A Tool for Automating Pre-Penetration Testing Reconnaissance

Muhammad Asim (23PWBCS0984)
Muhammad Sattar (23PWBCS0963)
Talal Rashid (23PWBCS1002)

Department of Computer Science
UET Peshawar

Supervisor: Ms. Kanwal Aneeq

December 24, 2025

Project Proposal Overview

The Core Concept

Problem: Manual reconnaissance is time-consuming and leads to tester fatigue.

- **Solution:** Web-based automation for OSINT and initial scanning.
- **Objective:** Combine web scraping + Specialized APIs.
- **Impact:** Shifts focus from *finding* to *addressing* vulnerabilities.
- **Deliverable:** Unified report with CIA Triad risk classification.

Feasibility Analysis

Technical

Stack: Python (Flask), SQLite.
H/W: Standard Dual-Core.
Risk: API Rate Limits (Mitigated via caching).

Economic

Cost: Minimal (Free Tier APIs).
Benefit: High reduction in man-hours per test.

Operational

Workflow: Matches Phase 1 of standard Pentesting.
Usability: Low training required.

Software Requirements (SRS)

Functional Requirements

- Input: Domain or IP Address.
- Processing: Hybrid API query + Local Nmap scan.
- Output: Web UI Dashboard + CSV/PDF Export.

Non-Functional Requirements

- **Performance:** Response within 10-20 seconds.
- **Security:** API Keys stored in `.env` variables.
- **Reliability:** Graceful error handling for API timeouts.

Architectural Approaches

We evaluated three methodologies:

- Approach A: API-Only Wrapper (Too expensive/outdated)
- Approach B: Local-Tool Aggregator (Too slow/noisy)
- **Approach C: The Hybrid Engine (Selected)**

Why Hybrid?

Combines **APIs** (VirusTotal, Hunter) for passive history with **Local Scans** (Nmap) for real-time verification. Balances speed with accuracy.

SDLC Phase 1: Planning & Analysis

Requirement Gathering

- **Stakeholders:** Pen-Testers, Security Students, Admins.
- **Gap Analysis:** Existing tools are either CLI-heavy (complex) or Enterprise (expensive).

Timeline

- Methodology: Agile.
- Duration: 10–13 Weeks.

Scope & Risk Definition

In-Scope

- Web Interface.
- OSINT Gathering.
- CIA Risk Assessment.

Out-of-Scope

- Active Exploitation (Brute force/Shells).
- Mobile App Scanning.

Risk Mitigation

Issue: API Rate Limits (HTTP 429).
Fix: Implemented retry logic and specific status code checks.

SDLC Phase 2: System Architecture

High-Level Stack

- **Frontend:** HTML/CSS + Jinja2 Templates.
- **Backend:** Python Flask Controller.
- **Database:** SQLite (Lightweight storage).

Data Flow Logic

1. User inputs Target.
2. `recon_engine.py` orchestrates parallel checks.
3. Results normalized to JSON $\rightarrow$ Stored in DB.
4. Rendered via `results.html`.

Database & UI Design

Database Schema (Table: scans)

ID	Target	Data
PK	Varchar	JSON (Raw Results)

Also tracks: Date Started, Status, Impact Summary.

UI Philosophy

- **Scannability:** Badges (Red = High Risk, Green = Safe).
- **Layout:** Card-based separation (Network vs. Email vs. Reputation).

SDLC Phase 3: Implementation Stack

Core Engine: Python 3

Key Libraries:

- `requests`: API Calls.
- `subprocess`: Running local Nmap.
- `sqlite3`: Persistence.

Integrations:

- VirusTotal (Reputation)
- Hunter.io (Email Patterns)
- IPInfo (Geolocation)

Coding Challenges & Solutions

1. Nmap Output Parsing

Issue: Complex XML output.
Solution: Custom `xml.etree.ElementTree` parser to extract clean service versions.

2. Environment Security

Issue: Hardcoded API Keys.
Solution: Used `python-dotenv` to keep credentials out of source code.

SDLC Phase 4: Testing Strategy

- **Unit Testing:** Verified JSON returns from IPInfo/Google DNS.
- **Integration Testing:** Full pipeline (Input $\rightarrow$ DB $\rightarrow$ UI).

Live Validation Examples

- *uetpeshawar.edu.pk:* Found 5 exposed emails, IIS Server 10.0.
- *google.com:* Verified malware detection logic (Mock).

Validation Results

- **Accuracy:** Successfully identified open ports (80, 443, 3306).
- **Logic:** Correctly flagged "High Risk" on critical ports (445, 3389).
- **Performance:**
  - API Scan: ∼15 seconds.
  - Full Scan (w/ Nmap): ∼2 minutes.

Prototypes (Evolution)

[Placeholder for Screenshots]

v1: Console
Raw JSON Output

v2: Basic UI
HTML Forms (No CSS)

v3: Final Dashboard
Badges, Tables, Charts

Project Outcomes

Successes

- Integrated 7 data sources.
- Created persistent scan history.
- Developed "Risk Algorithm" for executive summaries.

Lessons Learned

Relying solely on `crt.sh` was unstable (502 errors). We added strict timeout handling to prevent app hangs.

Conclusion & Future Roadmap

Summary: A functional, compliant tool that reduces manual workload by automating the "boring stuff."

Future Plans:

- **Reporting:** Generate PDF reports via Backend.
- **Scheduling:** Cron jobs for periodic scanning.
- **Auth:** User login/RBAC to protect scan data.

Thank You

Q & A

Live Demo Available Upon Request