Detecting Timing Side-Channel Vulnerabilities in Web APIs Using Remote Statistical Analysis

Julia Tran

Project Proposal

Abstract

Timing side-channel vulnerabilities arise when the execution time of a web application leaks secret-dependent information, such as password correctness or partial token matches. Although well documented in cryptographic systems, timing leaks remain common in real-world web APIs due to naive string comparison logic or improper use of constant-time routines.

This project proposes a lightweight, Python-based timing side-channel detector that evaluates web endpoints using black-box probing and statistical inference. The tool systematically generates repeated requests, measures latency distributions, and applies hypothesis testing to determine whether an endpoint leaks information through timing differences.

The proposal includes a controlled experimental setup involving both vulnerable and secure server implementations, the injection of artificial noise, and analysis of the detector's sensitivity under varying sample sizes. Draft figures illustrate the planned architecture and the kind of latency-separation plots we expect to obtain. The goal is to quantify the feasibility, accuracy, and limitations of remote timing detection for web APIs.

1 Introduction

Modern web services frequently expose authentication and token-validation logic via HTTP endpoints. While these services are designed to handle secret material securely, many rely on naive string comparison or conditional logic that inadvertently leaks timing information. With the rise of microservices and distributed API architectures, timing differences once considered negligible are now observable by remote clients.

The key problem is that developers rarely have tools to automatically evaluate whether their endpoints behave in constant time. Existing static-analysis and formal methods cannot always detect runtime timing differences, and penetration testing tools typically ignore microsecond-level leaks. As a result, subtle vulnerabilities can persist in production APIs.

This project proposes a black-box measurement approach: a Python-based probing system that repeatedly queries an endpoint, records latency samples, and applies statistical tests to identify timing side channels. The core insight is that even noisy measurements can reveal meaningful timing gaps when aggregated over many samples and tested statistically. The approach is practical (it only requires HTTP access), lightweight, and applicable to real web services without source-code access.

We describe the system architecture in Section 3, outline the experimental setup and metrics in Section 4, and discuss the related work that motivates this design. The remainder of the semester will focus on implementing this design, producing real measurements, and evaluating the detector's robustness under noise.

2 Motivation

Remote timing attacks have been demonstrated across cryptographic libraries, SSH daemons, SSL/TLS stacks, and high-level web frameworks. Kocher's early work on timing attacks, and later Brumley and Boneh's remote timing attacks on OpenSSL, established that information leakage is possible even over noisy networks. More recently, practical attacks have been demonstrated against API token validation in Python, Ruby, and Node.js frameworks due to naive string comparison.

Despite this body of work, developers lack an accessible tool to test their own endpoints for timing leaks. Most existing research focuses on local CPU-level timing channels, specialized cryptographic routines, or theoretical models rather than practical web API testing. This project fills that gap by building a remote, black-box timing side-channel detector targeted at common authentication workflows (e.g., login endpoints, token verifiers) that can be integrated into a developer's test workflow.

3 Proposed Design or Architecture

High-Level Architecture

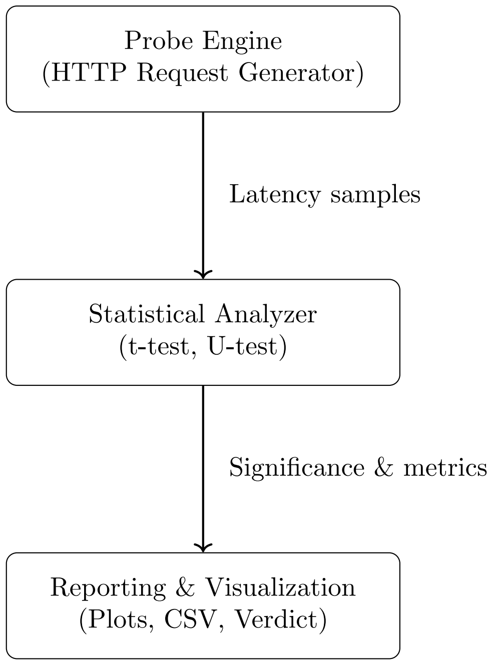

Figure 1: High-level architecture of the timing side-channel detection tool.

The tool consists of three major components (Figure 1). The *probe engine* generates HTTP requests and records high-resolution latency measurements. The *statistical analyzer* compares latency distributions for different input classes and applies hypothesis tests. Finally, the *reporting* layer produces plots and a human-readable verdict.

Probe Engine Workflow

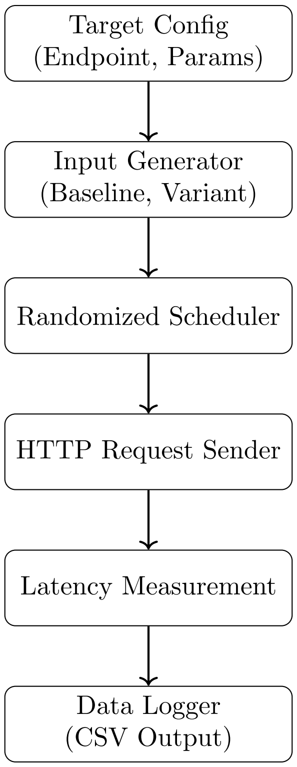

Figure 2: Probe engine workflow from configuration to logged samples.

The probe engine (Figure 2) loads a configuration describing the target endpoint, HTTP method, and parameter templates. It generates two input classes: a baseline input and one or more variant inputs designed to be "closer" to the true secret (e.g., sharing a prefix). Requests are scheduled in randomized order to avoid drift bias, and each round-trip time is measured and logged for later analysis.

Experimental Setup

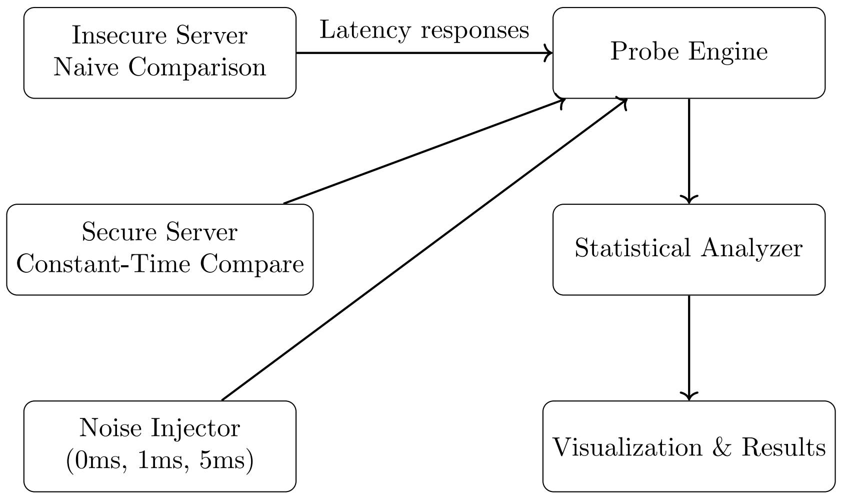

Figure 3: End-to-end experimental setup for timing side-channel analysis.

As shown in Figure 3, the experimental setup consists of two Flask-based servers: an insecure version that uses naive equality comparison for password checks, and a secure version that uses a constant-time comparison function. A noise injector module optionally adds random sleep delays to model jitter. The probe engine runs against each server configuration and produces datasets for the analyzer.

The statistical analyzer computes mean differences, variance, and p-values using Welch's t-test or a nonparametric alternative. A threshold-based rule determines whether a timing leak is likely. Key metrics include detection accuracy, false positive rate, sensitivity to noise, required sample size, and the runtime overhead of running the detector.

4 Evaluation / Experimental Results

The evaluation will follow a structured experimental methodology. First, a pair of test servers will be deployed: one using naive equality comparison for password checks, and one using a constant-time comparison function. The tool will collect latency samples for varying request counts (e.g., 200, 500, 1000, and 3000 samples per class).

Experiments will vary three primary axes:

1. type of comparison function (secure vs. insecure),
2. amount of artificial noise added to the server,
3. number of samples collected.

For each experiment, the plotted distributions and statistical test results will reveal whether timing differences are detectable. Expected trends include widening latency gaps for insecure comparisons and diminishing statistical confidence under high noise. Exceptions and anomalies such as network jitter will be documented.

Draft/Fake Result Figure

To illustrate the expected style of results, Figure 4 shows a draft plot with fake data, where the insecure endpoint exhibits a slightly higher and more spread-out latency distribution than the secure endpoint.

The evaluation will conclude with detection accuracy across conditions and identify practical sample-size thresholds. Sensitivity analysis will highlight failure cases and limitations in real-world, noisy environments.

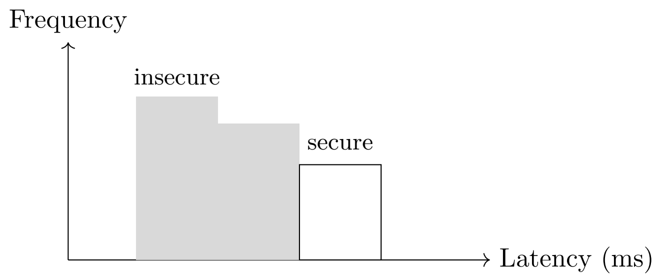

Figure 4: Draft (fake) result: illustrative latency distributions for insecure vs. secure endpoints.

5 Related Work

Foundational timing side-channel research began with Kocher, who showed that secret-dependent execution time in cryptographic algorithms can leak private keys [1]. Bortz and Boneh extended these ideas to web applications, demonstrating that HTTP response times can leak private information such as password validity and API token structure [2]. Their work provides one of the earliest formal analyses of web-based timing channels.

Morgan and Morgan later demonstrated that timing attacks remain practical in real-world web environments and introduced statistical techniques for distinguishing timing differences even over jittery networks [3]. Their methodology directly motivates the measurement and inference approach used in this project.

More recent work by Van Goethem et al. introduced *timeless timing attacks*, showing that concurrency features in HTTP/2 allow attackers to bypass traditional latency jitter and obtain high-resolution timing information remotely [4]. This highlights the continued relevance of timing channels in modern protocols.

Bozzolan et al. proposed a stochastic modeling framework for remote timing attacks, capturing the interplay between server behavior, queueing effects, and network noise [5]. This work informs our evaluation goals, especially regarding sensitivity under noise injection.

Finally, recent timing attacks on authentication protocols such as FIDO2 demonstrate that even modern security systems can contain subtle timing oracles [6]. Collectively, these works establish the feasibility and risk of timing attacks across decades of systems, motivating the need for developer-friendly testing tools. This project contributes a practical, black-box detector tailored to contemporary API authentication workflows.

6 Conclusions

This proposal outlines a lightweight, Python-based system for detecting timing side channels in web applications. By combining repeated measurement, randomized probing, and statistical hypothesis testing, the tool aims to reliably identify insecure comparison logic. The planned experiments will quantify the tool's robustness under varying noise conditions and provide insight into real-world feasibility. The resulting system will contribute a practical diagnostic tool for developers securing authentication and token-verification endpoints.

References

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in Cryptology—CRYPTO'96*. Springer, 1996, pp. 104–113.

[2] A. Bortz and D. Boneh, "Exposing private information by timing web applications," in *Proceedings of the 16th International Conference on World Wide Web*, 2007, pp. 621–628.

[3] T. Morgan and P. Morgan, "Web timing attacks made practical," Black Hat USA, Tech. Rep., 2015, https://www.blackhat.com/docs/us-15/materials/us-15-Morgan-Web-Timing-Attacks-Made-Practical-wp.pdf.

[4] T. Van Goethem, P. Papadopoulos, W. Joosen, and M. Vanhoef, "Timeless timing attacks: Exploiting concurrency to leak secrets over http/2," in *USENIX Security Symposium*, 2020, pp. 1427–1444.

[5] M. Bozzolan, K. Chen, D. Johnson, and I. Goldberg, "Stochastic models for remote timing attacks," *Proceedings on Privacy Enhancing Technologies*, vol. 2025, no. 1, pp. 123–140, 2025.

[6] M. Kepkowski, T. Gergely, and A. Skarmeta, "How not to handle keys: Timing attacks on fido authenticator privacy," *Computers & Security*, 2022.