SSH/TLS Fingerprint Visualizer — Final Project Proposal

Contributors: Jason Gonzalez & Nick Zanaboni

Overview

This project builds a reproducible system to collect, compute, and visualize SSH host-key and TLS certificate fingerprints across lab and public endpoints. The visualizer helps operators verify hosts, detect drift (key/cert changes), and explore fingerprint-based telemetry, with an emphasis on ethical, read-only data collection and clear reproducibility.

1) Problem Statement

SSH and TLS are foundational to secure communications, yet daily operations often rely on ad-hoc checks, trust-on-first-use (TOFU) prompts for SSH, and superficial certificate inspection for TLS. Key rotation, misconfiguration, or active on-path attacks can go unnoticed without baselines and change detection.

Goal: standardize fingerprint collection (SSH host keys; TLS server certificates), baseline them per host, visualize changes over time, and provide simple metrics that support verification and incident triage in a reproducible, containerized lab.

Why it matters:

- Reduces human error in TOFU workflows and host verification.
- Makes key/cert drift auditable and visible.
- Provides a safe way to learn and evaluate fingerprinting methods without intrusive scanning.

2) Threat Model

Assets.

- Authenticity of remote endpoints (SSH servers, TLS origins).
- Integrity of local baseline fingerprints and outputs.

Adversaries.

- On-path attackers (e.g., TLS interception, SSH MITM).
- Misconfigured services (accidental key reuse, weak rotation practice).
- Benign changes (routine certificate renewals) that should be distinguished from attacks.

Attack Surfaces.

- Read-only network handshakes (client connects to servers to read public material).
- Local storage of results (CSV/JSON).
- Visualization UI/CLI.

Assumptions.

- Data collection is passive or consensual; no exploitation or credential use.
- Test environment is a lab VM or limited set of well-known public sites.
- Modern stacks (OpenSSH with SHA-256 fingerprints; TLS 1.2/1.3) are in use.

Defensive/Analytic Approach.

- Compute SSH host-key and TLS certificate SHA-256 fingerprints.
- Baseline per host; detect and highlight changes across collection runs.
- (Pair uplift) Parse PCAPs to compute JA3/JA3S TLS client/server fingerprints for additional telemetry; compare distributions and detect anomalies.

---

3) Success Metrics

- **Coverage:** ≥ 50 distinct endpoints fingerprinted (mix of lab + well-known public) with repeatable results.
- **Correctness:** SSH/TLS SHA-256 fingerprints match reference tools (e.g., `ssh-keygen -lf -E sha256`, `openssl x509 -fingerprint -sha256`) for ≥ 95% of sampled hosts.
- **Drift Detection:** Any key/cert change is reported within the next collection cadence, with zero false negatives on synthetic rotations.

- **Reproducibility:** Fresh clone → `make bootstrap` → run collection scripts successfully on a clean VM/container.
- **(Pair uplift)** JA3/JA3S computed from PCAPs with ≥ 90% parser success on test captures; distributions summarized across at least two cohorts of sites.

4) Dataset / PCAP Plan

Live Queries (preferred):

- TLS: establish a client handshake, read the server certificate, compute SHA-256 fingerprint.
- SSH: retrieve host public keys (e.g., via `ssh-keyscan`) and compute SHA-256 fingerprints.
- Limit to synthetic/consented lab hosts and a small set of well-known public domains strictly for demonstration.

PCAPs (optional / pair uplift):

- Capture synthetic TLS handshakes in a lab VM using `tcpdump`.
- Parse ClientHello/ServerHello to compute JA3/JA3S.

Anonymization & Storage:

- Store results as CSV/JSON.
- Optionally mask hostnames in shared screenshots; keep raw data local only for correctness checks.
- Include a tiny, documentation-only example set in the repo; exclude large datasets to respect ethics and rate limits.

5) Risks and Ethics

- **No intrusive activity:** Only read-only handshakes and public keys/certs. No auth, fuzzing, or traffic manipulation.
- **Consent & scope:** Prefer synthetic targets or explicit consent; for public sites, keep volume minimal and within normal client behavior.

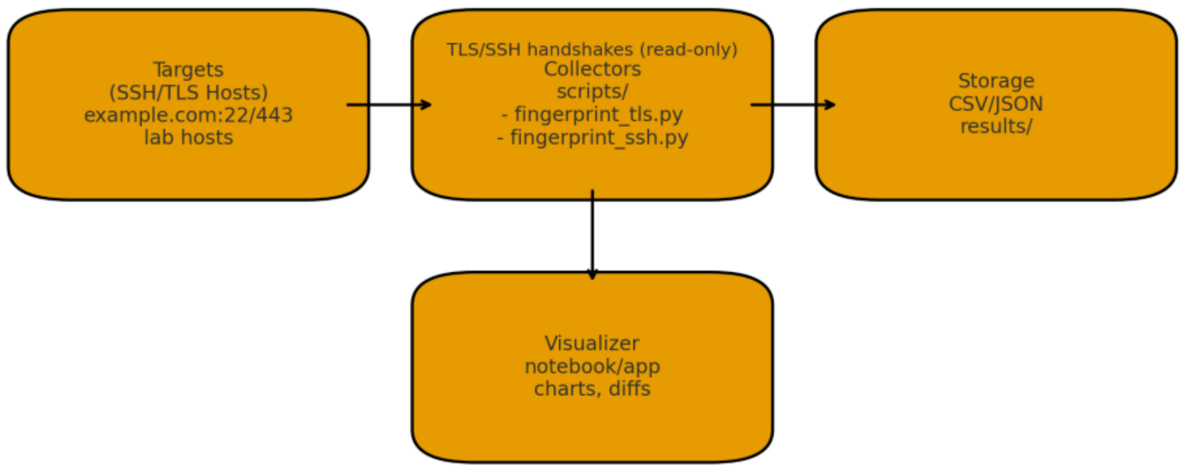
- **Privacy:** No PII collected. Do not publish sensitive internal hostnames or internal certificates.
- **Security:** Use containers/VMs; secure local outputs; allow host masking in visuals.
- **Mitigations:** Rate-limit queries; document consent; keep example datasets small and sanitized.

---

6) Architecture Diagram (text description)

Components and interactions:

1. **Collectors (scripts):**
    - `fingerprint_tls.py`: connects to `<host:port>` (e.g., `example.com:443`), retrieves the presented certificate, computes SHA-256 fingerprint (base64 or hex).
    - `fingerprint_ssh.py`: wraps `ssh-keyscan` / `ssh-keygen -E sha256` to compute SSH host-key fingerprints.
2. **Storage:**
    - Write results to CSV/JSON with timestamps and host identifiers; keep a baseline file and a latest file.
3. **Visualizer:**
    - Simple CLI or notebook to render charts/tables of fingerprints, highlight diffs vs. baseline, and summarize drift.
    - (Pair) PCAP parser adds JA3/JA3S columns and distribution views.



Runtime: `make bootstrap` sets up the Python venv and optional Docker dev container. Collectors run from the host or inside the container; visualizer reads stored results and produces charts.
