GlyphBreaker: Advanced Red Teaming Toolkit for LLMs

Author: Ritvik Indupuri

Version: 1.0

Table of Contents

Executive Summary

GlyphBreaker is a client-side, enterprise-grade security toolkit engineered for the systematic red teaming of Large Language Models (LLMs). It provides a high-fidelity environment for security researchers and AI developers to deconstruct, probe, and audit AI systems against the official OWASP Top 10 for LLMs.

The application is founded on three core principles:

- **Structured Auditing:** Moving beyond ad-hoc prompt testing to a methodical, reproducible, and OWASP-aligned vulnerability assessment framework.
- **Comparative Analysis:** Enabling the direct, side-by-side comparison of different models (Gemini, OpenAI, local Ollama instances) against identical adversarial inputs and system configurations.
- **Deep Diagnostic Tooling:** Providing operators with not just an interaction layer, but a suite of advanced tools for prompt deconstruction, real-time threat analysis, and fine-grained control over model parameters.

This document provides a detailed technical specification of the architecture and implementation of these core principles.

System Architecture

2.1 Architectural Model

GlyphBreaker is designed as a high-performance, client-side Single-Page Application (SPA) built with React (v19) and TypeScript. This client-centric model is a deliberate security decision, ensuring that all sensitive data—including API keys, session history, and proprietary prompts—remains within the sandboxed environment of the user's browser, thereby minimizing the external attack surface.

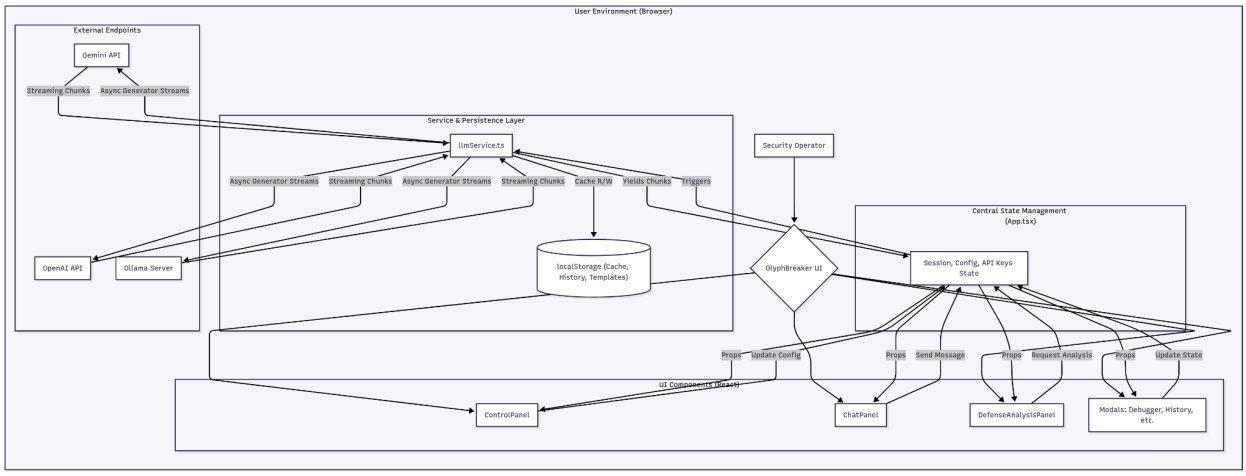
2.2 Technology Stack

- **Core Framework:** React & TypeScript
- **State Management:** Centralized React Hooks (`useState`, `useEffect`, `useCallback`, `useMemo`) for a predictable, unidirectional data flow.

- **API Communication:** Native `fetch` API and the official `@google/genai` SDK, managed within a dedicated service layer.
- **Styling:** Tailwind CSS for a utility-first, responsive design system.
- **Persistence:** Browser `localStorage` for non-sensitive session data and user-created assets.

2.3 System Architecture Diagram

Figure 1: **GlyphBreaker System Architecture & Unidirectional Data Flow**



The diagram above illustrates the application's unidirectional data flow pattern. A **Security Operator** interacts with modular **UI Components** (e.g., `ControlPanel`, `ChatPanel`). These interactions dispatch actions that trigger state updates in the central `App.tsx` component. This **Central State** is then passed down to the components as props, causing a re-render. For external communication, the state management layer invokes the `llmService.ts`, which abstracts all API logic, handles caching via `localStorage`, and streams responses from **External Endpoints** back to the central state, thus completing the reactive loop.

Core Components & Data Flow

3.1 `llmService.ts`: The Communications Core

This service acts as the central abstraction layer for all external API communication. It is engineered for abstraction, performance, and extensibility.

- **Multi-Provider Abstraction:** The `getProviderStream` async generator function serves as a factory, abstracting the significant implementation differences between the Gemini, OpenAI, and Ollama APIs. This allows the application to interact with a single, consistent interface regardless of the selected backend.
- **High-Performance Streaming:** All provider interactions are implemented using streaming to ensure a responsive UI. The Gemini SDK's `generateContentStream` method is used for optimized communication, while OpenAI and Ollama are handled via the native `fetch` API with `stream: true`, with chunks processed and decoded in real-time.
- **Intelligent Response Caching:** To accelerate iterative testing and reduce API costs, a sophisticated caching layer is implemented in `localStorage`.
    - A unique, deterministic cache key is generated by serializing the entire request context (provider, model, history, parameters).
    - On a cache hit, the full response is retrieved and programmatically split into chunks, which are yielded with a minimal delay to perfectly mimic a live API stream, ensuring a consistent user experience.
    - On a cache miss, the live stream is yielded to the UI while being concatenated and written to the cache upon completion.

3.2 Defense Analysis Engine

This feature is powered by a dedicated `gemini-2.5-flash` model acting as an AI security analyst, guided by a meticulously engineered prompt.

- **Advanced Prompt Engineering:** The analysis prompt primes the model to act as a "machine-to-machine analysis service" and an "expert AI Security Operations (AISecOps) analyst." It enforces strict output formatting (e.g., `SECTION:`, `BULLET:`) and requires the analysis to be framed

with concepts from deep learning security (e.g., "Adversarial Perturbation"), ensuring a technically rich, parsable output.

- **Memoized Parsing:** The frontend uses a `useMemo` hook to parse the streamed analysis text, ensuring that complex rendering logic only executes when the content changes, optimizing UI performance.

3.3 Interactive Prompt Debugger

This modal provides operators with crucial, pre-flight visibility into the final prompt payload sent to the LLM.

- **Vulnerability Highlighting:** Automatically highlights potential attack vectors using a curated list of injection keywords.
- **Payload Deconstruction:** Displays a simplified but structurally accurate preview of how the system prompt, conversation history, and user prompt are assembled into the final API payload.
- **Prompt Flattening:** An "Apply as System Prompt" feature allows for rapid, iterative testing by elevating a user prompt to a system-level directive.

3.4 Custom Attack Template Manager

This feature allows users to codify and persist their own testing methodologies across sessions.

- **Full CRUD Interface:** A dedicated modal provides a complete Create, Read, Update, and Delete interface.
- **Persistent Storage:** Templates are serialized to JSON and stored in `localStorage`, with state rehydrated on application load.
- **UUID for Identification:** Each template is assigned a unique v4 UUID for reliable lookups.
- **Seamless UI Integration:** User templates are dynamically rendered in the main control panel, clearly distinguished from the built-in OWASP library.

Security Implementation

4.1 Input Sanitization

All user-provided prompts are preemptively sanitized using a regular expression (`/<[^>]*>/g`) to strip any HTML-like tags before being processed or sent to an API. This serves as a critical defense-in-depth measure against injection and cross-site scripting (XSS) attacks.

4.2 Ephemeral Credential Management

OpenAI and Ollama credentials are intentionally designed to be ephemeral. They are held only in the component's volatile state and are never persisted to `localStorage`. This design requires re-entry per session but drastically reduces the risk of credential exposure from the browser's storage.

4.3 Secure Endpoint Integration

The application provides detailed, platform-specific instructions and a live status check to guide users in securely configuring the `OLLAMA_ORIGINS` environment variable, preventing common Cross-Origin Resource Sharing (CORS) errors and promoting a secure connection to local models.

Advanced Features & User Experience

- **Dynamic Content Rendering:** The `ChatPanel` includes a parser that can detect and render structured data formats, including JSON, tables, and bar charts, allowing for the testing of complex data generation tasks.
- **Session Management & Persistence:** The application supports a complete session workflow, including clearing, archiving, restoring, and renaming sessions. Nearly the entire application state is persisted to `localStorage`, allowing users to seamlessly resume their work.
- **Thematic User Interface:** The UI is built with a custom "Sentinel" theme, using a dark, high-contrast color palette to reduce eye strain. It features custom animations and interactive elements to provide clear, informative feedback during analysis.

Conclusion

GlyphBreaker is more than a user interface for LLMs; it is a purpose-built diagnostic and auditing platform designed to meet the rigorous demands of modern AI security. Its client-centric architecture prioritizes security and performance, while the abstracted service layer ensures both flexibility and extensibility.

By integrating a structured, OWASP-aligned testing methodology with advanced diagnostic tools and a multi-provider comparison framework, GlyphBreaker empowers security professionals to move beyond anecdotal testing. It provides the necessary tooling to conduct systematic, reproducible, and deeply technical vulnerability assessments. As the complexity and proliferation of LLMs continue to grow, tools like GlyphBreaker will be indispensable for ensuring these powerful systems are developed and deployed in a secure, robust, and responsible manner.