Helwan University

Faculty of Engineering
Communication and Electronics Engineering Department

DefendX

A Web-Based Security Analysis Platform

Graduation Project

Prepared by:	Ahmed Bakr Ahmed
	Ammar Yasser Mohamed
	Ragab Mohamed
	Sama Mohamed
	Shahd Wael
	Shorouq Mohammed Farouq
Supervised by:	Dr. Ahmed Abdelhaleem
Academic Year:	2025–2026
Submission Date:	____________________ 2026

Abstract

This thesis presents DefendX, a cybersecurity platform that focuses on social engineering risks. The project is implemented as a web system with a frontend, backend services, and external intelligence providers. The technical scope of this report is limited to two core tools: the Link Analyzer and the File Download Checker. The document explains architecture, security controls, threat analysis, and current implementation details for both tools.

Acknowledgements

TBD.

List of Abbreviations

Acronym	**Meaning**
API	Application Programming Interface
JWT	JSON Web Token
OAuth	Open Authorization
SSRF	Server-Side Request Forgery
TLS	Transport Layer Security

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.0.1 Project Overview

DefendX is a web-based security analysis platform focused on protecting users from social engineering attacks through a Flask API and a React/Vite dashboard. The system provides URL risk analysis, file download scanning, and operational logging.

Purpose

DefendX is a security analysis platform designed to identify and assess risks introduced through explicit user inputs, including:

- URLs
- Files

The platform provides explainable risk assessments, auditability, and actionable guidance without executing untrusted content. All analysis tools in DefendX operate exclusively on explicit user input and do not perform background monitoring, credential harvesting, or passive data collection.

Core Objectives

DefendX is designed to:

- Detect common and advanced threat indicators
- Provide transparent and explainable risk scores
- Preserve user privacy
- Maintain strong auditability
- Operate safely under partial failure

What DefendX Is

- A centralized security analysis engine
- A backend-driven trust decision system
- A modular tool-based architecture

What DefendX Is Not

- An antivirus engine
- A sandbox execution platform (by default)
- A replacement for endpoint security
- A real-time intrusion prevention system

Security Philosophy

DefendX follows these principles:

- Assume all external input is malicious
- Treat the frontend as untrusted
- Prefer false positives over false negatives
- Fail safely and visibly
- Avoid black-box decisions

High-Level Capabilities

Capability	Description
URL Risk Analysis	URL reputation, redirects, SSL, and optional dynamic checks
File Analysis	Static file inspection and reputation

Trust Model

- Backend: trusted
- Frontend: untrusted
- External services: untrusted but useful
- User input: hostile by default

System Boundary

DefendX makes security decisions server-side and returns results to clients. It does not execute untrusted content or take autonomous remediation actions.

Scope

- Backend API serving authenticated tool endpoints under `/api`.
- Frontend SPA that drives authenticated workflows and reports.
- User-invoked URL and file analysis workflows through authenticated API requests.

Primary Data Flows

- Users authenticate via `/api/auth/login` with RSA-encrypted credential fields.
- Backend issues JWT access and refresh tokens and stores refresh token state in the database.
- Tool endpoints process user inputs, compute risk scores, and log usage in `History` and `Tool_Counter` tables.
- Frontend displays findings, status, and usage history from the API.

Data Classes

- Account data: `User` (username, email, role, suspended flags).
- Auth data: JWT access and refresh tokens, refresh token server-side records.
- Tool data: user-provided URLs, file metadata, and tool outputs.
- Logs: tool usage history and per-user counters.

External Dependencies

- VirusTotal, Google Safe Browsing, and IPGeolocation (API-backed checks).
- Playwright for optional dynamic link analysis.

Out of Scope

- Backend APIs for tools present only in frontend routes (USB, PCAP, Audio, User Behavior) are not confirmed in this repository. See `00-Overview/known-limitations.md`.

Chapter 2

System Architecture

2.0.1 Architecture Summary

System Overview

DefendX uses a layered architecture with strict separation of concerns:

- Presentation layer (Frontend)
- Security decision layer (Backend)
- External intelligence providers
- Persistent storage

High-Level Components

- Backend API: Flask + Flask-RESTX with a single API prefix `/api` and docs at `/api/docs`.
- Frontend: React/Vite SPA proxied to `/api` during development and optionally served from `frontend/dist` or `frontend/build` in production.
- Data Store: SQLAlchemy models backed by SQLite for `dev.db` and `test.db` (production database is TBD).
- Async Tasks: Celery worker factory wired to Flask config (`CELERY_BROKER_URL`, `CELERY_RESULT_BACKEND`).

Frontend

- Single-page application
- Handles user interaction
- No security authority

Backend

- Flask-based API
- Central security engine
- Authentication and authorization
- Tool orchestration
- Risk scoring

Database

- Stores users
- Stores history (audit logs)
- Stores tool usage counters

External Services

- Reputation APIs
- Intelligence feeds

Backend Entry Points

- `backend/app/main.py` creates the Flask app via `create_app(devconfig)` and registers all namespaces.
- `backend/app/__init__.py` performs request hardening and serves the SPA when a static build is present.

API Namespaces

- `auth`: signup, login, refresh, logout, public-key.
- `user`: admin CRUD and user detail endpoints.
- `history`: admin and user tool history.
- `tool_counter`: per-user and global usage counters.
- `tools/link-analyzer`: quick, deep, and custom scans.
- `tools/file-checker`: file upload scanning.
- `celery`: start and status for a demo task.

Trust Boundaries

- Browser clients communicate with the API via JWT-protected endpoints.
- Tool analyzers call out to external reputational APIs and services.

Static Web Serving

- Frontend build artifacts are served by Flask if `frontend/dist` or `frontend/build` exists.
- Request hardening blocks path traversal and source file access (e.g., `.env`, `.py`, `.jsx`).

Data Flow

1. User submits input via frontend
2. Frontend sends request to backend API
3. Backend validates authentication and input
4. Security tool executes
5. Risk score computed
6. History and counters updated
7. Result returned to frontend

Design Constraints

- No direct frontend access to databases
- No frontend access to external intelligence services
- All trust decisions server-side
- Minimal data retention

Architectural Benefits

- Strong auditability
- Clear trust boundaries
- Easier security review
- Modular extensibility

2.0.2 Architecture Diagrams (Plain Text)

System Context

[Browser SPA] ----HTTPS----> [Flask API /api] [Flask API /api] -----> [SQLite DB: User, History, Tool_Counter, RefreshToken] [Flask API /api] -----> [External APIs: VirusTotal, Google Safe Browsing, IPGeolocation]

Tool Processing Flow

1) Client obtains public RSA key from `/api/auth/public-key`.
2) Client encrypts credentials and calls `/api/auth/signup` or `/api/auth/login`.
3) Backend decrypts fields, validates credentials, and returns access/refresh tokens.
4) Client submits a URL or file to a tool endpoint with an access token.
5) Backend runs analysis, calculates risk score, then logs `History` and updates `Tool_Counter`.
6) Client renders result, history, and counter metrics.

Optional Dynamic Scan (URL Risk Analysis Tool)

User-invoked deep/custom URL scans may use Playwright to collect dynamic page indicators. If Playwright is not installed or disabled, this step may fail and should be treated as optional or degraded.

2.0.3 Architecture Diagrams (Text Description)

System Architecture

User → Frontend SPA → DefendX Backend → Security Tools → Database

The frontend communicates with the backend exclusively via HTTPS. All authentication and authorization occur in the backend.

Tool Execution Flow

Request → JWT validation
→ Input validation
→ Tool execution
→ Risk scoring
→ History logging
→ Response

External Services Interaction

The backend communicates with:

- VirusTotal
- Google Safe Browsing
- IP intelligence providers
- Additional reputation or URL intelligence providers (deployment-dependent)

The frontend never communicates directly with these services.

Authentication Flow

1. User logs in
2. Backend issues JWT access and refresh tokens
3. Frontend uses access token for API calls
4. Backend validates token on each request

Security Boundary

The backend is the sole trusted component. Compromise of the frontend does not imply compromise of the security engine.

2.0.4 Frontend Architecture

Framework

- React SPA built with Vite.
- Routing is managed with React Router (`frontend/src/router/AppRouter.jsx`).

App Layout and Routing

- Public routes: `/` (login), `/register`.
- Protected routes: `/home`, `/profile`, `/history`, tool pages, and documentation pages.
- Admin routes: `/admins` (enforced via `AdminRoute`).

API Client

- Axios instance configured with base URL from `VITE_API_URL` or `/api` by default.
- Request middleware attaches access token from `localStorage`.
- Response middleware refreshes access token via `/api/auth/refresh` and retries once.

Tool Pages

- URL Risk Analysis Tool and File Download Checker.
- Additional tool pages exist (USB, PCAP, Audio, Behavioral Risk) but backend APIs are not confirmed in this repo. TBD (verify in code).

State and Storage

- Tokens and user profile are stored in `localStorage`.
- Admin role checks are derived from locally stored user payload.

Code Alignment Notes

- `/tools/user-behavior/analyze` is referenced in `frontend/src/api/index.js`, but no backend endpoint was found. TBD (verify in code).

Purpose

The DefendX frontend is the **user-facing interaction layer** of the platform. Its role is to collect user input, invoke backend APIs, and present security results in a clear and actionable manner.

The frontend does **not** make security decisions and is treated as an untrusted component.

Architectural Role

Within the overall system, the frontend acts as:

- A presentation layer
- A request orchestrator
- A visualization and UX layer

All security logic, validation, and risk assessment occur in the backend.

High-Level Structure

The frontend is implemented as a **single-page application (SPA)** with:

- Component-based UI
- Client-side routing
- Centralized API client
- Token-aware request handling

It communicates with the backend exclusively over HTTPS.

API Communication Model

Centralized API Client The frontend uses a centralized API client responsible for:

- Attaching the `Authorization: Bearer <access_token>` header
- Normalizing error handling
- Triggering token refresh logic when needed
- Retrying failed requests after refresh

This prevents duplicated authentication logic across components.

Authentication Flow (Client Perspective)

1. User submits login credentials
2. Frontend sends credentials to backend authentication endpoint
3. Backend returns access and refresh tokens
4. Frontend stores access token and begins authenticated requests

The frontend treats authentication failures as session state changes.

Tool Interaction Flows

Link Analyzer

1. User enters a URL
2. Frontend performs basic format validation
3. Backend Quick Scan is invoked
4. Results are rendered
5. User may optionally trigger Deep or Custom scans

File Download Checker

1. User uploads a file
2. Frontend checks file size limits
3. File sent to backend
4. Backend returns hash and risk verdict
5. Frontend clearly labels unsafe files

History View

The frontend retrieves history entries to display:

- Tool used
- Timestamp
- Risk outcome

This allows users to review past security checks.

Error Handling Strategy

The frontend categorizes errors as:

- Validation errors (user-correctable)
- Authentication errors (session-related)
- Tool failures (external dependency issues)
- System errors (unexpected)

Errors are shown explicitly without hiding risk.

Build and Serve Models

Two supported deployment models:

Separate Deployment

- Frontend hosted independently
- Backend exposed as API service

Backend-Served Static Assets

- Frontend built into static files
- Backend serves SPA with fallback routing

Both models preserve identical security behavior.

Frontend Guarantees

The frontend guarantees:

- No security decisions
- No secret storage beyond tokens
- No direct external API calls
- Clear representation of backend results

2.0.5 Backend Overview

Entry Points

- `backend/app/main.py` creates the Flask app via `create_app(devconfig)` and registers namespaces under `/api`.
- The API documentation is served at `/api/docs` via Flask-RESTX.

App Factory and Hardening

- `backend/app/__init__.py` provides `create_app()` and request hardening.
- The app blocks traversal markers and denies access to source file patterns such as `.env`, `.py`, `.js`, `.jsx`.
- If `frontend/dist` or `frontend/build` is present, Flask serves the SPA and its assets.

Namespaces and Routes

- `auth`: signup, login, refresh, logout, logout-all, public-key.
- `user`: admin CRUD on users and self-access read access.
- `history`: admin history and per-user history.
- `tool_counter`: per-user and global usage counters.
- `tools/link-analyzer`: quick, deep, and custom scans.
- `tools/file-checker`: file upload scanning.
- `celery`: start and status for a demo task.

Data Access

- SQLAlchemy models in `backend/app/models.py`.
- Alembic migrations live in `migrations/`.

Code Alignment Notes

- Some tool routes appear in the frontend without matching backend namespaces (USB, PCAP, Audio, Behavioral Risk). TBD (verify in code).

2.0.6 Backend Overview

Purpose

The DefendX backend is the **central security decision engine** of the platform. It is responsible for authentication, authorization, tool orchestration, risk scoring, audit logging, and controlled interaction with external intelligence services.

All trust decisions are made exclusively in the backend.

Core Responsibilities

The backend is responsible for:

- Exposing a RESTful API under `/api`
- Authenticating users using JWT
- Authorizing privileged actions
- Executing security tools
- Aggregating and scoring risk signals
- Writing audit history
- Tracking tool usage
- Enforcing request-level security controls

Technology Stack

- Python
- Flask (application framework)
- Flask-RESTX (API namespaces and Swagger docs)
- JWT-based authentication
- Relational database (SQLite in development)

Application Entry Point

The backend is started via a main application entry point which:

1. Creates the Flask application using an app factory
2. Loads configuration from environment variables
3. Registers API namespaces
4. Enables API documentation at `/docs`
5. Optionally serves frontend static assets

API Structure

All API endpoints are mounted under:
/api
Namespaces are used to group functionality logically, such as:

- Authentication
- User management
- Individual security tools
- Infrastructure services (history, counters)

This modular approach improves maintainability and auditability.

Backend as a Trust Boundary

The backend is designed as the **only trusted component** in the system:

- It is the only component with database access
- It is the only component allowed to call external intelligence APIs
- It is the only component that evaluates risk

Compromise of the frontend does not compromise backend security decisions.

Error Handling Philosophy

- Authentication and validation errors fail fast
- Tool failures return partial results when possible
- External API failures increase risk instead of masking it
- Internal observability failures do not block tool execution

---

Backend Guarantees

The backend guarantees:

- No execution of untrusted content
- No silent security decisions
- No implicit trust in client input
- Deterministic and explainable outputs

Chapter 3

Security Model and Threat Analysis

3.0.1 Threat Model

Assets

- User identities, roles, and account status.
- JWT access and refresh tokens.
- URLs and file hashes submitted for inspection.
- Audit and usage logs.

Trust Boundaries

- Browser SPA is an untrusted client.
- Backend API is the primary trust enforcement point.
- External reputation providers are third-party dependencies.
- Database stores authentication and logging state.

STRIDE Analysis

Spoofing

- Risk: Credential reuse or token theft.
- Mitigation: RSA-encrypted credential fields on the client, JWT access + refresh with rotation, refresh token blocklist enforcement.

Tampering

- Risk: Client-side modification of payloads or headers.
- Mitigation: JWT verification on protected endpoints; request validation via Flask-RESTX models for tool inputs.

Repudiation

- Risk: Users deny having run tools or changed account data.
- Mitigation: `History` and `Tool_Counter` logging on tool endpoints.

Information Disclosure

- Risk: Access to source files, `.env`, or backend code via path traversal.
- Mitigation: Request path hardening and forbidden file marker checks in `backend/app/__init__.py`.

Denial of Service

- Risk: Excessive scans, file uploads, or dynamic scans.
- Mitigation: None visible in code; rate limiting and quotas are TBD (verify in code).

Elevation of Privilege

- Risk: Non-admins accessing admin routes.
- Mitigation: `require_admin()` checks JWT role claim; admin-only routes enforce role checks.

Additional Security Notes

- Link Analyzer blocks private/loopback IP targets to reduce SSRF risk (`is_public_http`).
- Refresh token revocation is enforced server-side using `RefreshToken` table.
- Authentication token state is stored server-side with expiry and revocation controls.

Open Threats / TBD

- Rate limiting, WAF, and abuse detection are not shown in code (TBD (verify in code)).
- Secure file storage for uploaded artifacts beyond temporary files is TBD (verify in code).
- Production database hardening and encryption at rest are TBD (verify in code).

3.0.2 Threat Model (STRIDE)

Spoofing

- Threat: Token impersonation
- Control: JWT signature validation, short token lifetime

Tampering

- Threat: Request manipulation
- Control: Server-side validation, immutable history

Repudiation

- Threat: User denies actions
- Control: Per-user audit logging

Information Disclosure

- Threat: Sensitive data disclosure
- Control: Hash-only processing, no raw data storage

Denial of Service

- Threat: Resource exhaustion
- Control: File size limits, tool counters

Elevation of Privilege

- Threat: Unauthorized admin access
- Control: Role-based access control

Residual Risks

- SSRF via DNS rebinding
- Reputation API dependency
- Heuristic scoring limitations

All residual risks are documented and accepted.

3.0.3 Security Controls Mapping

This is a control-to-implementation map based on the current repository. Items not evidenced in code are marked TBD.

Access Control

- Control: Role-based access enforcement for admin operations.
- Evidence: `backend/app/user.py` uses `require_admin()` and role checks; `backend/app/__init__.py` defines `require_admin()`.
- Status: Implemented.

Authentication and Session Management

- Control: Password hashing and JWT-based session tokens with refresh rotation.
- Evidence: `backend/app/auth.py`, `backend/app/models.py` (`RefreshToken`).
- Status: Implemented.

Credential Transport Security

- Control: RSA encryption for credential fields in the client, decryption server-side.
- Evidence: `frontend/src/api/encryption.js`, `backend/app/security.py`.
- Status: Implemented.

Audit Logging

- Control: Tool usage history with timestamps and user metadata.
- Evidence: `backend/app/history.py`, `backend/app/models.py`.
- Status: Implemented.

Operational Metrics

- Control: Per-user counters for tool usage.
- Evidence: `backend/app/toolcount.py`, `backend/app/models.py`.
- Status: Implemented.

Input Validation and Hardening

- Control: Request validation models and URL target filtering.
- Evidence: Flask-RESTX models in `backend/Tools/link_analyzer/route/schemas.py`, `is_public_http` in `backend/Tools/link_analyzer/route/route.py`.
- Status: Implemented.

Path Traversal Protection

- Control: Block traversal markers and forbidden file extensions.
- Evidence: `backend/app/__init__.py` request filtering.
- Status: Implemented.

Secrets Management

- Control: Environment-based secrets and API keys.
- Evidence: `backend/config.py` uses `decouple` config variables.
- Status: Partially implemented; rotation and vault integration are TBD.

Rate Limiting and Abuse Prevention

- Control: API rate limiting and quotas.
- Evidence: Not found.
- Status: TBD (verify in code).

Area	Control
Authentication	JWT access & refresh tokens
Authorization	Role-based access control
Input Validation	Server-side validation
Sensitive Data	Hash-only processing
Email Security	OAuth-scoped access
File Safety	Static analysis only
Auditability	History logging
Abuse Prevention	Tool usage counters
Privacy	Minimal retention
Failure Safety	Risk-increasing failures

3.0.4 Authentication and Authorization

Credential Encryption

- Frontend encrypts `username`, `email`, and `password` with RSA.
- Public key is fetched from `/api/auth/public-key`.
- Backend decrypts fields via `backend/app/security.py` before processing.

Signup

- Endpoint: `POST /api/auth/signup`.
- Validates required fields and uniqueness of `username` and `email`.
- Stores hashed password (`werkzeug.security.generate_password_hash`).

Login

- Endpoint: `POST /api/auth/login`.
- Enforces account suspension after three failed attempts (`User.record_failed_login`).
- Issues JWT access and refresh tokens with explicit expiry.
- Stores refresh token JTI in the `RefreshToken` table.

Refresh

- Endpoint: `POST /api/auth/refresh` (requires refresh token).
- Validates existing refresh token, checks expiry, revokes old token, and issues rotated access + refresh tokens.

Logout

- Endpoint: `POST /api/auth/logout` (requires refresh token).
- Revokes the supplied refresh token.

Logout All

- Endpoint: `POST /api/auth/logout-all`.
- Revokes all active refresh tokens for the user.

Authorization Model

- Role stored in JWT claims (`role`).
- `require_admin()` enforces admin-only access to protected routes.
- Admin CRUD operations have additional constraints (master admin restrictions in `backend/app/user.py`).

Frontend Session Handling

- Access and refresh tokens are stored in `localStorage`.
- Client token-refresh middleware renews expired access tokens and updates both tokens on rotation.
- Admin route access is based on stored user role in `localStorage`.

3.0.5 Authentication and Authorization

Overview

DefendX uses a **token-based authentication model** built on JSON Web Tokens (JWT), combined with role-based authorization controls.

Authentication and authorization are enforced **before any security tool logic executes**.

Token Types

Access Token

- Short-lived
- Sent in the `Authorization: Bearer <token>` header
- Used for all authenticated API requests
- Not stored server-side

Refresh Token

- Long-lived
- Stored in the database
- Used to obtain new access tokens
- Explicitly validated and revocable

Authentication Flow

1. User submits credentials
2. Backend validates credentials
3. Backend issues:
   - Access token
   - Refresh token
4. Frontend stores and uses access token
5. Refresh token used only for token renewal

Refresh Token Validation

On refresh requests:

1. Token is decoded
2. Token identifier is checked against the database
3. Expired or revoked tokens are rejected
4. New access token is issued

Refresh tokens are **validated server-side**, unlike access tokens.

Authorization (RBAC)

Certain endpoints require elevated privileges.

Role-Based Access Control

- User roles are embedded in JWT claims
- Decorators enforce required roles
- Authorization checks occur before route execution

Example use cases:

- Administrative actions
- User management
- System-level operations

OAuth Isolation

Any third-party OAuth integration is **not** used for application authentication.

Key properties:

- OAuth identity is separate from DefendX identity
- OAuth tokens are stored server-side
- OAuth scopes are limited to minimum required access

This prevents third-party identity from becoming an authorization authority.

Security Properties

The authentication system ensures:

- Stateless access validation
- Server-controlled session longevity
- Explicit privilege enforcement
- Clear separation of identity sources

Known Constraints

- Access tokens are not blocklisted
- Refresh tokens are not automatically rotated

These constraints are documented and accepted as part of the design.

3.0.6 Security Hardening

Request Path Hardening

- Traversal markers and forbidden file extensions are blocked in `backend/app/__init__.py`.
- Static assets are served only from `frontend/dist` or `frontend/build` when present.

Token Revocation Enforcement

- Refresh tokens are stored with JTI and checked via `@jwt.token_in_blocklist_loader` in `create_app()`.
- Access tokens are not currently blocked server-side (only refresh tokens).

URL Target Validation

- Link Analyzer validates target URLs, disallowing private/loopback/link-local IPs (`is_public_http`).

Credential Handling

- RSA-based encryption is used for credential fields to reduce exposure in transit within client environments.
- Backend rejects decryption failures for long encrypted-like strings.

CORS

- Public key endpoint has explicit origin allowlist for local dev.
- Global CORS allows all origins (`*`) for other routes (verify if this is desired for production).

File Handling

- File upload analysis writes to OS temp directory and removes files after analysis.

Security Gaps (TBD)

- Rate limiting and abuse prevention are not evident in code (TBD (verify in code)).
- CSRF protection and secure cookie strategy are not used because tokens are stored in `localStorage` (risk: XSS). TBD (verify in code).
- Production TLS termination and WAF configuration are not defined in code (TBD (verify in code)).

3.0.7 Backend Security Hardening

Purpose

This document describes **request-level and application-level security controls** enforced by the DefendX backend to reduce attack surface and prevent common exploitation techniques.

Request Filtering

A global request filter is applied before route handling.

Blocked patterns include:

- Path traversal attempts (`../`, encoded variants)
- Requests targeting sensitive directories:
  - `.git`
  - `node_modules`
- Requests for source code files:
  - `.py`
  - `.ts`
  - `.env`
  - `.json` (non-API)

Requests matching these patterns are rejected immediately.

Input Validation

All inputs are validated server-side, including:

- URLs
- File uploads
- Email identifiers
- Credential inputs

Client-side validation is advisory only.

CORS Policy

CORS is enabled to support frontend integration.

Characteristics:

- Controlled origins
- Explicit method allowances
- Header restrictions

Sensitive endpoints may have stricter rules.

Sensitive Data Handling

The backend enforces:

- No plaintext credential storage
- No raw email persistence
- No file execution
- Minimal metadata retention

Secrets are never logged or returned in responses.

External API Safety

When interacting with external services:

- Timeouts are enforced
- Failures are handled explicitly
- Missing data increases risk instead of returning “safe”

External service availability is treated as untrusted.

SSRF Considerations

URL-based tools enforce:

- Scheme restrictions (HTTP/HTTPS)
- Hostname-based public target validation

Limitations:

- Hostname resolution is not validated against private IP ranges

Mitigation relies on:

- Conservative scoring
- Deployment-level egress controls

Failure Containment

Failures in non-critical subsystems (e.g., logging) do not:

- Block security analysis
- Produce false success states

The system prefers partial visibility over total failure.

Security Posture Summary

The backend applies:

- Defense in depth
- Least privilege
- Explicit trust boundaries
- Conservative failure handling

3.0.8 Frontend Security Model

Token Storage

- Access and refresh tokens are stored in `localStorage`.
- Risk: XSS can expose tokens. No in-code mitigation beyond standard frontend practices (TBD (verify in code)).

Credential Encryption

- RSA encryption is applied to `username`, `email`, and `password` fields before submit.
- Public key is fetched from `/api/auth/public-key`.

Session Management

- Access token is attached to all API requests.
- Refresh token is used only for `/auth/refresh` and replaced on rotation.

Route Protection

- Protected routes require presence of access token.
- Admin routes require role check from user payload.

Error Handling

- API errors are parsed and surfaced to UI with descriptive messages.

CORS and API Base URL

- Default API base is `/api`, aligning with backend proxy or same-origin deployment.

Trust Assumptions

The frontend is treated as **fully untrusted**.

Assumptions:

- Client-side logic can be modified
- Client-side validation can be bypassed
- Requests may be forged or replayed

All security enforcement must occur server-side.

Token Handling

Access Tokens

- Short-lived JWTs
- Sent in Authorization headers
- Never embedded in URLs

Refresh Tokens

- Not directly accessed by frontend JavaScript when possible
- Used only for token renewal
- Validated server-side

Exact storage strategy may vary by frontend implementation.

Recommended Token Storage Strategies

In-Memory Storage

- Reduced XSS exposure
- Token lost on refresh

Secure Cookies (HTTP-only)

- Strong XSS protection
- Requires CSRF protections

LocalStorage is discouraged for sensitive tokens.

Client-Side Validation

Client-side validation is used for:

- UX improvement
- Early feedback
- Preventing obvious mistakes

Client-side validation is **never authoritative**.

CORS and Browser Security

The frontend relies on backend CORS configuration.

Security considerations:

- Strict origin configuration
- Limited allowed methods
- Controlled headers

Browser-enforced policies complement backend controls.

Error and Failure Safety

Frontend behavior under failure:

- Does not assume safety when errors occur
- Displays partial results explicitly
- Warns users when intelligence data is unavailable

Prevented Attack Classes

By design, the frontend does not allow:

- Privilege escalation via UI
- Direct external intelligence API access
- Trust decisions based on client state

Security Responsibilities Summary

Responsibility	Location
Authentication	Backend
Authorization	Backend
Risk Scoring	Backend
Input Validation	Backend

Responsibility	Location
Secret Storage	Backend
Visualization	Frontend

Security Model Summary

The frontend enhances usability while preserving strict trust boundaries.

Compromise of the frontend does not imply compromise of DefendX's security logic.

Chapter 4

Link Analyzer

4.1 Overview

The Link Analyzer evaluates a submitted URL and returns a risk-oriented result. The tool combines URL checks, reputation lookups, and optional deeper inspection paths to estimate whether a link is likely safe or suspicious.

This tool matters for social engineering defense because malicious campaigns often start with a URL. Early link triage helps users and analysts block phishing pages, malware delivery links, and impersonation attempts before account or device compromise.

Table 4.1: Link Analyzer API Endpoints

Endpoint	**Description**
`POST /quick-scan`	Fast URL inspection
`POST /deep-scan`	Deeper URL inspection with enrichment
`POST /custom-scan`	Custom scan options

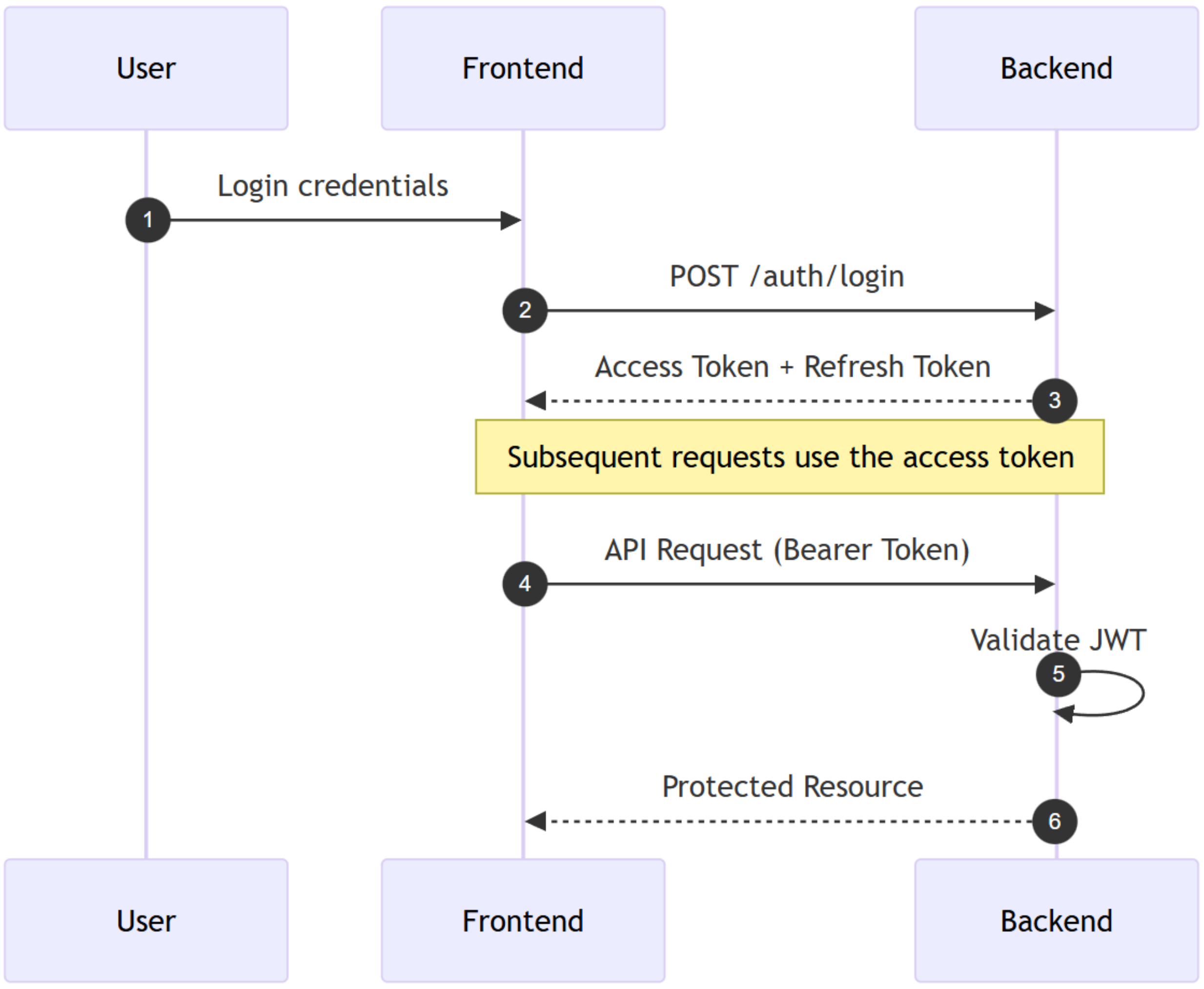


Figure 4.1: Authentication workflow and JWT-protected request sequence for Link Analyzer execution.

4.2 Scope and Assumptions

- Supported input is a URL in request payload. Scan modes include quick, deep, and custom paths.
- Out of scope: full browser sandboxing guarantees, user education workflow, and policy enforcement outside this API tool.
- Trust boundaries: user input is untrusted; external intelligence services are semi-trusted; internal scoring logic is trusted system code.

4.3 Inputs and Outputs

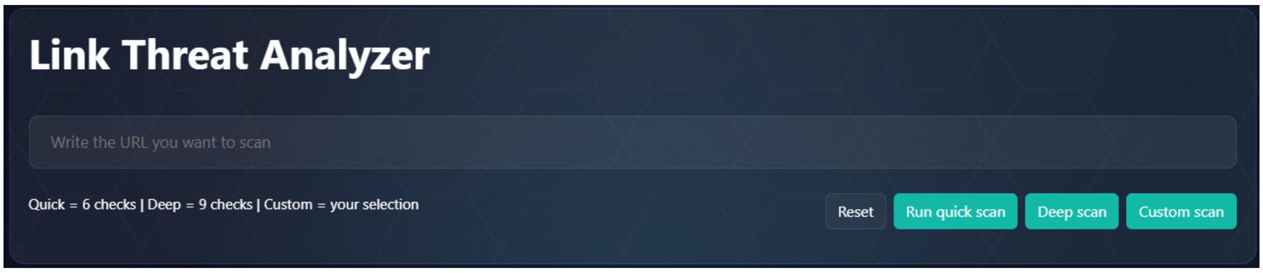
Inputs

- URL field: required, non-empty.

- Scheme expected: HTTP/HTTPS.
- URL normalization details: TBD (verify in code).
- Size and payload limits for request body: TBD (verify in code).
- Custom scan option accepts selected checks list.

Outputs

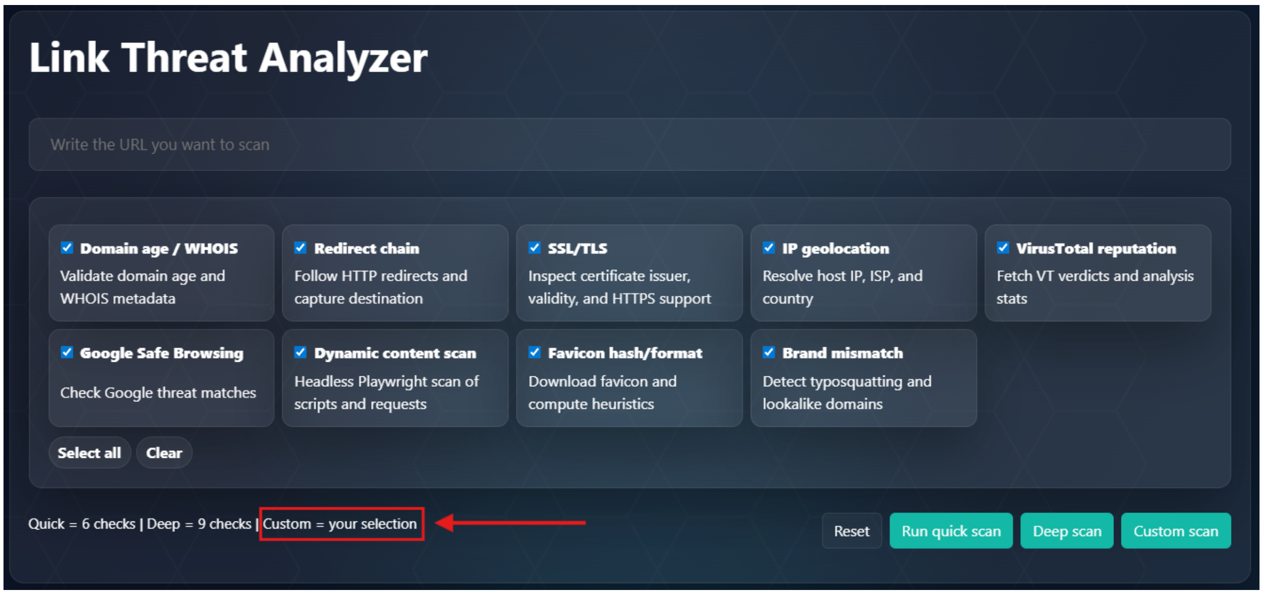
- Risk object with score, level, and reasons.
- Check-level details (domain/reputation/redirect/SSL and optional deep checks).
- Indicator flags and metadata fields: exact schema TBD (verify in code).


Link Threat Analyzer

Write the URL you want to scan

Quick = 6 checks | Deep = 9 checks | Custom = your selection

Figure 4.2: Link Analyzer user interface (URL input and scan modes).


Link Threat Analyzer

Write the URL you want to scan

☑ Domain age / WHOIS
Validate domain age and WHOIS metadata

☑ Redirect chain
Follow HTTP redirects and capture destination

☑ SSL/TLS
Inspect certificate issuer, validity, and HTTPS support

☑ IP geolocation
Resolve host IP, ISP, and country

☑ VirusTotal reputation
Fetch VT verdicts and analysis stats

☑ Google Safe Browsing
Check Google threat matches

☑ Dynamic content scan
Headless Playwright scan of scripts and requests

☑ Favicon hash/format
Download favicon and compute heuristics

☑ Brand mismatch
Detect typosquatting and lookalike domains

Quick = 6 checks | Deep = 9 checks | Custom = your selection

Figure 4.3: Link Analyzer custom scan configuration interface with selectable inspection options.

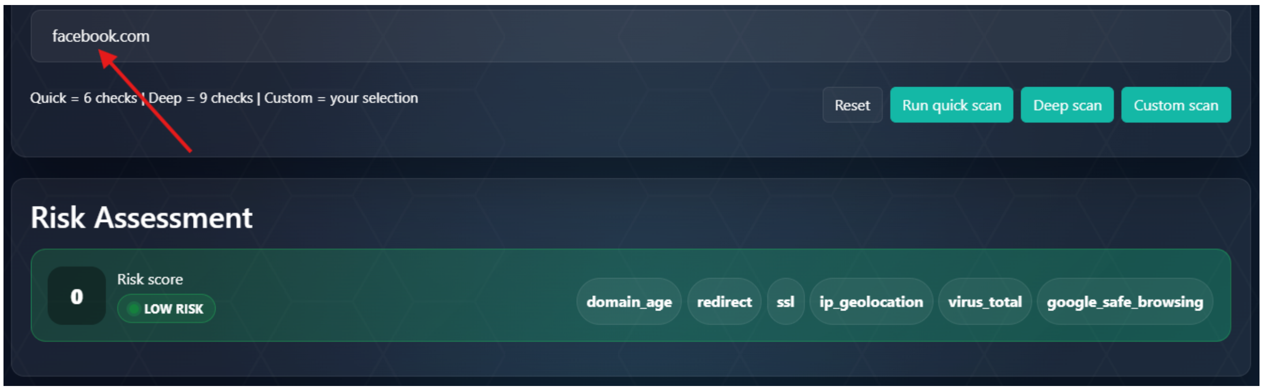


Figure 4.4: Link Analyzer result interface showing a low-risk classification.

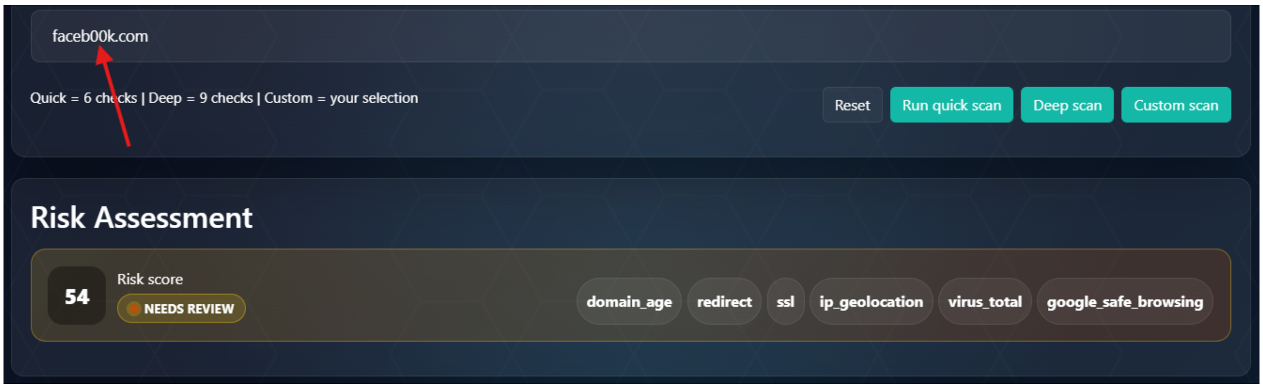
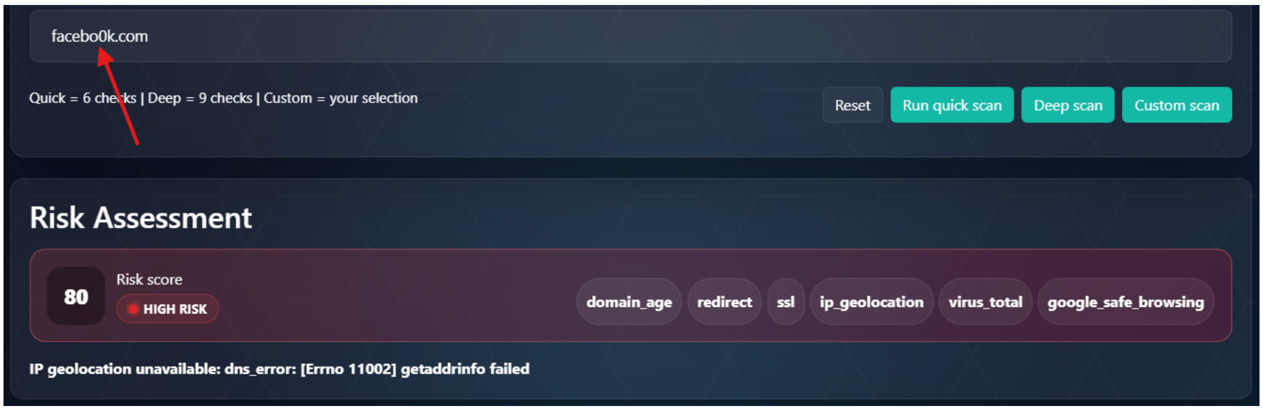
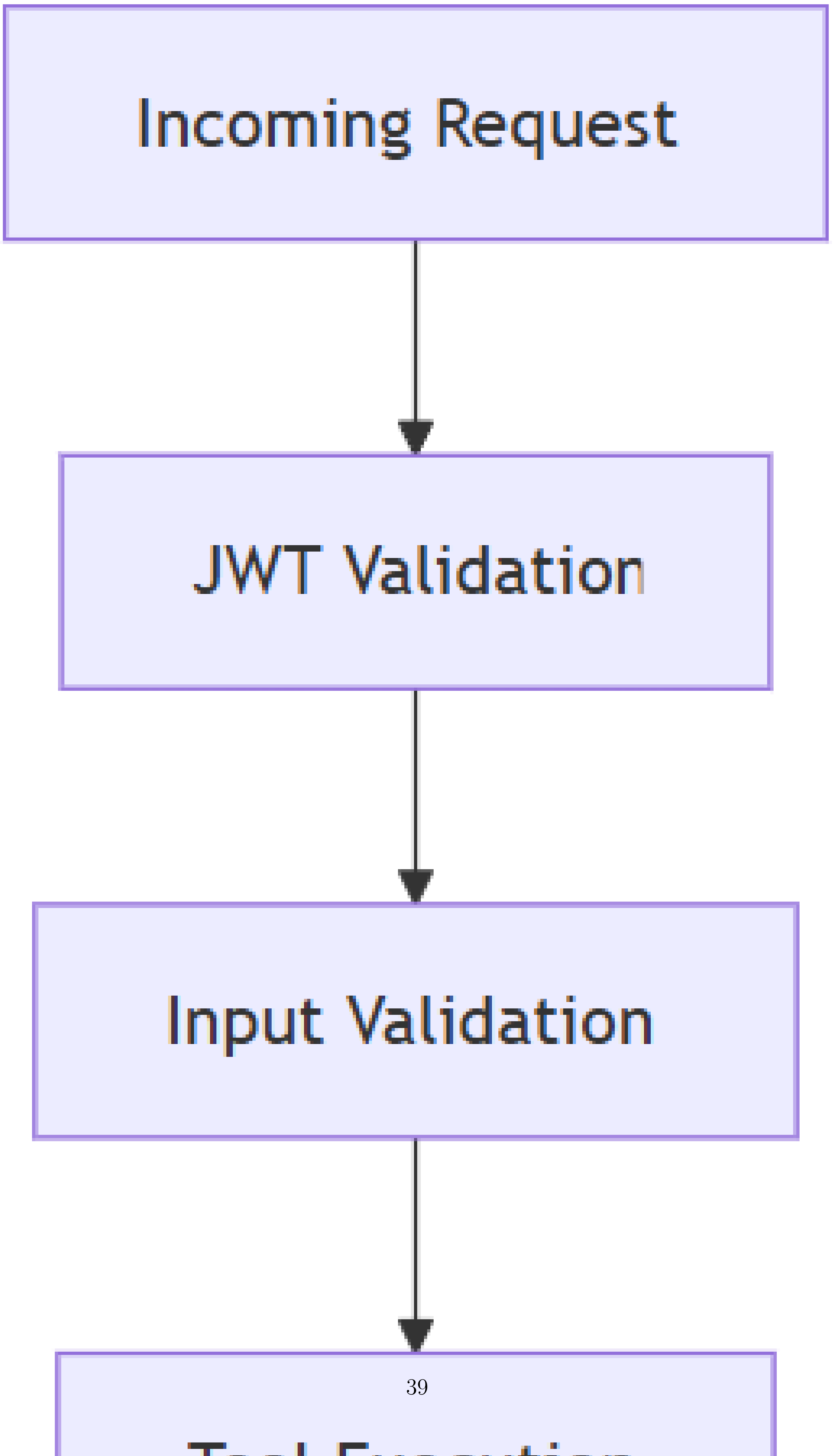


Figure 4.5: Link Analyzer result interface showing a needs-review classification.



IP geolocation unavailable: dns_error: [Errno 11002] getaddrinfo failed

Figure 4.6: Link Analyzer result interface showing a high-risk classification.


Incoming Request
JWT Validation
Input Validation

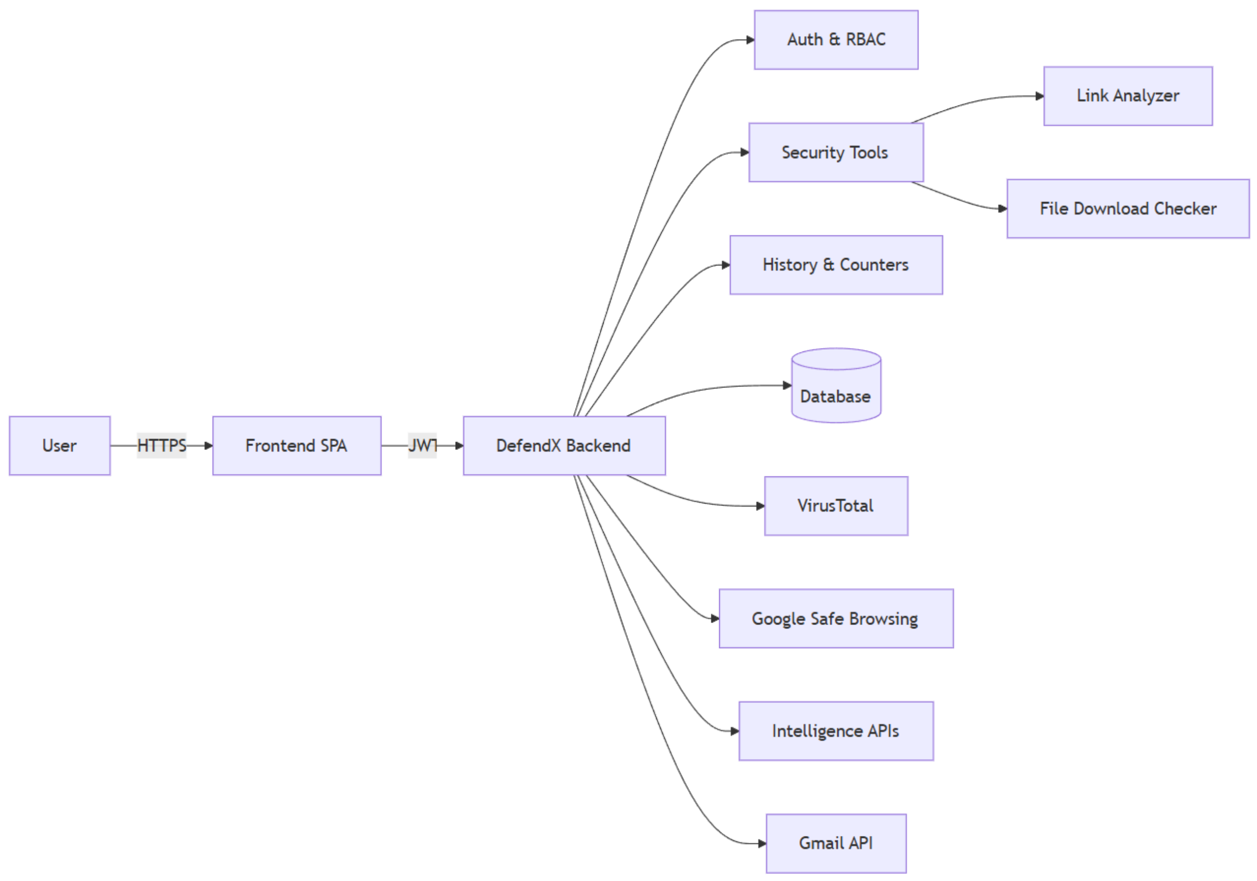


Figure 4.8: Link Analyzer: External intelligence inputs and resulting tool output indicators in the system architecture context.

4.4 Processing Pipeline

1. Accept and validate URL request.
2. Apply target safety checks before outbound processing.
3. Execute selected checks (quick, deep, or custom path).
4. Collect check outputs and compute aggregate risk.
5. Return structured response and record tool usage.

Error handling behavior

- Invalid input is rejected before analysis.
- External API timeout or failure handling policy: TBD (verify in code).
- Missing data from one check is handled conservatively in scoring.

Logging and audit

- History entries and tool counters are updated per scan.
- Sensitive payload retention policy details: TBD (verify in code).
- Authentication secrets are not expected in this tool input.

4.5 Detection Logic

- Indicators include domain age, redirect behavior, SSL status, IP/geolocation context, and reputation signals.
- Optional deep checks include dynamic behavior, favicon signals, and brand mismatch indicators.
- Score combination logic is rule-based/weighted in practice, but exact weights and thresholds are TBD (verify in code).

4.6 Security and Abuse Resistance

- Input is validated and normalized before processing.
- URL target filtering is used to reduce SSRF exposure, including private/loopback target rejection.
- Rate limiting and abuse controls at gateway/API level: TBD (verify in code).
- Tool usage counters support abuse monitoring.

4.7 Privacy Considerations

- Tool accepts URL input and returns analysis output.
- Stored versus transient fields in history are deployment-dependent: TBD (verify in code).
- Token handling relies on platform authentication layer, not this tool-specific logic.

4.8 Limitations

- Results depend on external service availability and quality.
- Dynamic checks may not cover all evasive behavior.
- SSRF protections are hostname/target-rule based and require strict maintenance.
- Risk score is an analyst aid, not proof of compromise.

4.9 Test Plan (Scalable)

Unit tests

- URL parser and validator edge cases.
- Score calculation for isolated indicator combinations.
- Check selector behavior for custom scan mode.

Integration tests

- End-to-end API response structure for each scan mode.
- External API success/failure fallback behavior.

Adversarial tests

- Redirect chains, homograph-like domains, and suspicious TLS setups.
- Known phishing URLs and benign controls.

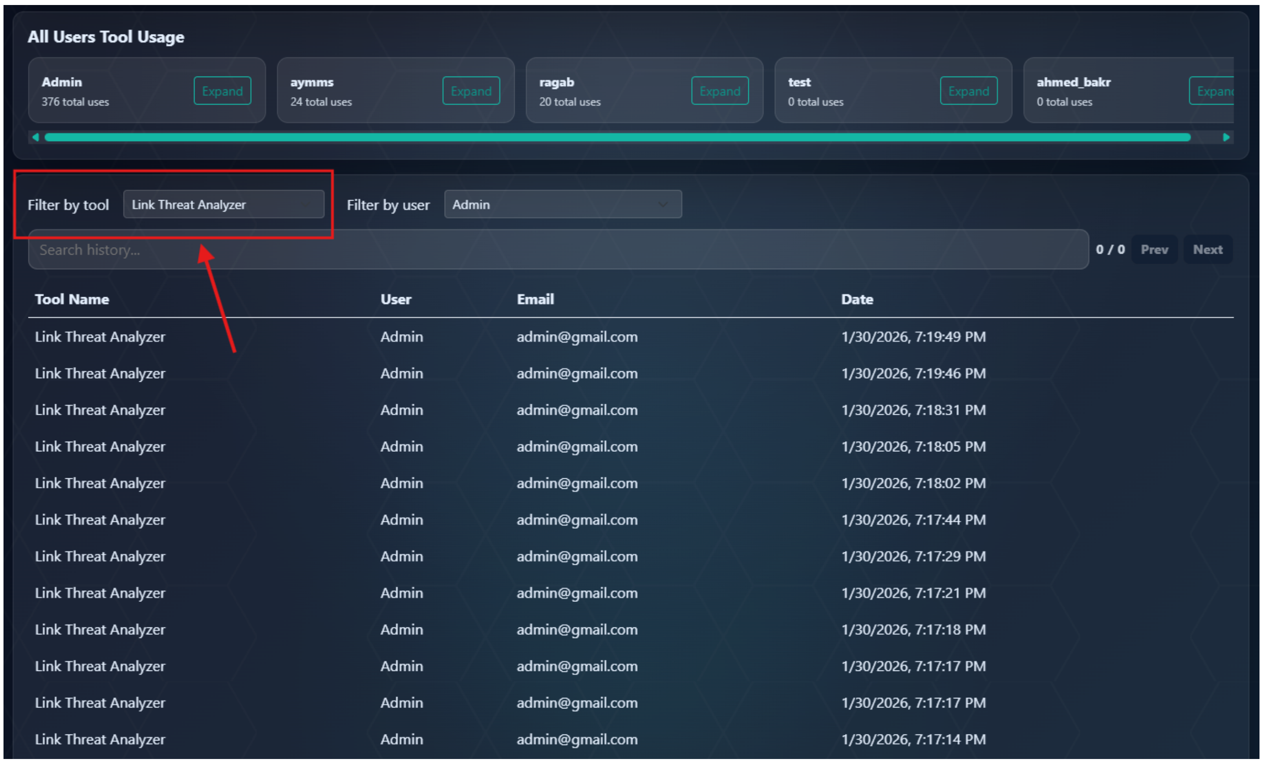
Performance tests

- Latency by scan mode.
- Timeout behavior under upstream API delays.

4.10 Operational Notes

- Monitor scan failures, timeout rates, and risk-level distributions.
- Monitor tool counter growth for usage and abuse patterns.
- Troubleshooting steps: verify API keys, check upstream service status, inspect validation failures, and review history entries.

4.10.1 Audit Trail and History Logging


All Users Tool Usage

Admin 376 total uses

aymms 24 total uses

ragab 20 total uses

test 0 total uses

ahmed_bakr 0 total uses

Filter by tool: Link Threat Analyzer

Filter by user: Admin

Search history... 0 / 0 Prev Next

Tool Name	User	Email	Date
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:19:49 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:19:46 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:18:31 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:18:05 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:18:02 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:17:44 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:17:29 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:17:21 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:17:18 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:17:17 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:17:17 PM
Link Threat Analyzer	Admin	admin@gmail.com	1/30/2026, 7:17:14 PM

Figure 4.9: Audit trail showing historical Link Analyzer executions filtered by tool.

Chapter 5

File Download Checker

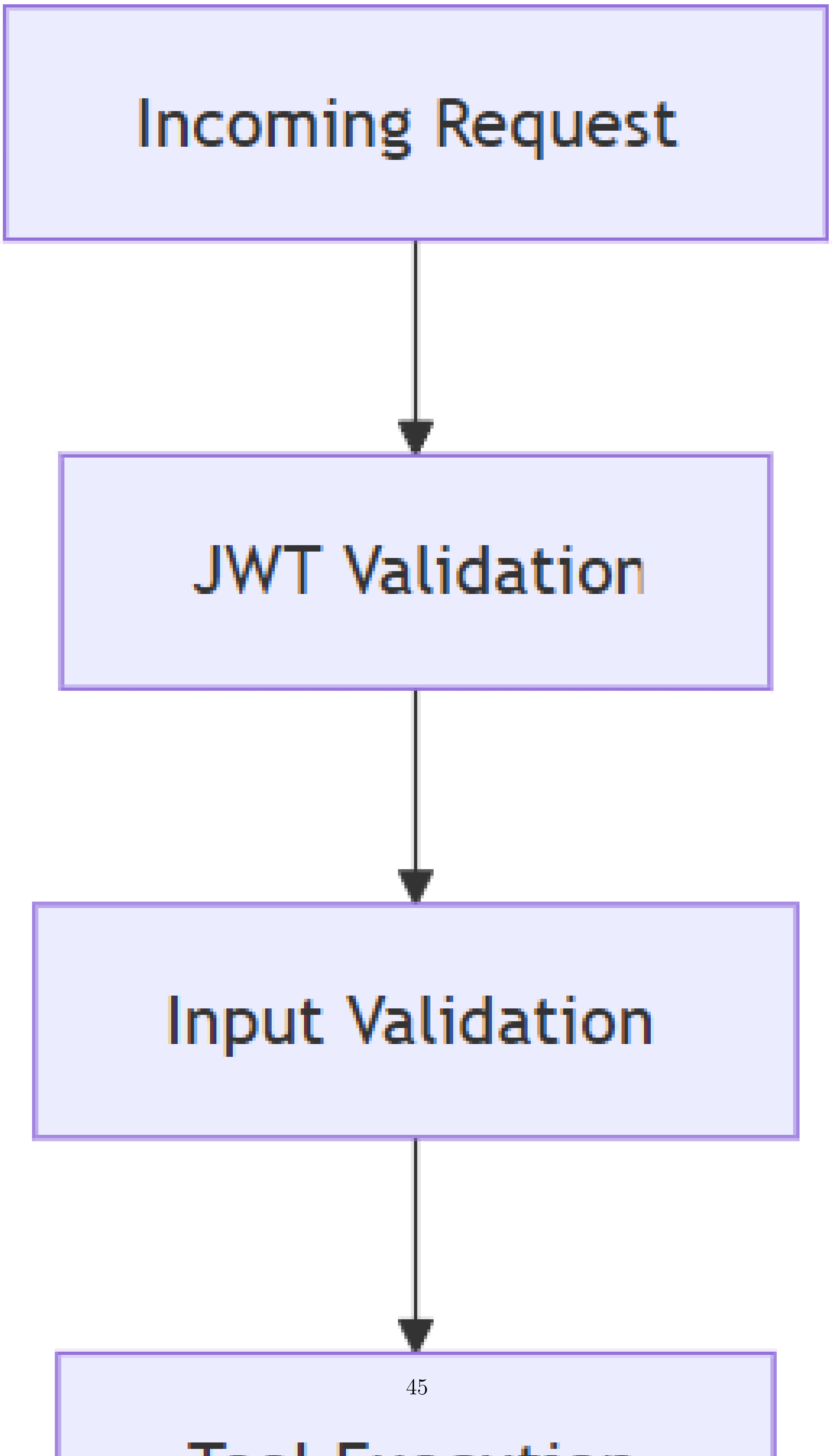
5.1 Overview

The File Download Checker evaluates uploaded files and estimates whether they are unsafe. The tool performs static inspection and reputation lookup, then returns a structured risk result.

This tool matters for social engineering defense because file-based lures are common in phishing and fraud campaigns. Static triage helps reduce exposure to malicious attachments before users open them.

Table 5.1: File Download Checker API Endpoints

Endpoint	**Description**
`POST /tools/file-checker`	Upload file for static analysis


Incoming Request
JWT Validation
Input Validation

5.2 Scope and Assumptions

- Supported input is uploaded file data through multipart form requests.
- Out of scope: dynamic sandbox execution and runtime behavior detonation.
- Trust boundaries: uploaded file is untrusted; malware intelligence service is semi-trusted; scoring and validation code are trusted system components.

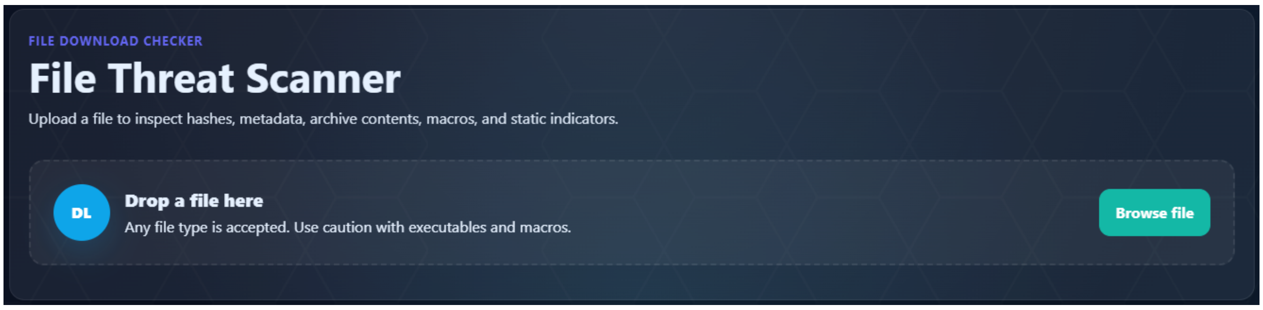
5.3 Inputs and Outputs

Inputs

- Multipart file field is required.
- File size limits are enforced, exact thresholds TBD (verify in code).
- Allowed/restricted file types list: TBD (verify in code).
- Filename normalization and sanitization are applied.

Outputs

- File metadata (name, extension, MIME type).
- Hash and static indicators (including entropy and type-specific signals where available).
- Risk result with score, level, and reasons.
- Exact response schema and optional fields: TBD (verify in code).


FILE DOWNLOAD CHECKER

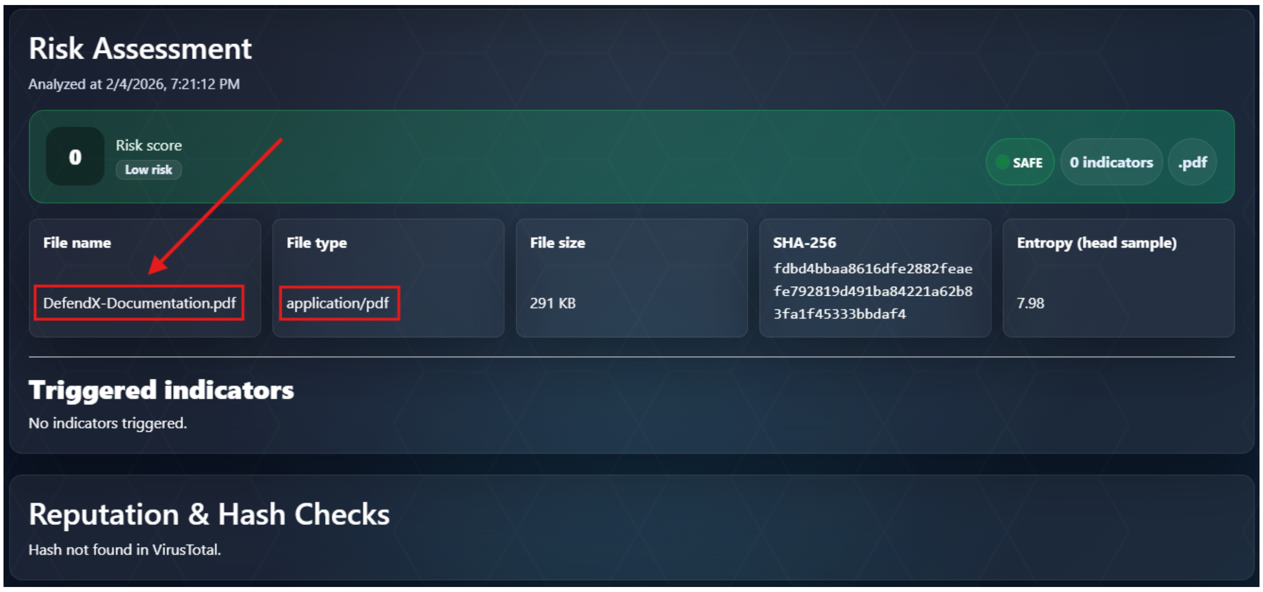
File Threat Scanner

Upload a file to inspect hashes, metadata, archive contents, macros, and static indicators.

Drop a file here

Any file type is accepted. Use caution with executables and macros.

Figure 5.2: File Download Checker user interface (upload / drop zone).



Risk Assessment

Analyzed at 2/4/2026, 7:21:12 PM

0 Risk score Low risk — SAFE | 0 indicators | .pdf

File name	File type	File size	SHA-256	Entropy (head sample)
DefendX-Documentation.pdf	application/pdf	291 KB	fdbd4bbaa8616dfe2882feaefe792819d491ba84221a62b83fa1f45333bbdaf4	7.98

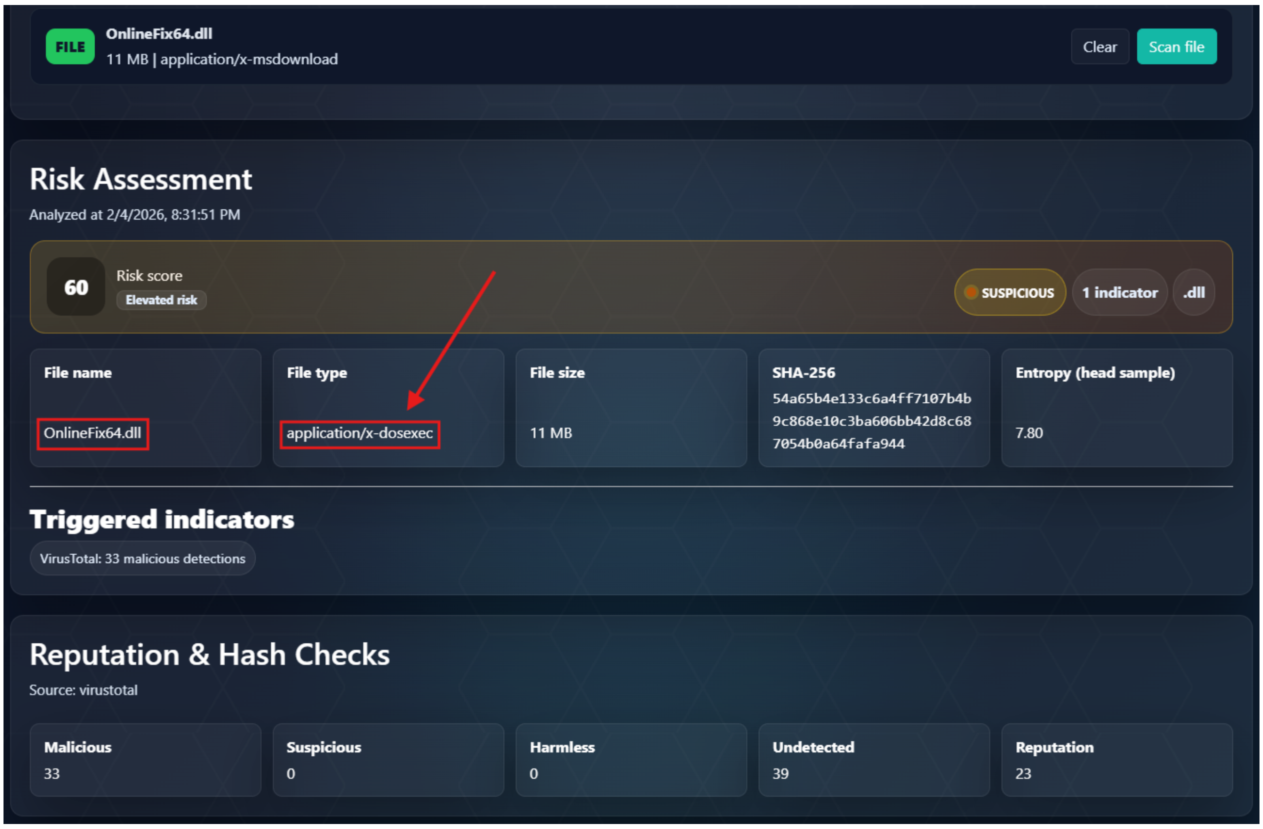
Triggered indicators

No indicators triggered.

Reputation & Hash Checks

Hash not found in VirusTotal.

Figure 5.3: File Download Checker result interface showing a low-risk classification.



FILE OnlineFix64.dll

11 MB | application/x-msdownload

Risk Assessment

Analyzed at 2/4/2026, 8:31:51 PM

60 Risk score Elevated risk — SUSPICIOUS | 1 indicator | .dll

File name	File type	File size	SHA-256	Entropy (head sample)
OnlineFix64.dll	application/x-dosexec	11 MB	54a65b4e133c6a4ff7107b4b9c868e10c3ba606bb42d8c687054b0a64fafa944	7.80

Triggered indicators

VirusTotal: 33 malicious detections

Reputation & Hash Checks

Source: virustotal

Malicious	Suspicious	Harmless	Undetected	Reputation
33	0	0	39	23

Figure 5.4: File Download Checker result interface showing a suspicious classification with triggered indicators.

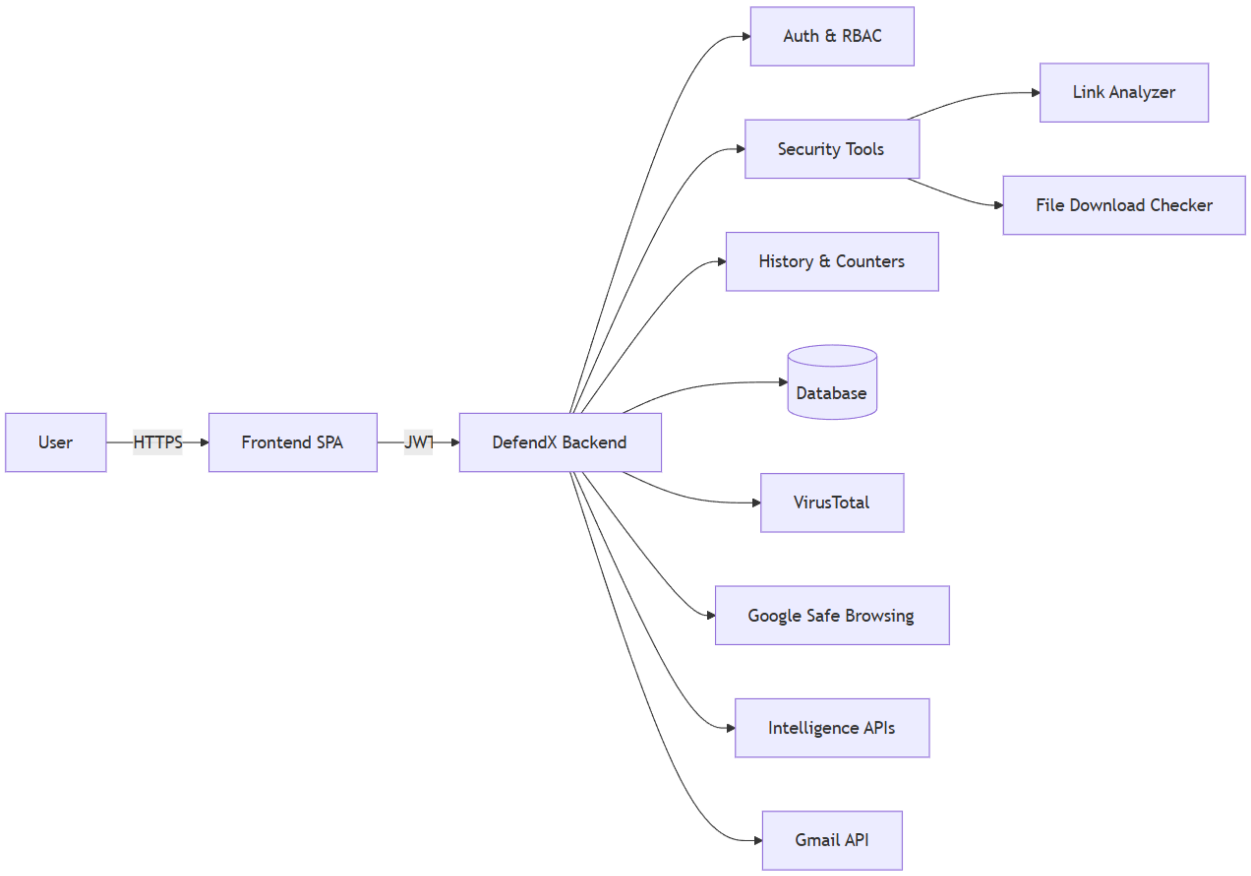
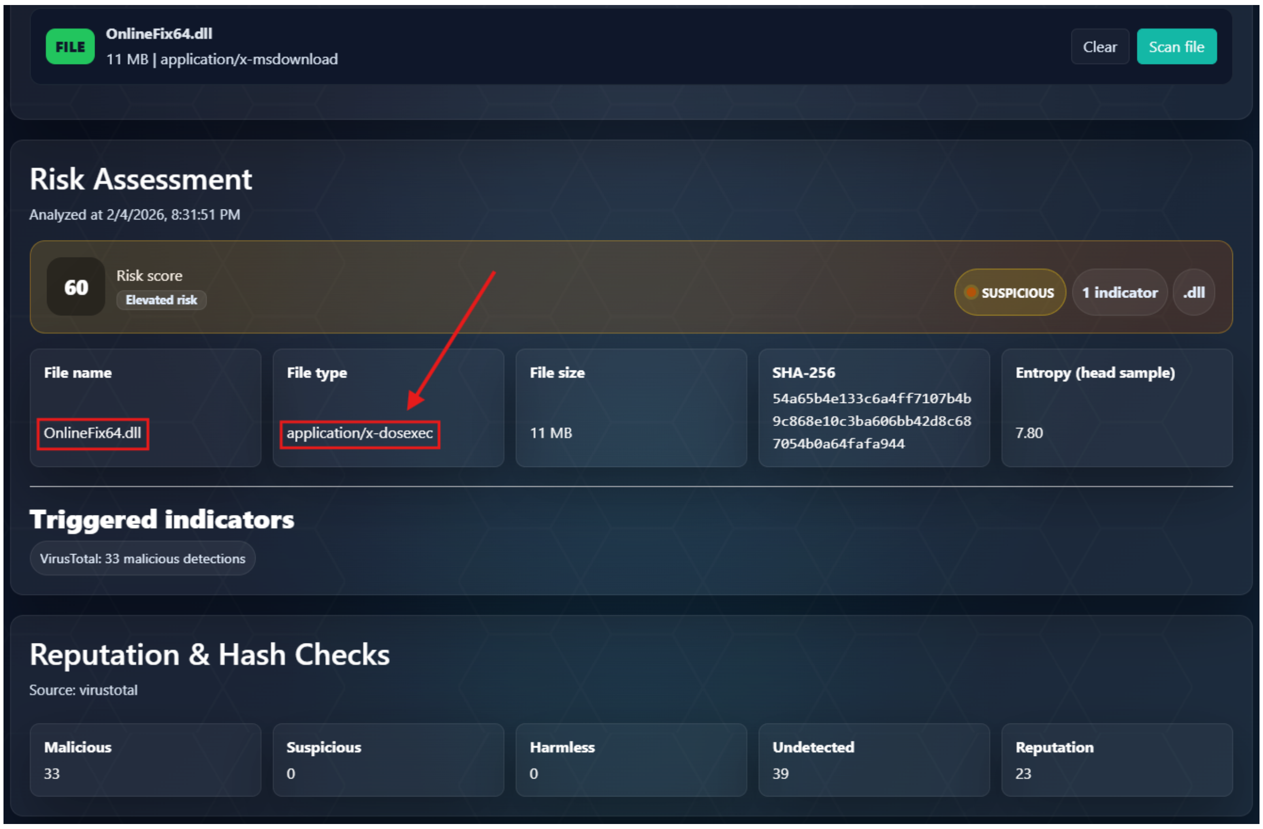


Figure 5.5: File Download Checker: Hashing and Reputation Lookup



OnlineFix64.dll
11 MB | application/x-msdownload

Risk Assessment

Analyzed at 2/4/2026, 8:31:51 PM

60 Risk score
Elevated risk

SUSPICIOUS 1 indicator .dll

File name	File type	File size	SHA-256	Entropy (head sample)
OnlineFix64.dll	application/x-dosexec	11 MB	54a65b4e133c6a4ff7107b4b9c868e10c3ba606bb42d8c687054b0a64fafa944	7.80

Triggered indicators

VirusTotal: 33 malicious detections

Reputation & Hash Checks

Source: virustotal

Malicious	Suspicious	Harmless	Undetected	Reputation
33	0	0	39	23

Figure 5.6: File Download Checker: Example Result Screen

5.4 Processing Pipeline

1. Validate upload request and file presence.
2. Sanitize metadata and apply size/type pre-checks.
3. Store file temporarily for analysis.
4. Run static analysis and compute hash.
5. Query reputation service using computed hash.
6. Aggregate indicators into risk output and return response.
7. Record usage and cleanup temporary artifacts.

Error handling behavior

- Invalid or missing file input is rejected immediately.
- Unsupported file states return controlled error output.
- External reputation API failure behavior: TBD (verify in code).
- Temporary file cleanup failure handling: TBD (verify in code).

Logging and audit

- History and tool counter are updated per request.
- Full file content should not be logged.
- Logged metadata fields are deployment-dependent: TBD (verify in code).

5.5 Detection Logic

- Indicators include MIME/extension consistency, entropy signals, static suspicious patterns, and reputation verdicts.
- Hash-based lookup contributes known-malicious or unknown status context.
- Scoring appears rule-based with thresholds; exact weighting and cutoffs are TBD (verify in code).

5.6 Security and Abuse Resistance

- Input hardening checks file presence, size, and sanitized names.
- Files are analyzed statically and are not executed.
- Abuse controls such as rate limits and per-user quotas: TBD (verify in code).
- Tool counters support anomaly tracking.

5.7 Privacy Considerations

- Uploaded files are processed for analysis and may use temporary storage.
- Retention and deletion timing depend on deployment configuration.
- Authentication tokens are handled at platform level, not by file content logic.

5.8 Limitations

- Static-only analysis can miss behavior that appears only at runtime.
- Reputation coverage is limited for new or rare files.
- Encrypted or obfuscated payloads can reduce indicator quality.
- Final score supports triage; it is not a full malware verdict.

5.9 Test Plan (Scalable)

Unit tests

- Validator behavior for file size/type/empty input.
- Hashing and metadata extraction correctness.
- Score mapping for known indicator combinations.

Integration tests

- End-to-end upload and response schema verification.
- External reputation lookup success/failure paths.

Adversarial tests

- Renamed extensions and MIME mismatch samples.

- Packed/obfuscated files and malformed payload cases.

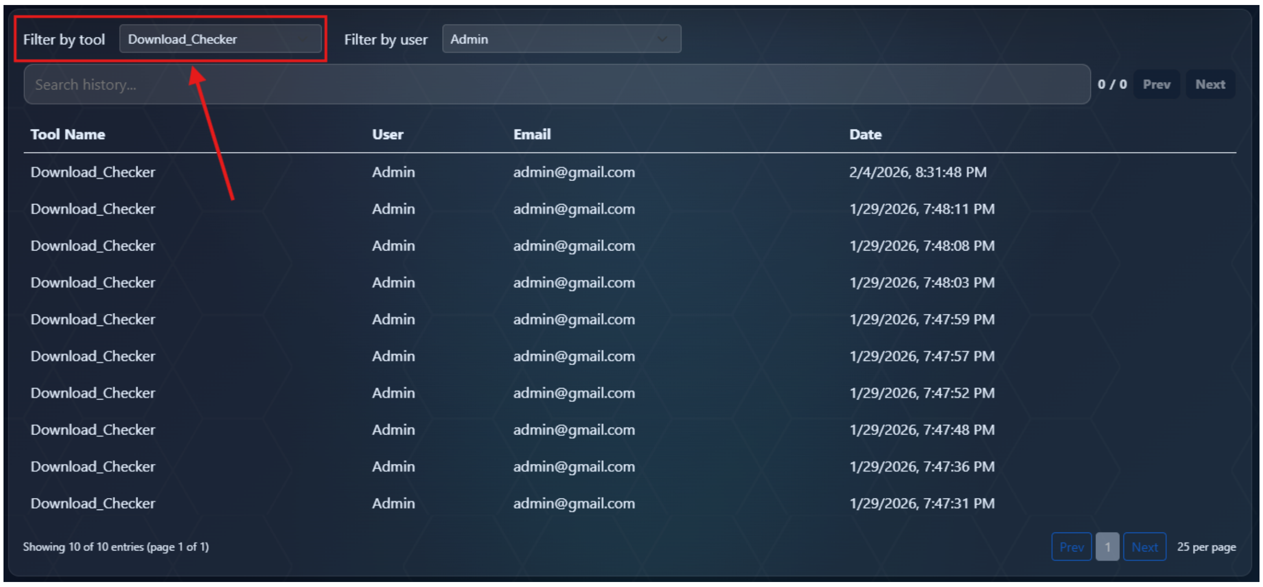
Performance tests

- Latency versus file size and type.
- Throughput and timeout behavior at concurrent load.

5.10 Operational Notes

- Monitor upload failures, analysis errors, and reputation lookup availability.
- Track tool usage counters and distribution of risk levels.
- Troubleshooting steps: verify file limits, inspect temp storage health, confirm API key validity, and review history entries.

5.10.1 Audit Trail and History Logging



Tool Name	User	Email	Date
Download_Checker	Admin	admin@gmail.com	2/4/2026, 8:31:48 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:48:11 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:48:08 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:48:03 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:47:59 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:47:57 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:47:52 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:47:48 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:47:36 PM
Download_Checker	Admin	admin@gmail.com	1/29/2026, 7:47:31 PM

Showing 10 of 10 entries (page 1 of 1)

Figure 5.7: Audit trail showing historical File Download Checker scans filtered by tool.

Chapter 6

Evaluation

This chapter presents a structured discussion of observed system behavior under representative usage conditions. The evaluation is focused on functional outcomes, consistency of risk signaling, and practical constraints of the implemented workflow.

6.1 Test Scenarios

Evaluation scenarios were defined to reflect realistic user actions for both implemented tools. For the URL Risk Analysis Tool, scenarios include benign URLs, suspicious links requiring manual review, and clearly high-risk links. For the File Download Checker, scenarios include files with low-risk characteristics and files that trigger suspicious static indicators. In each case, the objective is to verify whether the returned risk level and explanatory output remain coherent with the observed input characteristics.

6.2 Link Threat Analyzer Results

The URL Risk Analysis Tool produces tiered results that differentiate low-risk, review-required, and high-risk link submissions. Observed outputs indicate that the tool provides a consistent risk classification format, accompanied by supporting indicators that improve interpretability for end users and analysts. The result presentation supports triage by combining a categorical risk level with concise rationale rather than relying on a single opaque score.

6.3 File Download Checker Results

The File Download Checker returns structured output for uploaded files, including metadata, analysis indicators, and a summarized risk decision. Across the tested scenarios, the tool distinguishes low-risk files from suspicious submissions using static evidence available at scan time. The response format remains suitable for operational review because it links the final classification to specific observed attributes.

6.4 System Limitations

The current evaluation remains limited in scope and does not yet provide large-scale statistical validation. Results are influenced by static analysis boundaries, the availability and quality of external intelligence, and environment-specific deployment settings. Accordingly, the presented findings should be interpreted as implementation-level validation of workflow behavior rather than exhaustive performance benchmarking.

Chapter 7

Conclusion and Future Work

Conclusion

DefendX provides a practical architecture for detecting social engineering risk in user-facing workflows. The current implementation demonstrates clear request flow, security boundaries, and explainable outputs for two production-relevant tools: Link Analyzer and File Download Checker.

Future Work

7.0.1 Future Work

- Implement backend APIs for tools present in the frontend only (USB, PCAP, Audio, Behavioral Risk). TBD (verify in code).
- Add API rate limiting, request quotas, and abuse detection. TBD (verify in code).
- Provide production configuration (`prodconfig`) with secure database, secret management, and TLS guidance. TBD (verify in code).
- Harden token storage and client-side security (for example XSS protection strategy). TBD (verify in code).
- Expand audit logs with request metadata, tool inputs, and admin actions where appropriate. TBD (verify in code).
- Formalize Playwright sandboxing and resource limits for dynamic scans. TBD (verify in code).

Stronger SSRF Protections

Resolve hostnames and block private IP ranges.

Dynamic File Analysis

Optional sandbox-based execution.

Enhanced URL Signal Scoring

Expanded URL signal normalization and weighting.

Expanded Reputation Coverage

Additional defensive reputation data sources for URL and file analysis.

Configurable Risk Scoring

Externalized scoring weights.

Native Metrics

Tool latency and error metrics.

API Versioning

Introduce versioned endpoints.

7.0.2 Known Limitations

- Tool coverage mismatch: Frontend routes exist for USB Analyzer, PCAP Analyzer, Audio Analyzer, and Behavioral Risk Analyzer, but corresponding backend APIs were not found in this repository. Marked as TBD (verify in code).
- No explicit rate limiting: No rate limit or throttling middleware is present in the backend code (TBD (verify in code)).
- Production configuration: `prodconfig` is empty in `backend/config.py`. Production DB settings and secrets are TBD (verify in code).
- Playwright dependency: Link Analyzer deep/custom scans rely on Playwright. If Playwright is not installed or reachable, dynamic analysis may fail or be skipped (TBD (verify in code)).
- SQLite dev/test: Default configs are SQLite for development and testing, which is not suitable for production workload.
- Frontend token storage: Tokens are stored in `localStorage`, which is vulnerable to XSS (mitigations not shown in code).

Code Alignment Notes

- `frontend/src/api/axios.js` defaults to `/api` base URL, while root README mentions `VITE_API_URL` defaulting to `http://localhost:5000/api`. Actual default is relative `/api` (TBD: confirm intended behavior).
- `frontend/src/api/index.js` references `/tools/user-behavior/analyze`, but no backend endpoint was found (TBD (verify in code)).

Static File Analysis Only

Files are not dynamically executed or detonated.

Hostname-Based SSRF Protection

Public target validation does not resolve IP addresses.

Partial Email Header Normalization

SPF/DKIM/DMARC headers are not fully standardized.

Deployment-Dependent File Retention

Uploaded files may persist temporarily.

External Dependency Reliance

Reputation APIs may be unavailable or rate-limited.

Heuristic Risk Scoring

Risk scores are rule-based, not machine-learning-driven.

Appendix A

Other Tools (Out of Scope)

Other DefendX tools may exist in the broader platform roadmap. They are intentionally excluded from the main thesis chapters to keep this report focused on Link Analyzer and File Download Checker. Future revisions can add those tools as independent chapters once implementation and evaluation are complete.

Bibliography

[1] OWASP Foundation, “OWASP Top 10:2021,” https://owasp.org/www-project-top-ten/, 2021, accessed: 2026-02-04.