📘 ScamBane: Technical Architecture & Development Documentation

---

📌 Overview

ScamBane is a cross-platform cybersecurity system that proactively intercepts, analyzes, and blocks malicious URLs, files, and images in real time before they harm the user. It supports Android, iOS, Windows, macOS, and offers a centralized web landing portal for awareness and downloads.

🔧 Technology Stack

Layer	Tech Stack
Mobile	React Native (Android, iOS)
Desktop	Electron (macOS + Windows)
Web	React (landing + redirect)
Backend	Rust (URL/logic) + Python (AI/ML)
Notifications	Firebase Cloud Messaging

Distribution	**App Store, Play Store, Custom Website**

🛠️ Platform Capabilities

Feature	Android	iOS	Windows	macOS	Web
URL Interception	✅	🔸	✅	✅	🔸
File/Image Interception	✅	🔸	✅	✅	🔸
Real-Time Threat Analysis	✅	✅	✅	✅	✅
Block/Delete Malicious File	✅	🔸	✅	✅	N/A
Steganography Detection	✅	✅	✅	✅	✅
Push Notification Alerts	✅	✅	✅	✅	N/A

🔸 Limited support due to Apple system restrictions. iOS uses Share Extensions and Shortcuts.

✅ System-Wide URL Interception

Intercepts and analyzes links opened from browsers, messaging apps, and across the system.

✅ Smishing Defense

Scans SMS messages to extract and scan suspicious URLs in real time.

✅ File & Download Protection

Analyzes all downloads (PDFs, executables, images) before access. Malicious content is instantly blocked and deleted.

✅ Steganography Detection

Detects hidden malware in images using advanced AI techniques and lsb programming .

✅ Push Notifications

Real-time alerts on detected threats using Firebase and native notification systems.

🔄 Threat Interception Workflow

Step-by-Step:

1. **Trigger**
    - Android: Accessibility Service, FileObserver, Notification Listener
    - iOS: Share Extension, Manual Trigger
    - Desktop: Electron session hooks + File Watcher
2. **Capture**
    - Extract URL/file/image metadata
3. **Send to Backend**
    - Rust: URL scanner, phishing detection
    - Python: AI-based file and steganography analysis
4. **Evaluate**
    - `Result: { type: 'phishing' | 'malware' | 'steg' | 'safe', action: 'block' | 'allow' }`
5. **Enforce Action**
    - `block`: Delete file, block access, notify user
    - `allow`: Proceed with download or opening

6. **Notify**
    - Push alert or system toast with threat details

📱 Mobile Application: React Native

Android Highlights:

- Full system access
- Real-time interception:
    - `AccessibilityService`: monitors UI for URLs
    - `NotificationListener`: captures message previews
    - `DownloadManager` or `FileObserver`: intercepts files
- Safe/Block decision via backend

iOS Highlights:

- iOS sandbox restricts system access
- Implement via:
    - **Share Extension**: User sends a link/file/image to ScamBane
    - **Shortcuts App**: Automate "Scan before open"
    - **Custom Web Browser**: Optional secure browser in-app

Output:

- Android: `.apk` / `.aab`
- iOS: `.ipa` (via Xcode & App Store)

🔁 **Universal Link Interception**

- Intercepts links tapped from any app — WhatsApp, Telegram, SMS, Chrome, Instagram, Facebook, etc.
- ● Links are routed to Scambane's backend.
- ● If safe, opens in the browser/app.
- ● If unsafe, a block screen is shown and access is denied.
-
- 🔥 **Unsafe Link Handling (Mobile)**
- ● 🚫 Blocked access to unsafe links
- ● ⚠️ Custom block screen with explanation (e.g., phishing, smishing, malware)
- ● 🧹 Automatically deletes unsafe URLs from:
- ○ Clipboard
- ○ Message context (if permissions allow)
- ○ Logs, notifications, and local storage
-

🧠 **Smishing & Phishing Detection**

- Uses both reputation-based and AI-driven heuristics:
- ● VirusTotal, Google Safe Browsing, IPQualityScore
- ● URLHaus, PhishTank, AbuseIPDB, Maltiverse
- ● FastText NLP model (for smishing)
- ● ClamAV + YARA for file scanning
-

🔐 **Mobile User Login**

- ● Register with username + phone + OTP (static or API)
- ● JWT-based authentication
- ● Role-based access: admin, user

💻 Desktop Application: Electron

- Monitors all file downloads and web requests
- Uses `session.webRequest.onBeforeRequest` to intercept URLs
- Watches `Downloads/` folder via `fs.watch`
- Uses Node bindings to delete or allow access
- Uses custom alerts for feedback

Output:

- Windows: `.exe` via NSIS or Electron Builder
- macOS: `.dmg` Universal build

🔥 Unsafe Link Handling (Desktop)

- 🚫 Blocks opening of unsafe links
- ⚠️ Shows block screen with explanation
- 🧹 Deletes the unsafe URL from:
  - Clipboard
  - Logs and notifications (if access granted)

🌐 Web Portal: React

- Purpose: marketing, education, download access
- Routes:
  - `/` – Landing page
  - `/download` – Auto redirect to App Store/Play Store

    - `/desktop` – Desktop installers
- Uses:
    - Vercel or Netlify for deployment
    - Firebase Hosting (optional)

Mobile Redirect Logic (React):

js
CopyEdit

```
useEffect(() => {
  const ua = navigator.userAgent;
  if (/android/i.test(ua)) {
    window.location.href =
"https://play.google.com/store/apps/details?id=com.scambane";
  } else if (/iPhone|iPad|iPod/.test(ua)) {
    window.location.href = "https://apps.apple.com/app/id123456789";
  }
}, []);
```

🔧 Backend Architecture

Rust (in `rust-core/`)

- Handles:
    - URL threat detection (phishing, shortlink decoding)
    - Threat scoring
- Fast and efficient using Actix or Axum

Python (in `python-ai/`)

- ML pipelines:

    - File-based malware detection
    - Steganographic payload detection in images
- Framework: FastAPI
- Models: PyTorch or TensorFlow

API Structure:

Endpoint	Method	Description
`/scan/url`	POST	Scan and classify URL
`/scan/file`	POST	Upload and scan file
`/scan/image`	POST	Upload and detect steganography
`/report/threat`	POST	Send feedback/report

🔐 Security Considerations

- All API calls must be encrypted over HTTPS
- Token-based or device UUID authentication
- Firebase Auth or JWT for user identification
- Sandbox analysis for uploaded content
- Content is not stored unless flagged by the user

📦 CI/CD & Deployment

Component	Hosting Platform	Tool

Mobile Apps	App Store / Play Console	Fastlane
Web App	Vercel / Netlify / Firebase	GitHub Actions
Desktop Apps	S3 / GitHub Releases	Electron Builder
Backend	AWS EC2	Docker + NGINX

🔗 Final Download URLs (Sample)

Platform	Download URL
Android	`https://play.google.com/store/apps/details?id=com.scambane`
iOS/iPadOS	`https://apps.apple.com/app/id123456789`
Windows	`https://scambane.com/downloads/scambane-setup.exe`
macOS	`https://scambane.com/downloads/scambane-mac.dmg`
Web	`https://scambane.com`

🔑 Authentication, Trials, and Monetization

📱 Mobile (Android & iOS)

🔐 User Authentication

- **Signup/Login**: OTP-based via mobile number.
- **Auth Flow**:
    1. User enters mobile number.

2. Receives OTP via SMS (Firebase Auth recommended).
3. On successful verification, JWT issued for session.

🎯 Trial Limitation & Monetization

- **Free Usage**: First 3 scans allowed (URL, file/image, or download combined).
- **Post-Trial Restriction**:
  - After 3 scans: prompt user to **subscribe/purchase**.
  - Payment handled via in-app purchases (Google Play Billing, StoreKit).
- **User Scope**: Mobile use is **personal/individual only**.
- **Progress Tracking**:
  - Trial count tracked via backend with device UUID or Auth ID.
  - Reset only on account deletion.

💻 Desktop (Windows & macOS)

🔐 Trial & Licensing System

- **Free Trial**: 3 threat interception actions (URL/image/file).
- **Activation Required After Trial**:
  - Generates unique **activation key** from backend.
  - License Types:
    - **Individual** – 1 device
    - **Small Company** – up to 5 devices
    - **Enterprise** – unlimited or custom quota

- **License Verification**:
    - Enforced via backend API using hardware/device fingerprint.
    - Stored securely using encryption + local caching.

🔧 Activation Workflow

1. User installs app.
2. Gets 3 free scans/interceptions.
3. After limit, activation screen shows:
    - Input license key
    - Verify against backend
    - On success: unlock full version

Desktop:

🔐 License Key Generation & Management

🎯 License Tiers

Tier	Max Devices	Reuse Policy
Individual	1	One-time use only
Small Company	5–50	Max n devices allowed
Enterprise	51+	Unlimited reuse

🔧 Backend License Generation Logic (Python – FastAPI)

1. Generate License Key

python
CopyEdit

```
import uuid
from datetime import datetime, timedelta
from typing import Literal

LICENSE_TIERS = {
    "individual": 1,
    "small": 50,
    "enterprise": float("inf")
}

def generate_license_key(tier: Literal["individual", "small",
"enterprise"]) -> dict:
    license_key = str(uuid.uuid4()).upper()
    return {
        "key": license_key,
        "tier": tier,
        "max_activations": LICENSE_TIERS[tier],
        "used": 0,
        "created_at": datetime.utcnow().isoformat(),
        "expires_at": (datetime.utcnow() +
timedelta(days=365)).isoformat()
    }
```

2. Store in Database (Example Schema - PostgreSQL)

sql

CopyEdit

```
CREATE TABLE license_keys (
    key UUID PRIMARY KEY,
    tier TEXT CHECK (tier IN ('individual', 'small', 'enterprise')),
    max_activations INT,
    used_activations INT DEFAULT 0,
    created_at TIMESTAMP,
    expires_at TIMESTAMP
);
```

✅ License Activation API

Endpoint: `POST /activate`

Request JSON:

json
CopyEdit

```
{
  "key": "ABCD-1234-EFGH-5678",
  "device_id": "device-uuid-or-fingerprint"
}
```

Backend Logic:

1. Lookup key in DB
2. Check:
    - Key exists
    - Not expired
    - `used_activations < max_activations`
3. Store device association
4. Increment usage

Response:

- ✅ `200 OK` with license info if valid
- ❌ `403 Forbidden` if expired/used up

🔁 Activation Tracking Table

sql
CopyEdit

```
CREATE TABLE license_activations (
    id SERIAL PRIMARY KEY,
```

```
    license_key UUID REFERENCES license_keys(key),
    device_id TEXT,
    activated_at TIMESTAMP DEFAULT now(),
    UNIQUE(license_key, device_id)
);
```

🚫 Enforcement Rules

- **Individual**: 1 activation max. Once used, mark as consumed.
- **Small Company**: Allow reuse up to the assigned `max_activations`.
- **Enterprise**: No limit; just log activations for audit.

✅ Your license service using FastAPI and SQLite has been scaffolded. It includes:

- `/generate` – to create a license key for individual/small/enterprise.
- `/activate` – to activate a license with device tracking and limit enforcement.
- Database schema for both keys and activations.
- Reuse protection and expiry validation.

Code for license key:

```
from fastapi import FastAPI, HTTPException, Depends, Header
from pydantic import BaseModel
from uuid import uuid4
from datetime import datetime, timedelta
from typing import Literal, Optional
import sqlite3

app = FastAPI()

# --- Configuration ---
ADMIN_TOKEN = "supersecretadmintoken123"  # Replace with secure secret in production

# --- Database Setup (SQLite for demo) ---
```

```
conn = sqlite3.connect("licenses.db", check_same_thread=False)
cursor = conn.cursor()

cursor.execute('''
CREATE TABLE IF NOT EXISTS license_keys (
    key TEXT PRIMARY KEY,
    tier TEXT CHECK (tier IN ('individual', 'small', 'enterprise')),
    max_activations INTEGER,
    used_activations INTEGER DEFAULT 0,
    created_at TEXT,
    expires_at TEXT
)
''')

cursor.execute('''
CREATE TABLE IF NOT EXISTS license_activations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    license_key TEXT,
    device_id TEXT,
    activated_at TEXT,
    UNIQUE(license_key, device_id),
    FOREIGN KEY (license_key) REFERENCES license_keys(key)
)
''')

conn.commit()

# --- Models ---
class LicenseRequest(BaseModel):
    tier: Literal["individual", "small", "enterprise"]

class ActivationRequest(BaseModel):
    key: str
    device_id: str

# --- Helper Functions ---
LICENSE_TIERS = {
    "individual": 1,
    "small": 50,
    "enterprise": 1000000  # Arbitrary large number
}

def generate_license(tier: str) -> dict:
    license_key = str(uuid4()).upper()
```

```
    now = datetime.utcnow()
    license_data = (
        license_key,
        tier,
        LICENSE_TIERS[tier],
        0,
        now.isoformat(),
        (now + timedelta(days=365)).isoformat()
    )
    cursor.execute('''
        INSERT INTO license_keys (key, tier, max_activations, used_activations, created_at,
expires_at)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', license_data)
    conn.commit()
    return {"key": license_key, "tier": tier}

# --- Dependency ---
def verify_admin_token(x_token: Optional[str] = Header(None)):
    if x_token != ADMIN_TOKEN:
        raise HTTPException(status_code=401, detail="Unauthorized")

# --- Routes ---
@app.post("/generate")
def create_license(req: LicenseRequest, token: str = Depends(verify_admin_token)):
    return generate_license(req.tier)

@app.post("/activate")
def activate_license(req: ActivationRequest):
    cursor.execute("SELECT tier, max_activations, used_activations, expires_at FROM
license_keys WHERE key = ?", (req.key,))
    row = cursor.fetchone()
    if not row:
        raise HTTPException(status_code=404, detail="License key not found")

    tier, max_activations, used_activations, expires_at = row
    if datetime.utcnow() > datetime.fromisoformat(expires_at):
        raise HTTPException(status_code=403, detail="License expired")

    cursor.execute("SELECT COUNT(*) FROM license_activations WHERE license_key = ?",
(req.key,))
    current_activations = cursor.fetchone()[0]
    if current_activations >= max_activations:
        raise HTTPException(status_code=403, detail="Activation limit reached")
```

```
    try:
        cursor.execute('''
            INSERT INTO license_activations (license_key, device_id, activated_at)
            VALUES (?, ?, ?)
        ''', (req.key, req.device_id, datetime.utcnow().isoformat()))
        cursor.execute('''
            UPDATE license_keys SET used_activations = used_activations + 1 WHERE key = ?
        ''', (req.key,))
        conn.commit()
    except sqlite3.IntegrityError:
        raise HTTPException(status_code=400, detail="Device already activated with this license")

    return {"status": "activated", "tier": tier, "device_id": req.device_id}
```

📱 Mobile Auth & Trial Logic Plan

1. OTP Auth Flow

- Firebase Authentication (recommended) for easy SMS OTP handling.
- Flow:
    - User enters phone number.
    - Firebase sends OTP.
    - User enters OTP → success = receive UID/token.
- Store UID/token in local storage.

2. Trial Tracking (per user/device)

- Backend stores scan counts (`url_scan_count`, `file_scan_count`, etc.).
- Max combined trials = 3.

- Backend returns `{ allowed: false, reason: "limit_exceeded" }` after 3.

3. Purchase Gateway After Limit

- Redirect user to pricing page or open in-app payment modal.
- Firebase can also store user's "payment status" as a custom claim or in Firestore.

4. Mobile Client Logic

- Before scan → make `GET /user/allow-scan` API call.
- After scan → `POST /user/increment-scan`.

5. UI Logic

- If not signed in → force OTP auth.
- If scan limit exceeded → show "upgrade required" screen.

Should be built according to GDPR compliance

🧱 **DevSecOps**

- Trivy (Docker scanner)
- Gitleaks (Secrets detection)
- OWASP ZAP (Web vulnerability scanner)

🔒 **Security Enhancements**

- Let's Encrypt (SSL)
- Optional: Auth0
- Have I Been Pwned integration for breached credentials

🧠 **Implementation Advice**

● Use deep links & intent filters on mobile to capture any link.

● Reroute all intercepted content to backend before launching.

● Desktop agent acts as transparent proxy for browser/system-wide safety.

● Use Redis to cache scan results to preserve API quota.

● Sandbox unknown files before download execution.

● Encrypt all network communication (HTTPS, JWT signing).

Tools, APIs, and External Services Used in Scambane

Below is a consolidated list of the tools, APIs, and external services we will be using for **phishing**, **smishing**, **steganography**, **malware**, and various types of **cyber threats**.

---

1. Phishing Detection

- **VirusTotal API**
    - **Purpose**: Detect malicious URLs and files.
    - **API**: VirusTotal API

- **Google SafeBrowsing API**
    - **Purpose**: Check URLs for malware, phishing, or harmful content.
    - **API**: SafeBrowsing API
- **URLVoid**
    - **Purpose**: Analyze URLs for threats by checking multiple databases.

    - **API**: URLVoid

- Url Haus
- Maltiverse
- Abuse IPDB

---

2. 📩 SMS/NLP Smishing Detection

- **Twilio API**
    - **Purpose**: Parse incoming SMS messages for malicious links.
    - **API**: Twilio SMS API
- **Google SafeBrowsing (for SMS URL Threat Detection)**
    - **Purpose**: Scan URLs in SMS messages for phishing or malicious content.
    - **API**: SafeBrowsing API
- **SpamAssassin**
    - **Purpose**: Analyze SMS content for spam-like patterns indicative of smishing.
    - **Tool**: SpamAssassin

● FastText NLP

● Spamassassin

● Regex + heuristics

---

3. Steganography Detection

- **StegExpose**
    - **Purpose**: Analyze images for hidden data or malicious payloads.

    - **Tool**: StegExpose

- **OpenStego**
    - **Purpose**: Detect and decode hidden data within images.
    - **Tool**: OpenStego
- **Custom LSB (Least Significant Bit) Algorithm**
    - **Purpose**: Detect hidden messages in image pixel data.
    - **Tool**: Python (via **Pillow** library)

      Create an advanced Custom Lsb programming

---

4. Malware Detection

- **VirusTotal API**
    - **Purpose**: General malware detection for files and URLs.
    - **API**: VirusTotal API
- **ClamAV**
    - **Purpose**: Open-source antivirus engine for scanning files for malware (Trojans, Worms, etc.).
    - **Tool**: ClamAV
- **YARA**
    - **Purpose**: Create custom rules to detect patterns related to specific malware types (Trojans, Ransomware, Worms).
    - **Tool**: YARA
- **Cuckoo Sandbox**

    - **Purpose**: Dynamic analysis of suspicious files in a sandbox environment.
    - **Tool**: Cuckoo Sandbox
- **Malwarebytes API**
    - **Purpose**: Detect and block malicious files, including Trojans, Worms, and ransomware.
    - **API**: Malwarebytes API

Malware Bazaar

Hybrid analysis

---

5. Botnet and Backdoor Detection

- **Zeek (formerly Bro)**
    - **Purpose**: Network monitoring to detect traffic indicative of botnet activity and backdoor communication.
    - **Tool**: Zeek
- **Snort**
    - **Purpose**: Intrusion detection system (IDS) to detect botnet and backdoor activity based on network traffic.
    - **Tool**: Snort
- **Suricata**
    - **Purpose**: Another IDS/IPS tool that can detect and block botnet activities.
    - **Tool**: Suricata

---

6. Keyloggers Detection

- **OSSEC**
    - **Purpose**: Host-based intrusion detection system (HIDS) for detecting keyloggers and suspicious activities.
    - **Tool**: OSSEC
- **Rootkit Hunter**
    - **Purpose**: Detect rootkits, including keyloggers and backdoors.
    - **Tool**: Rootkit Hunter

---

7. Ransomware and Crimeware Detection

- **Cuckoo Sandbox**
    - **Purpose**: Detect ransomware through dynamic analysis of files.
    - **Tool**: Cuckoo Sandbox

Summary of APIs Requiring Keys

Tool/Service	API Key Required	Sign-Up Link
VirusTotal	Yes	VirusTotal API

IPQuality Score	Yes	
Google SafeBrowsing	Yes	SafeBrowsing API
urlhaus	Yes	
Twilio	Yes	Twilio API
SpamAssassin	No	SpamAssassin
CrowdStrike Falcon	Yes	CrowdStrike Falcon API

Key Considerations

- Some tools, like **YARA**, **ClamAV**, **OSSEC**, **Rootkit Hunter**, **StegExpose**, and **OpenStego**, do **not** require API keys and can be integrated directly into your system.
- **VirusTotal**, **PhishTank**, and **SafeBrowsing** are essential for URL and phishing protection, requiring API keys for real-time threat detection.
- **Twilio** and **Google SafeBrowsing** are needed for SMS and URL scanning, requiring API keys for effective integration.