious, or safe.



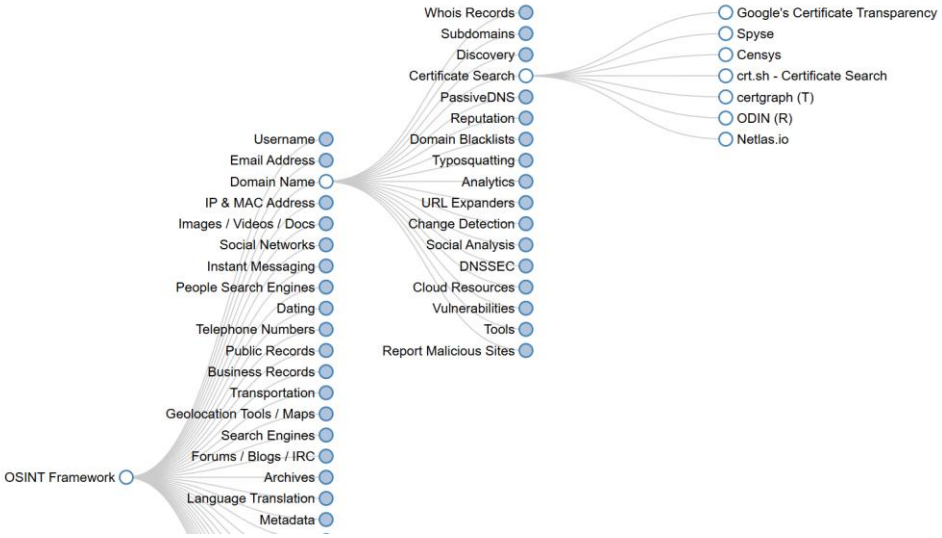
Domain Name Analysis – Whois Records (Who.is)

Retrieves domain registration details such as registrar, owner information, and domain age. Useful for detecting recently registered domains, which are often used in phishing.



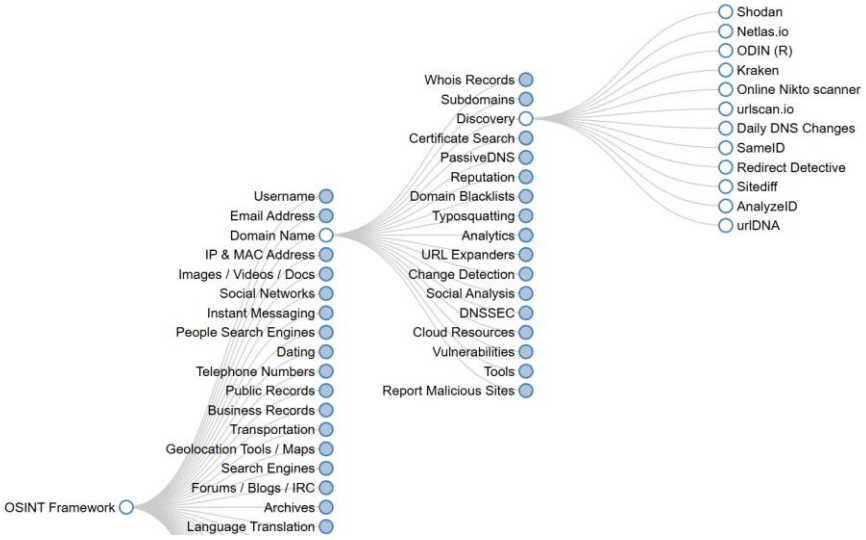
Domain Name Analysis - Certificate Transparency (crt.sh, Google's Certificate Search)

Searches certificate transparency logs to validate SSL/TLS certificates associated with a domain. Detects suspicious or mismatched certificates.



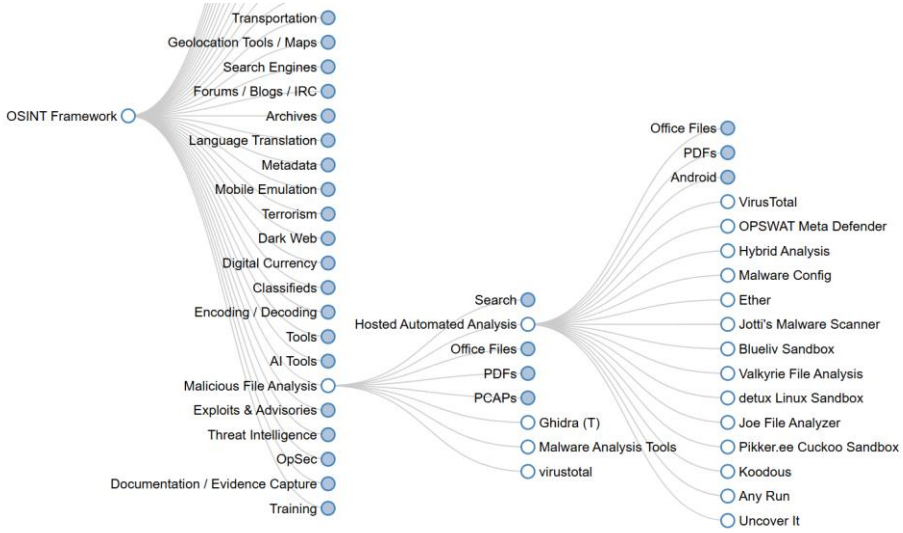
Domain Name Analysis - Discovery (urlscan.io)

Performs automated scanning of a URL, providing screenshots, page analysis, and verdicts. Used in PhishEye to cross-check suspicious domains visually and structurally.



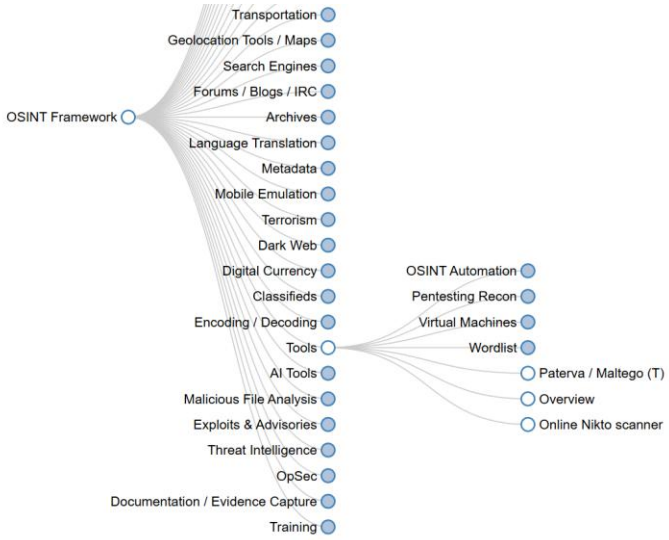
Malicious File Analysis - Hosted Automated Analysis (VirusTotal)

Beyond URLs, VirusTotal is also used for hosted file analysis. PhishEye leverages this capability to detect potentially malicious attachments or hosted executables.



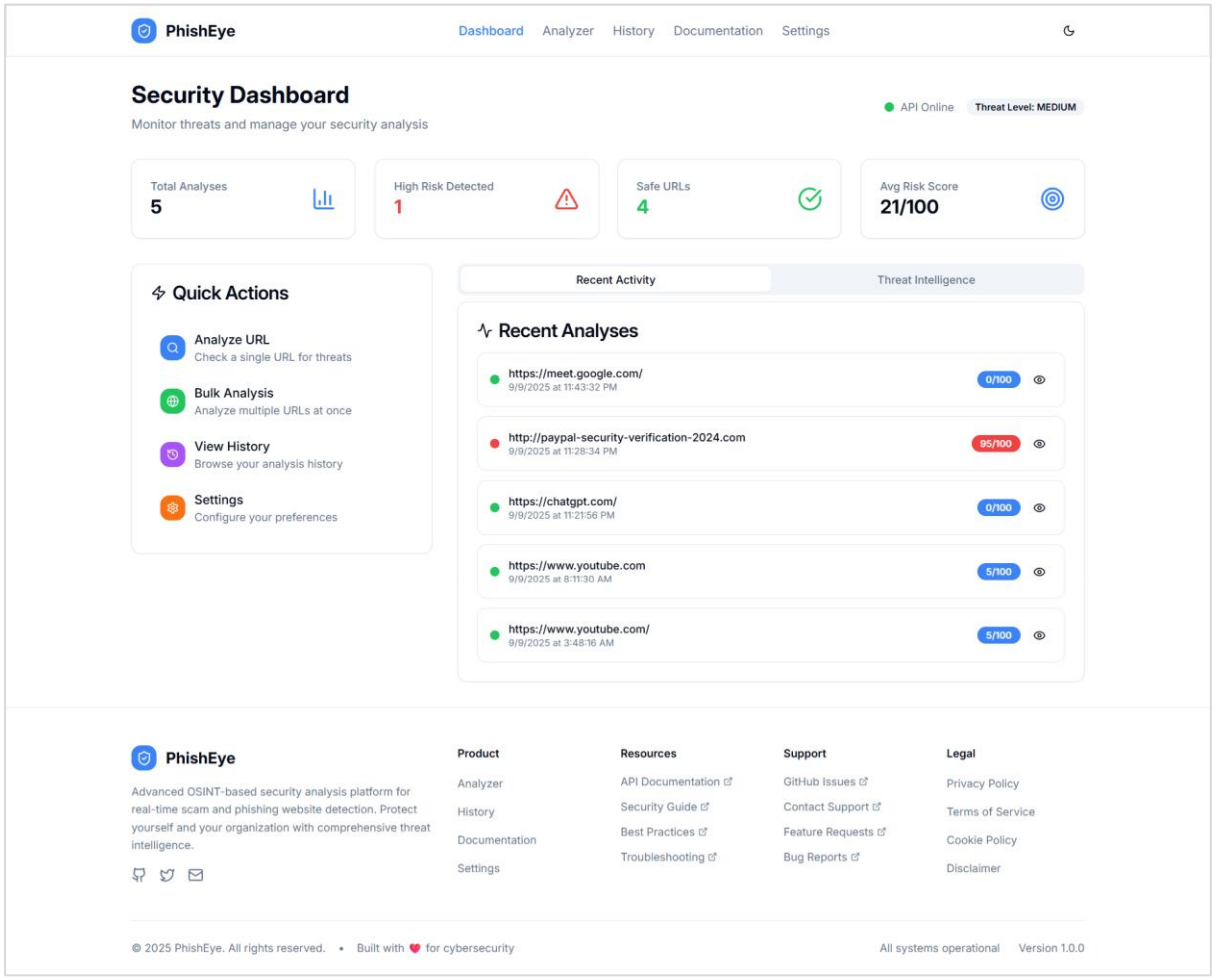
Tools – Risk Scoring (overview)

A custom-built module in PhishEye allowing batch scanning of up to 50 URLs simultaneously, with progress tracking and real-time results.



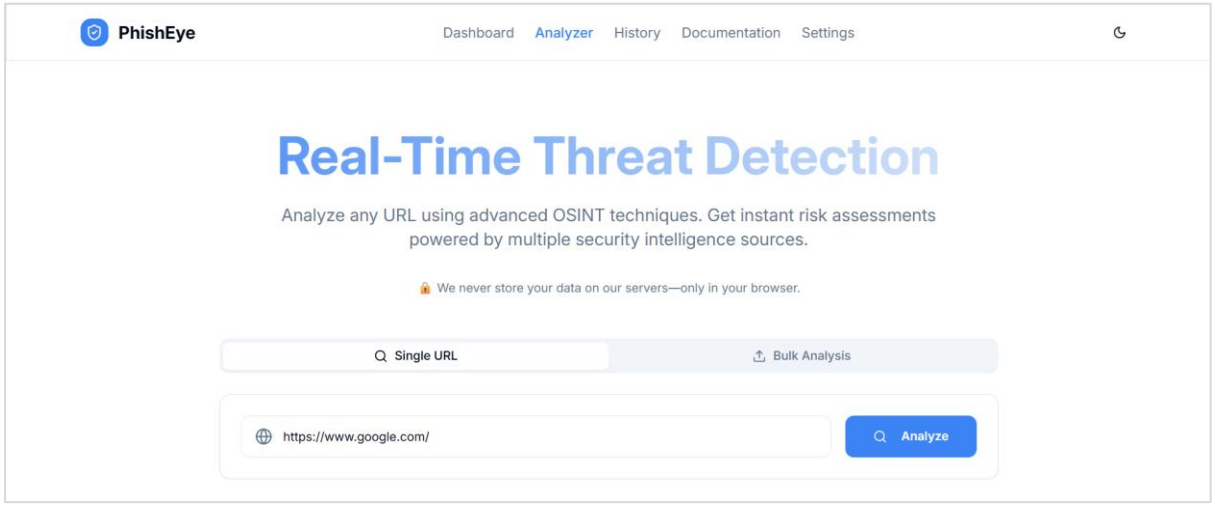
Chapter 4: Snapshots

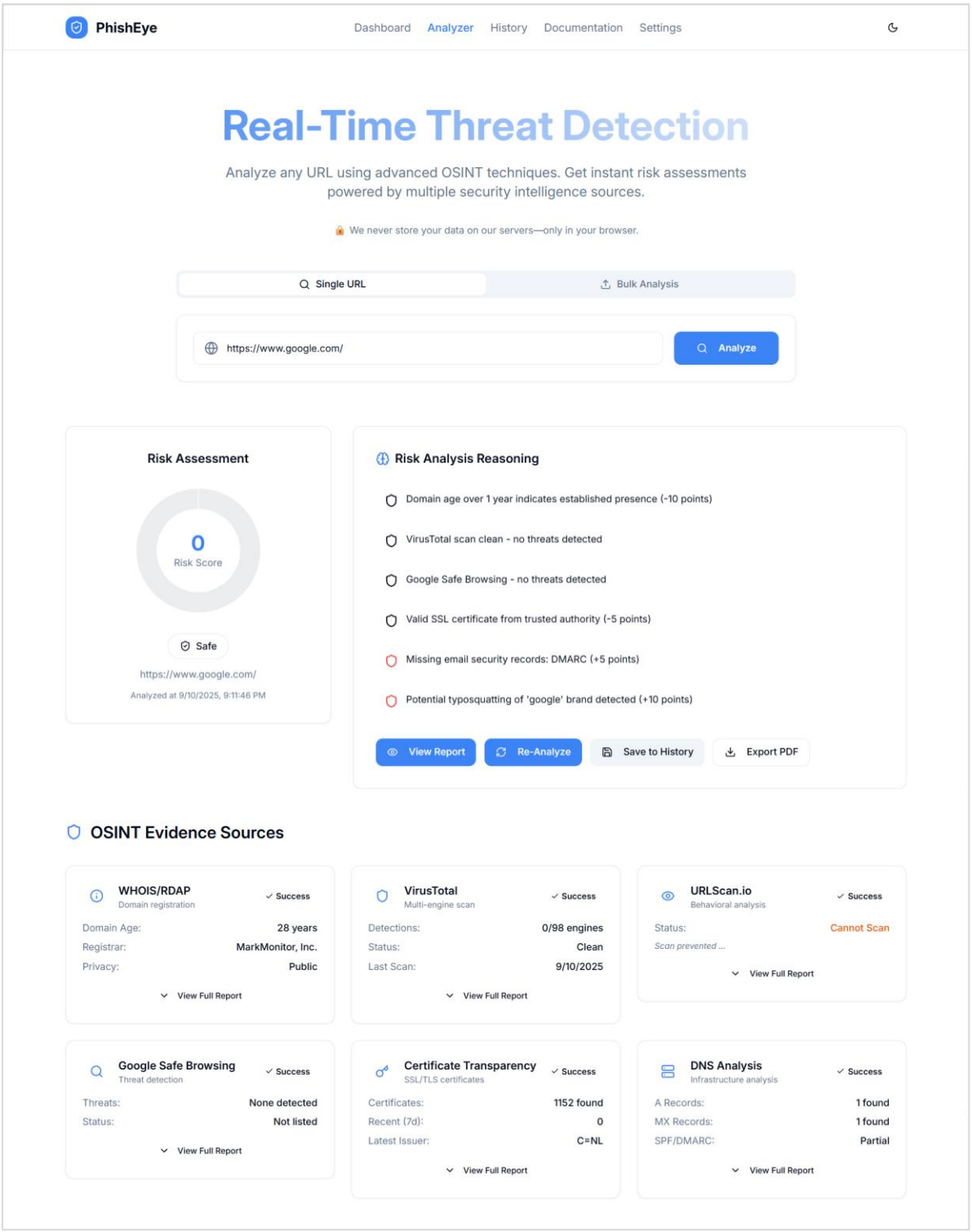
4.1 Dashboard



Central hub with live statistics (total analyses, high-risk count, safe URLs, average score), API health, threat level indicator, quick actions, recent activity, and a simulated threat intelligence feed.

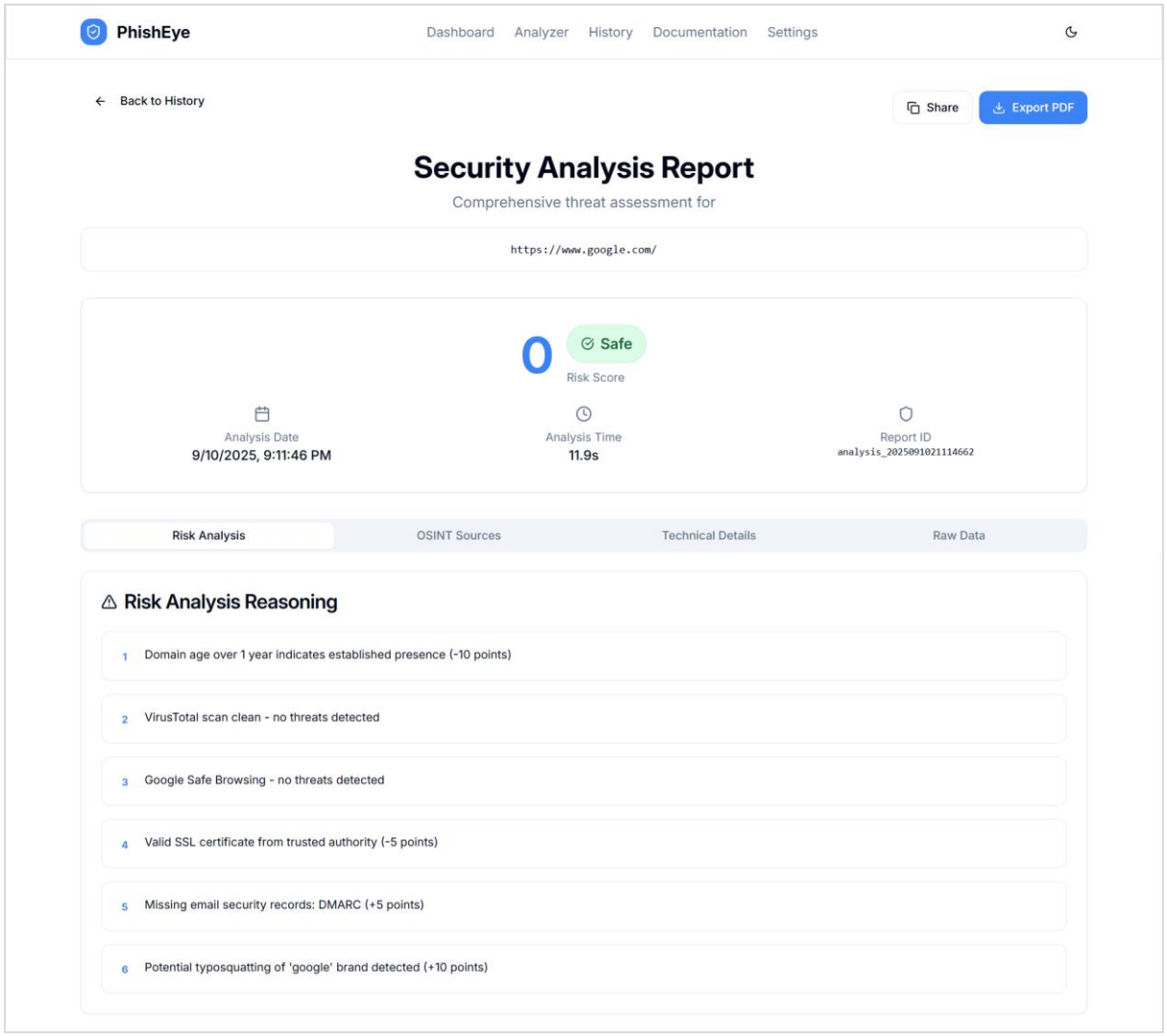
4.2 Analyzer





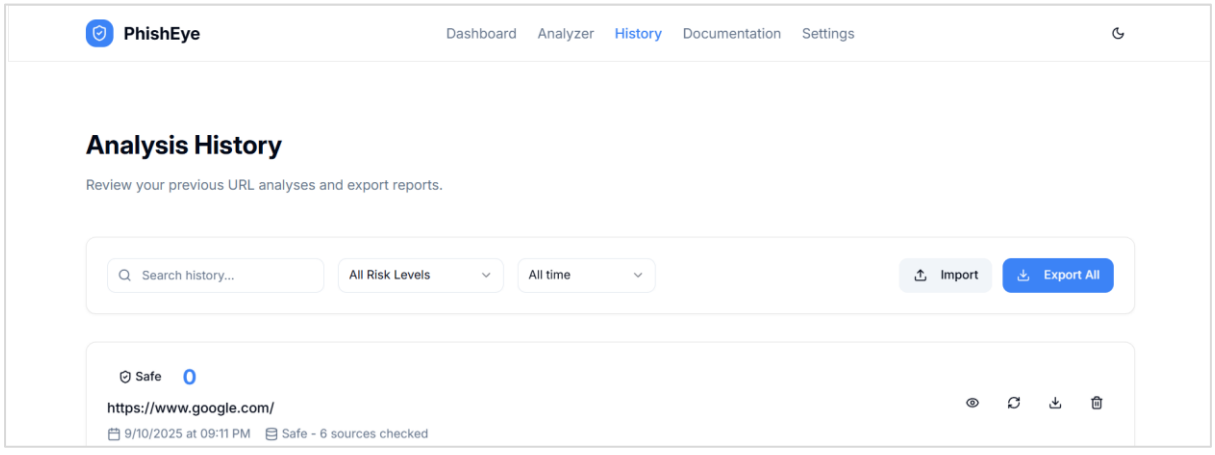
Single and Bulk modes (up to 50 URLs). Shows interactive risk meter (donut chart), evidence cards per source, human-readable reasons, and actions (View Report, Re-Analyze, Save, Export PDF).

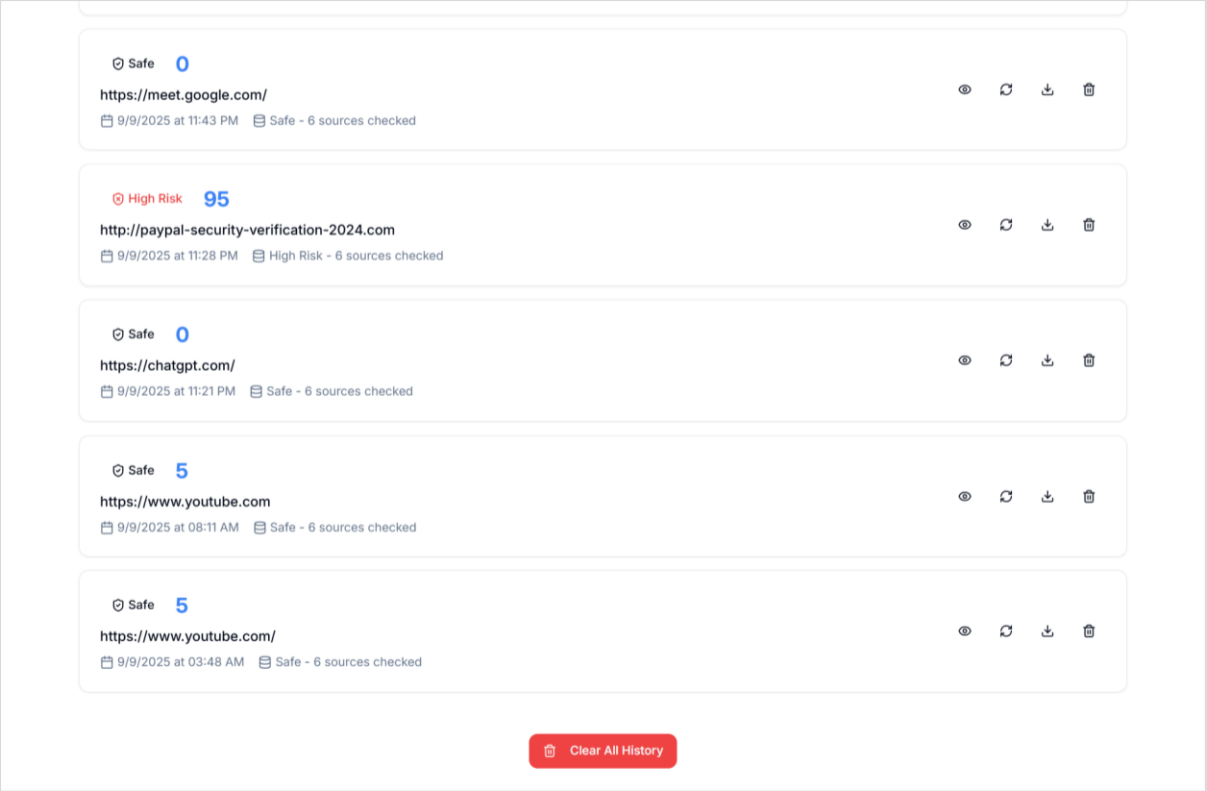
4.3 Report



Tabbed view: Risk Analysis, OSINT Sources, Technical Details, Raw Data. Includes score, verdict, metadata, per-source results, and PDF export capability.

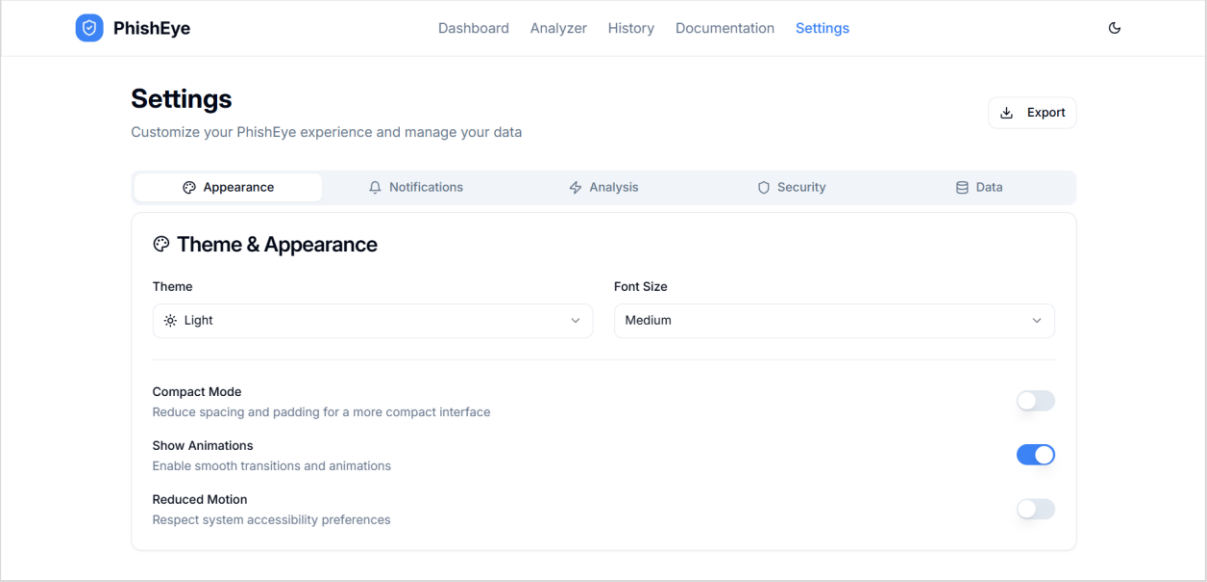
4.4 History





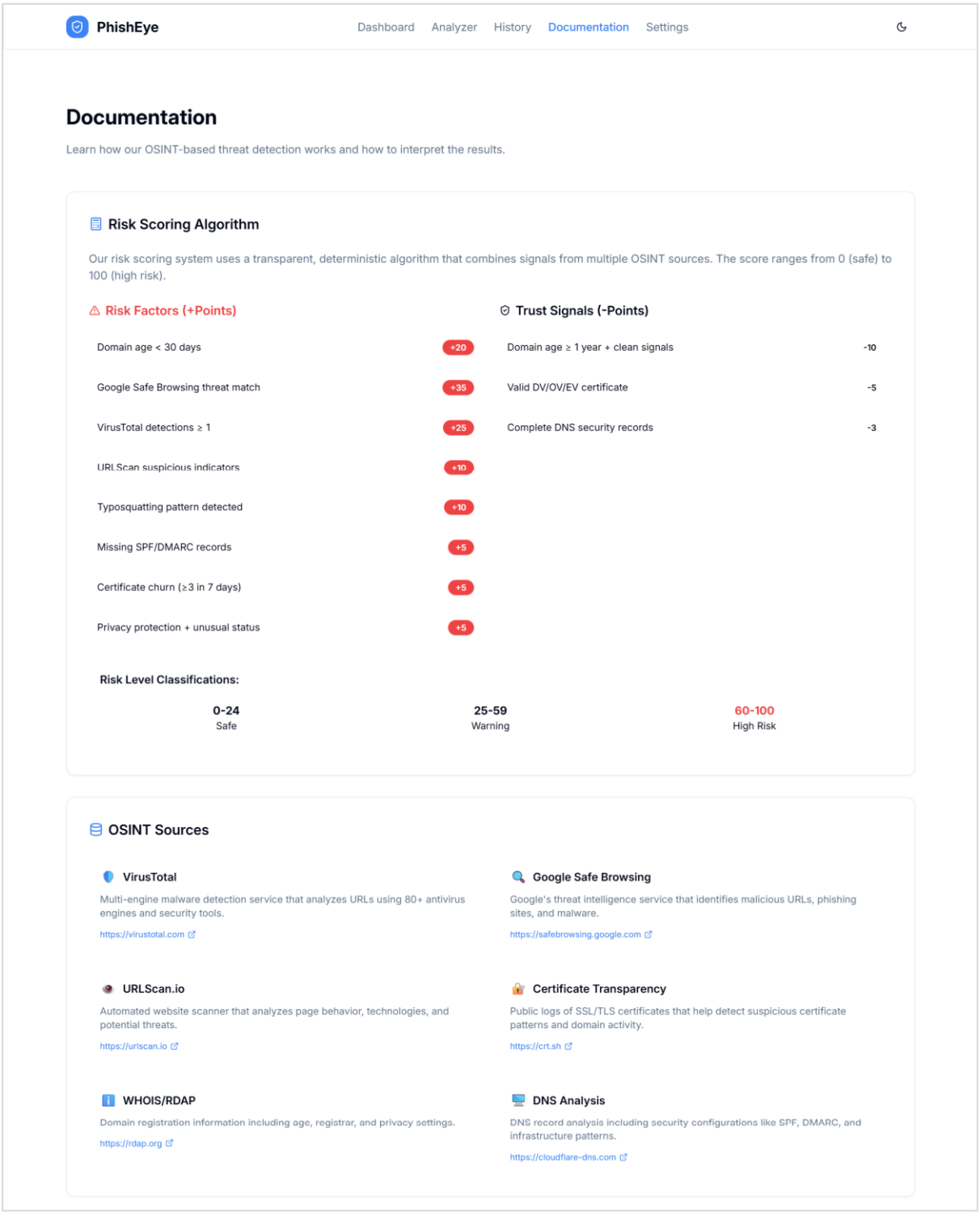
Paginated list with filters (URL, risk level, time period), quick actions per entry, and bulk operations (export/import/clear). Shows storage usage.

4.5 Settings



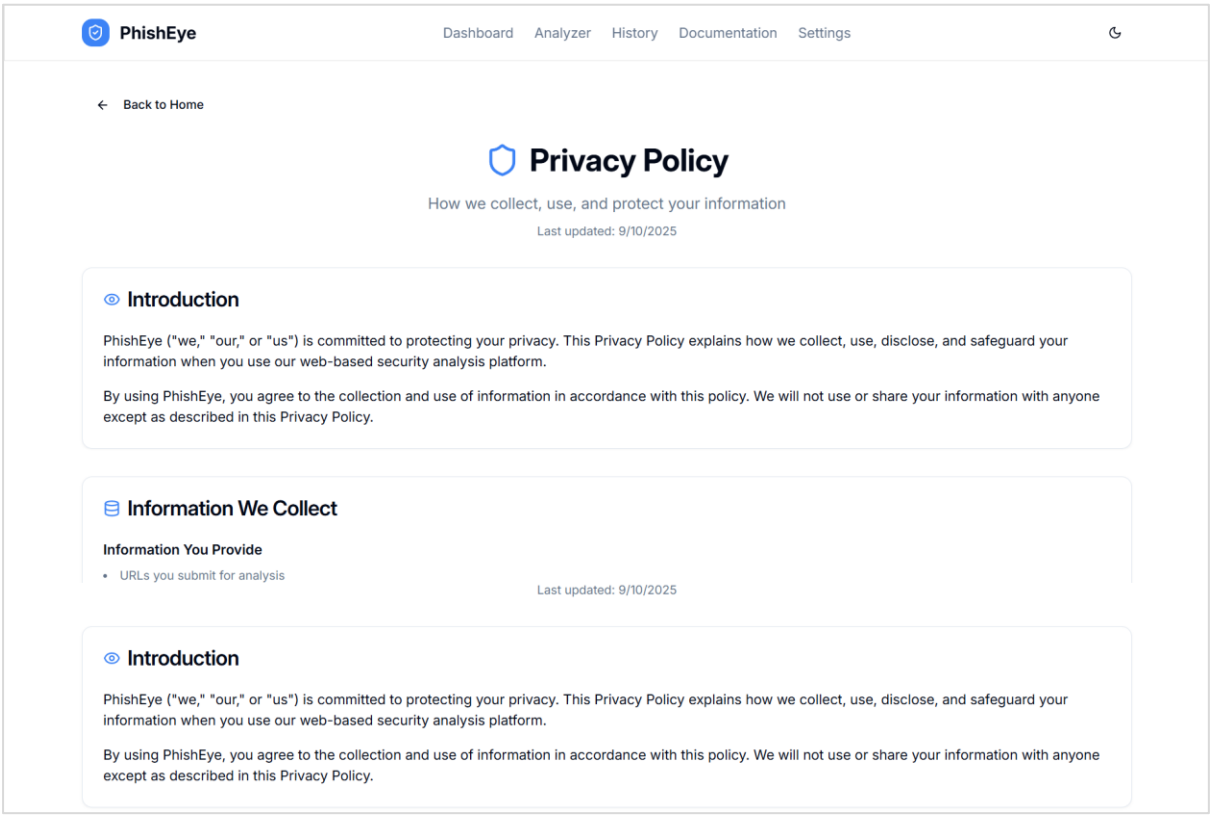
Five tabs: Appearance (theme, font, compact mode, animations)Notifications (browser/sound/email/vibration)Analysis (auto-start, bulk, deep scan, timeouts), Security (local encryption, anonymization, retention), Data Management (export/import, branding, storage monitor).

4.6 Documentation



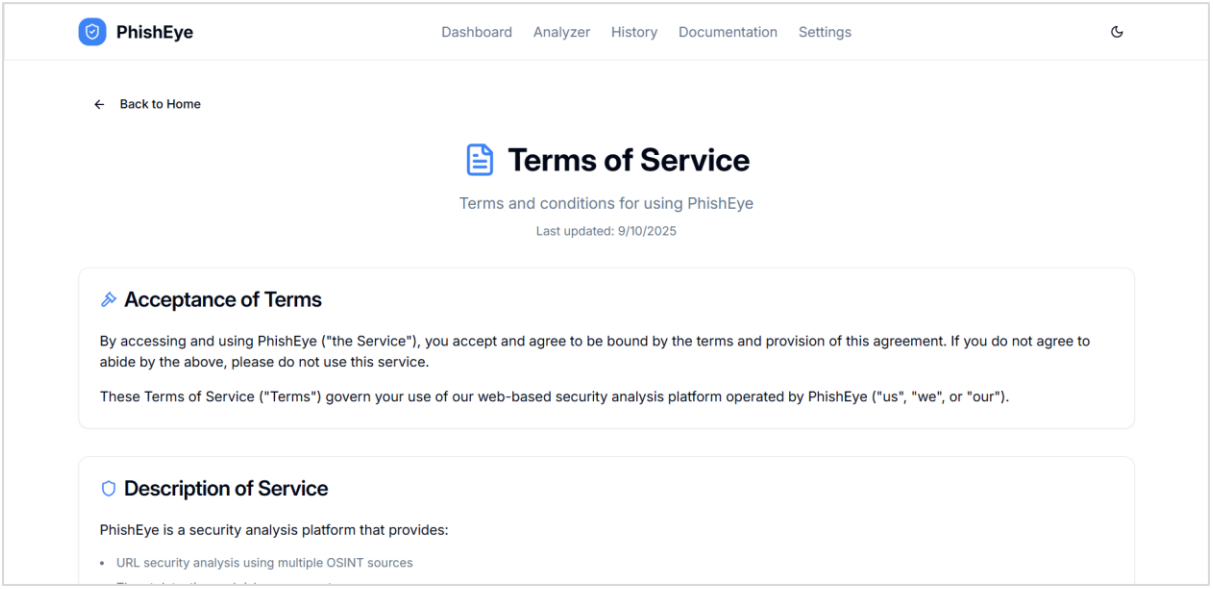
Explains scoring methodology, data sources, privacy and limitations, and provides external links. Aims for transparency and user education.

4.7 Privacy Policy



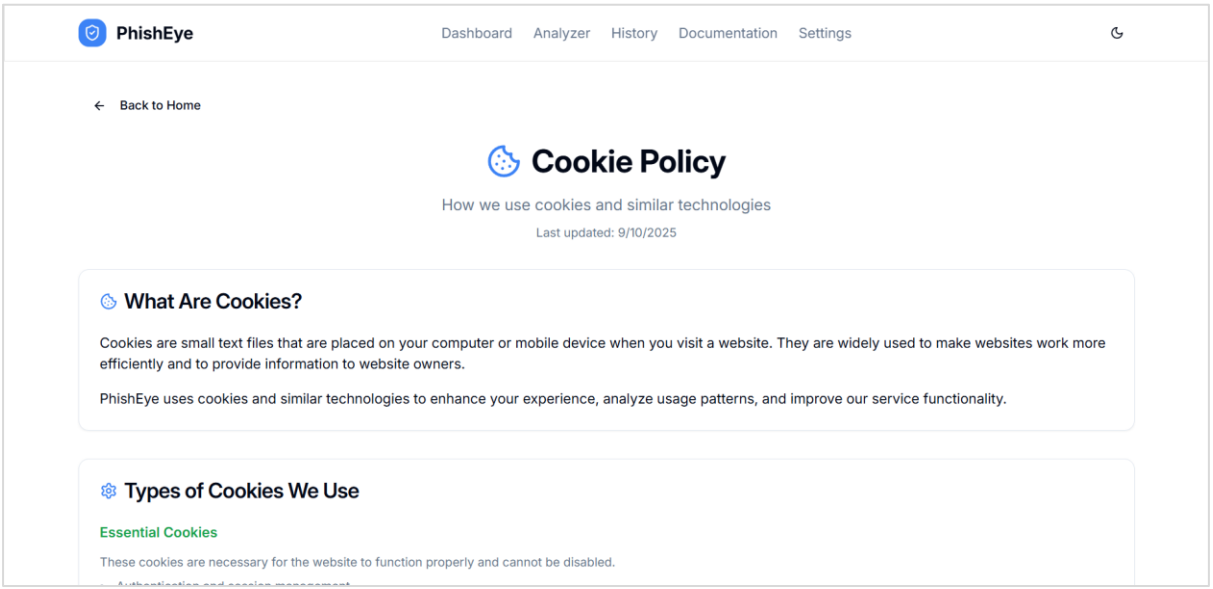
Details on information collection and use, storage and security, user rights, and third-party providers. Aligns with privacy best practices.

4.8 Terms of Service



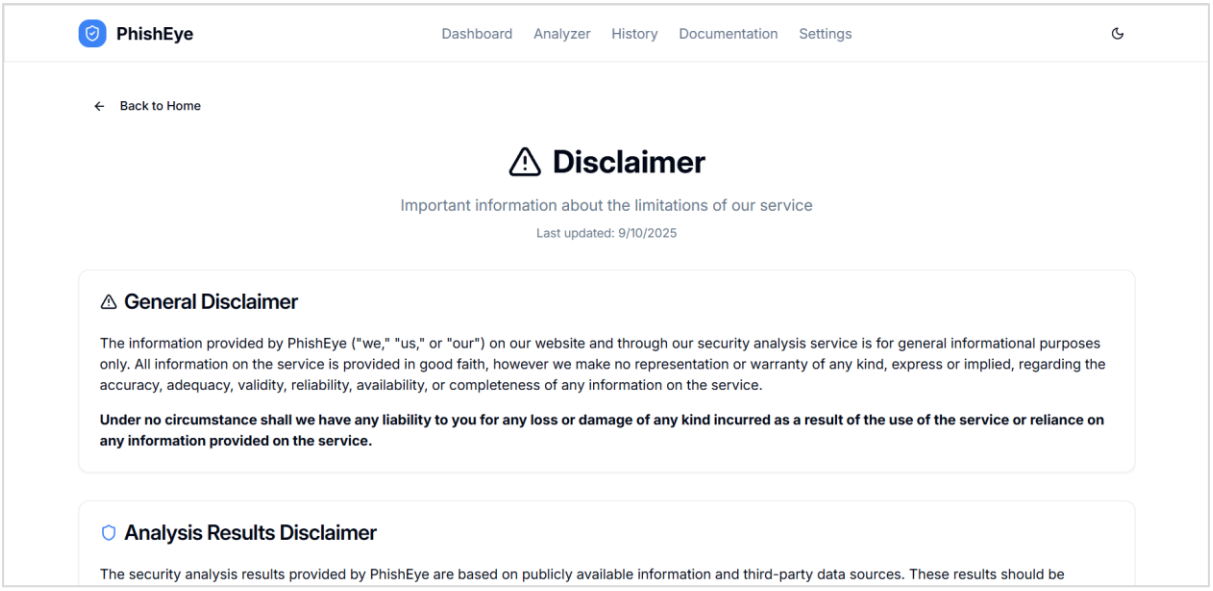
Covers service description, user responsibilities, prohibited uses, IP rights, limitation of liability, indemnification, termination, and changes.

4.9 Cookie Policy



Explains cookie types/purposes, retention durations, third-party cookies, and browser-level management instructions.

4.10 Disclaimer



Clarifies limitations of analysis, third-party data dependence, absence of professional guarantees, and scope boundaries.

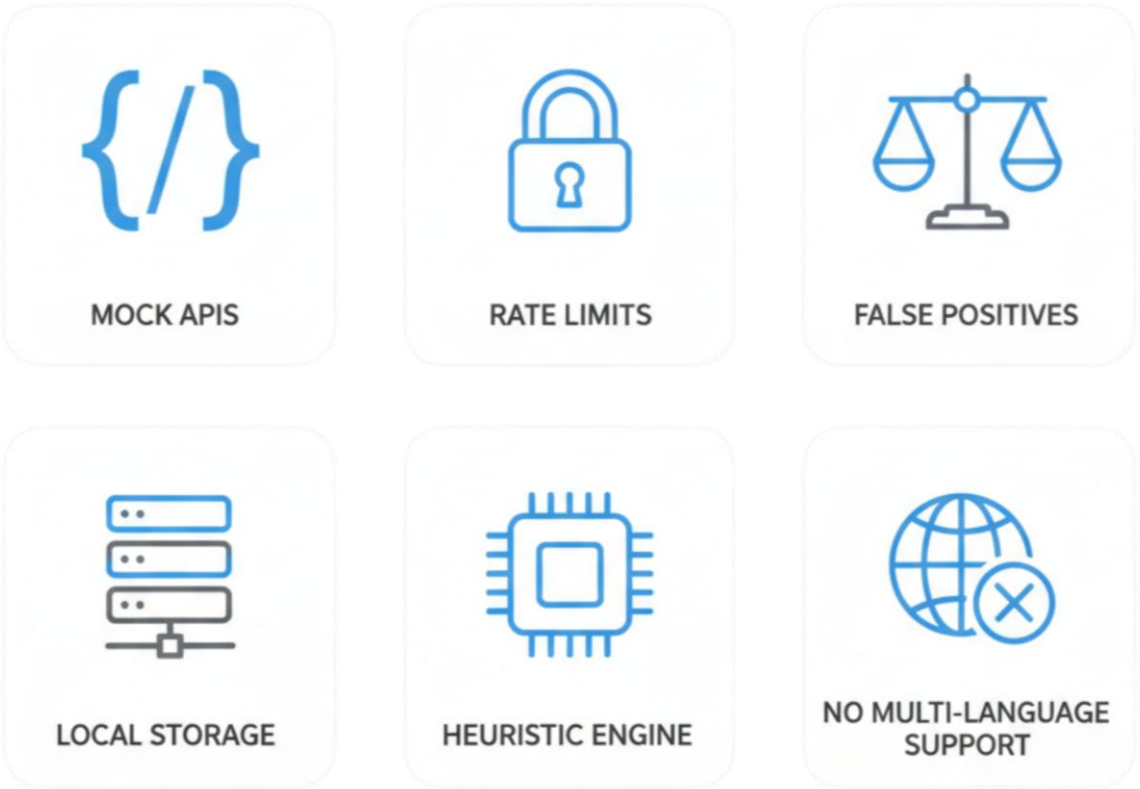
Chapter 5: Future Development

5.1 Limitations

The modular split between the TypeScript API and Python OSINT services simplifies extension. New providers can be added by implementing a provider adapter in osint.py and registering its contribution in the scorer, preserving transparency through evidence cards.

Although PhishEye demonstrates the feasibility of an OSINT-based phishing detection framework with a futuristic UI, the current implementation has some constraints:

- **Mocked APIs**: Current version uses deterministic stub responses instead of live integrations.
- **Heuristic Scoring Only**: No machine learning or adaptive detection models yet.
- **API Rate Limits**: Real services like VirusTotal and Google Safe Browsing have request limits on free tiers.
- **No Persistence**: History is stored locally; no backend database for multi-user storage.
- **False Positives/Negatives**: Pure heuristics may misclassify certain URLs without broader context.
- **Limited Internationalization**: Currently English-only UI without localization support.



5.2 Future Enhancements

Several improvements are envisioned to strengthen PhishEye and make it production-ready:

• Machine Learning Integration for pattern recognition and adaptive thresholds

• User Authentication & RBAC for multi-user teams

• Custom Risk Rules and organization-level policies

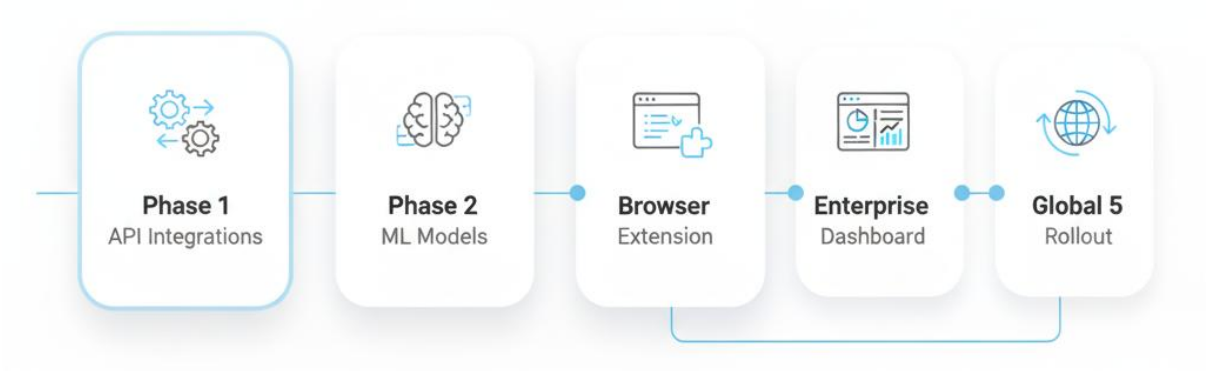
• Webhooks & Integration APIs for real-time notifications

• Advanced Reporting (custom templates, scheduled delivery)

• Expanded Threat Feeds and IOC enrichment

• Performance: caching layer, CDN, lazy loading, service workers

• Security: rate limiting, input hardening, audit logs, regular pentesting



Chapter 6: Conclusion

The mini-project **PhishEye – Advanced URL Security Scanner** was conceived and developed to address one of the most pressing issues in cybersecurity today: the rapid growth of phishing and scam websites. With the increasing dependence on digital platforms for communication, commerce, and finance, users are more exposed than ever to malicious attempts that mimic trusted entities. This project demonstrates how **Open-Source Intelligence (OSINT)** can be harnessed in a systematic way to identify threats in real time by consolidating information from multiple public intelligence sources and presenting it in an accessible, educational format.

The system is built as a **frontend-only application** using **Next.js, TypeScript, Tailwind CSS, Arwes, and GSAP**, with deterministic API stubs designed for easy integration of live services in the future. By focusing on modularity and visualization, PhishEye goes beyond simple blacklist checks and instead provides a **multi-dimensional analysis** of each URL. It evaluates aspects such as domain age, SSL/TLS quality, HTTP headers, DNS configuration, reputation feeds, and brand similarity. The results are consolidated through a heuristic scoring engine that outputs a clear **0–100 score**, categorized into Low, Medium, or High risk bands. This combination of OSINT-driven intelligence and user-centric presentation offers both **practical protection** and **learning opportunities** for students, researchers, and cybersecurity enthusiasts.

One of the most significant contributions of this project lies in its **user interface design and educational intent**. Unlike conventional scanners that deliver raw data or simple verdicts, PhishEye provides evidence cards, redirect chain graphs, animated timelines, and a cinematic risk meter that resemble a “sci-fi hacking console.” This not only engages users but also helps them understand *why* a URL is considered suspicious. For students, the tool doubles as a teaching aid to explore web security fundamentals. For organizations, it offers a proof-of-concept model that can be extended into enterprise dashboards or browser extensions

While the current version has limitations—such as reliance on mocked API responses, heuristic-only detection, and local storage—it provides a solid foundation for future growth. Planned enhancements include **live API integrations** with platforms like VirusTotal, Google Safe Browsing, and URLHaus; **machine learning models** for adaptive detection; and **browser extensions** for seamless real-time protection. These developments will enable PhishEye to move from a student-driven prototype to a practical open-source tool with real-world applications..

In conclusion, PhishEye v1.0.0 delivers an end-to-end, privacy-preserving OSINT analyzer with 11 functional pages, bulk URL processing, professional PDF reports, and transparent evidence-based scoring. The architecture is intentionally modular, enabling straightforward addition of new intelligence sources and ML-based capabilities planned for subsequent releases.

APPENDICES

Sr No	Appendix	Description	Page. No.
1	Dashboard	Central hub with statistics, API health, threat level, quick actions, and activity feed	20
2	Analyzer	Single and bulk analysis with risk meter, evidence cards, reasoning, and export options	20
3	Report	Multi-tab reports with risk analysis, OSINT sources, technical details, and raw data	22
4	History	Analysis history with filters, pagination, quick actions, and bulk operations	22
5	Settings	Five-tab settings for appearance, notifications, analysis, security, and data management	23
6	Documentation	Explains scoring methodology, OSINT sources, privacy, and limitations	24
7	Privacy Policy	Data collection, storage, usage policies, and user rights	25
8	Terms of Service	Service scope, user responsibilities, limitations, and termination policies	25
9	Cookie Policy	Cookie types, purposes, retention, third-party use, and management	26
10	Disclaimer	Clarifies system limitations and third-party data dependencies	26

GLOSSARY OF TERMS

Term	Description
Phishing	A cyberattack that tricks users into providing sensitive data by imitating trusted websites.
OSINT	Gathering and analyzing data from publicly available sources for intelligence purposes.
Heuristic Scoring	A rule-based evaluation system used to assign phishing risk levels.
WHOIS	A protocol providing details about domain name ownership.
Risk Meter	A graphical representation of a URL's phishing risk level.
Redirect Chain	A sequence of URL redirects that can indicate suspicious behavior.
Evidence Card	A visual UI element showing specific results of phishing checks.
API Stub	Mocked API endpoints used for development and testing.
Frontend-only App	An application where all logic runs on the client-side browser.
Backend API	The server-side service (Express.js in PhishEye) that processes requests and coordinates OSINT checks.

Bulk Analysis	The process of analyzing multiple URLs (up to 50) simultaneously
Certificate Transparency (CT Logs)	Public logs used to verify SSL/TLS certificates and detect fraudulent or suspicious certificates.
VirusTotal	An OSINT platform that aggregates results from 70+ antivirus engines for URL and file analysis
URLScan.io	An OSINT tool that scans websites to provide screenshots, network requests, and verdicts.
Local Storage	Browser-based storage used to save analysis history without requiring a backend database.
PDF Report	Exportable professional document containing analysis results, evidence, and risk scoring.
Monitoring	A real-time interface in PhishEye that shows simulated threat detection and system health.

LIST OF REFERENCES

Ref. No.	Reference (IEEE Format)
[1]	A. Sharma, A. Gupta, and R. Choudhary, "A State-of-the-Art Review on Phishing Website Detection Techniques," *IEEE Access*, vol. 12, pp. 124567–124589, 2024. doi: 10.1109/ACCESS.2024.3514972.
[2]	A. Prakash and D. Das, "Systematization of Knowledge (SoK): A Systematic Review of Software-Based Web Phishing Detection," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1907–1934, 2017.
[3]	L. Han, P. Chen, and S. Xu, "Phishing Webpage Detection: Unveiling the Threat Landscape and Investigating Detection Techniques," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 1, pp. 1–28, 2024.
[4]	Y. Le, M. Alrubaian, and C. Xu, "URL2Vec: URL Modeling with Character Embeddings for Fast and Accurate Phishing Website Detection," in *Proc. IEEE ISPA/IUCC/BDCloud/SocialCom/SustainCom*, Melbourne, Australia, 2018, pp. 123–130.
[5]	Google Safe Browsing API. [Online]. Available: https://safebrowsing.google.com/
[6]	VirusTotal API. [Online]. Available: https://www.virustotal.com/gui/home/url
[7]	URLHaus Project, abuse.ch. [Online]. Available: https://urlhaus.abuse.ch/
[8]	PhishTank. [Online]. Available: https://phishtank.org/
[9]	Arwes.dev – Futuristic Sci-Fi UI Framework. [Online]. Available: https://arwes.dev/
[10]	GSAP – GreenSock Animation Platform. [Online]. Available: https://gsap.com/

BIBLIOGRAPHY

Sr. No.	Bibliographic Entry
1	M. Stamp, *Information Security: Principles and Practice*, 3rd ed. Wiley, 2022.
2	W. Stallings, *Cryptography and Network Security: Principles and Practice*, 8th ed. Pearson, 2020.
3	J. Erickson, *Hacking: The Art of Exploitation*, 2nd ed. No Starch Press, 2008.
4	C. Hadnagy, *Social Engineering: The Science of Human Hacking*, Wiley, 2018.
5	OWASP Foundation, “Phishing Prevention Cheat Sheet.” [Online]. Available: https://cheatsheetseries.owasp.org/