

Portfolio Security Architecture and Threat Mitigation Report

Author: Ritvik Indupuri

Affiliation: Purdue University – Cybersecurity Major

Date: October 2025

Table of Contents

1. Executive Summary
 2. System Overview
 3. Figure 1. Portfolio Security Architecture Diagram
 4. Identified Threats and Mitigation Strategies
 5. Figure 2. Threat Model and Mitigations Diagram
 6. Conclusion
 7. References
-

Executive Summary

This report provides a comprehensive overview of the security architecture and hardening process implemented in Ritvik Indupuri's personal portfolio web application and integrated AI chatbot.

The purpose of this analysis is to demonstrate adherence to enterprise-level cybersecurity practices, including defense-in-depth, zero-trust principles, and secure-by-design methodology (OWASP Foundation, 2025).

The portfolio was developed not only to highlight technical skill but also to serve as a functional example of secure application engineering. Each identified risk was analyzed, mitigated, and validated following current cybersecurity standards (Supabase, 2025; Mozilla Developer Network, 2025).

System Overview

The portfolio application consists of three core components that work together under Supabase's managed infrastructure to ensure both usability and security.

Core Components

- **Frontend (React + Supabase Auth):** Handles user sessions, secure authentication, and input validation.
 - **Backend (Supabase):** Implements Row-Level Security (RLS), role-based access control (RBAC), and database triggers to restrict access.
 - **AI Chatbot:** Provides conversational access to portfolio data while enforcing strict data isolation, sanitization, and request throttling (DOMPurify, 2025).
-

Security Architecture

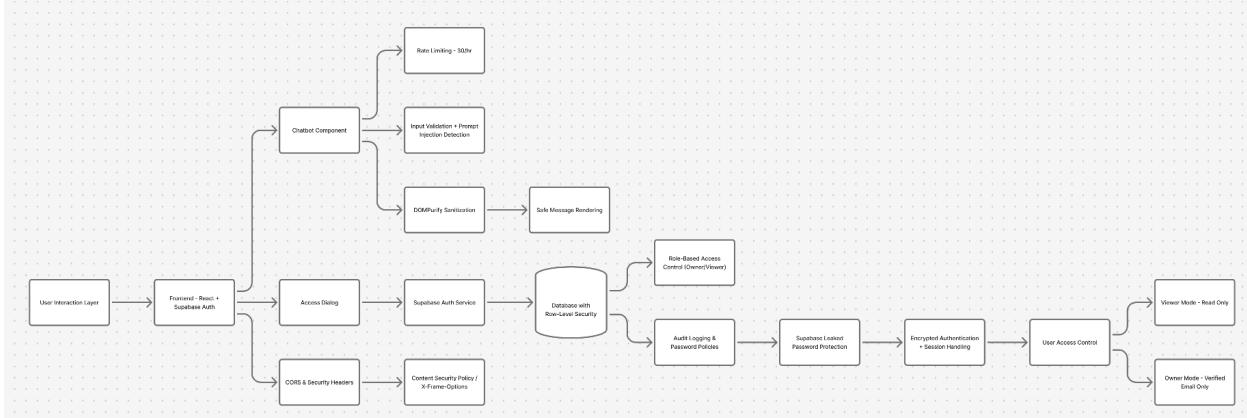


Figure 1: Portfolio Security Architecture Diagram

Note. This diagram illustrates the layered security design across the portfolio's architecture. It shows the progression from user interaction to authentication and database access, highlighting how security policies, validation layers, and sanitization steps prevent unauthorized access and data leaks.

Identified Threats and Mitigation Strategies

This section outlines each identified threat, its associated risk, and the implemented mitigation strategy. All solutions align with current OWASP Top 10 recommendations (OWASP Foundation, 2025).

1. Unauthorized Owner Access

Threat: The application previously stored an access flag in localStorage, allowing anyone with console access to gain owner privileges.

Mitigation: Implemented Supabase JWT-based authentication with server-enforced RBAC. Owner privileges are now verified via `has_role(auth.uid(), 'owner')` and assigned exclusively to the verified email address.

2. Cross-Site Scripting (XSS)

Threat: The chatbot previously displayed unsanitized HTML, allowing potential script execution.

Mitigation: Integrated DOMPurify to sanitize all chatbot messages before rendering. This ensures only safe HTML elements (bold, italic, paragraph) are permitted (DOMPurify, 2025).

3. Weak Password Protection

Threat: Original password policy allowed short or reused credentials.

Mitigation: Enforced a minimum of eight characters with uppercase, lowercase, numeric, and special character requirements. Enabled Supabase's Leaked Password Protection to reject compromised passwords (Supabase, 2025).

4. Rate Limiting and Abuse Prevention

Threat: Without rate limits, APIs were vulnerable to brute-force or spam attacks.

Mitigation: Added IP-based rate limiting through Supabase edge functions. The chatbot allows 30 requests per hour per IP, and contact forms allow 5, with automatic cleanup of expired sessions.

5. Prompt Injection and Chatbot Manipulation

Threat: Malicious users could attempt to override the chatbot's system prompt to expose hidden data.

Mitigation: Added prompt injection detection that identifies manipulation phrases and rejects them. The chatbot is limited to querying public portfolio tables only (Lovable AI Gateway, 2025).

6. Missing Security Headers and CORS Configuration

Threat: Absence of standard browser headers and open CORS policies exposed the application to cross-origin attacks.

Mitigation: Implemented strict HTTP security headers: Content-Security-Policy (CSP), X-Frame-Options, X-Content-Type-Options, and Referrer-Policy. Configured CORS to only accept requests from trusted domains (Mozilla Developer Network, 2025).

7. Session Handling and Race Conditions

Threat: Login dialogs sometimes closed before Supabase finished validating the user session.

Mitigation: Updated session-handling logic in [Index.tsx](#) to synchronize state loading and user interactions. The dialog now persists until an explicit selection is made.

8. Input Validation and Sanitization

Threat: Unvalidated user input could lead to injection or data corruption.

Mitigation: Added Zod schema validation for all form inputs, ensuring type safety and proper formatting before database entry (Zod, 2025).

9. Chatbot Data Exposure

Threat: Without scope restrictions, the chatbot could access internal or private data.

Mitigation: Limited chatbot queries to read-only public portfolio data. No personal data is accessible, and API calls run under least-privilege policies enforced by RLS (Supabase, 2025).

10. Development Server Exposure

Threat: The development server was accessible from external interfaces.

Mitigation: Restricted the Vite development host to [127.0.0.1](#), ensuring only local access during development.

Threat Model and Mitigations

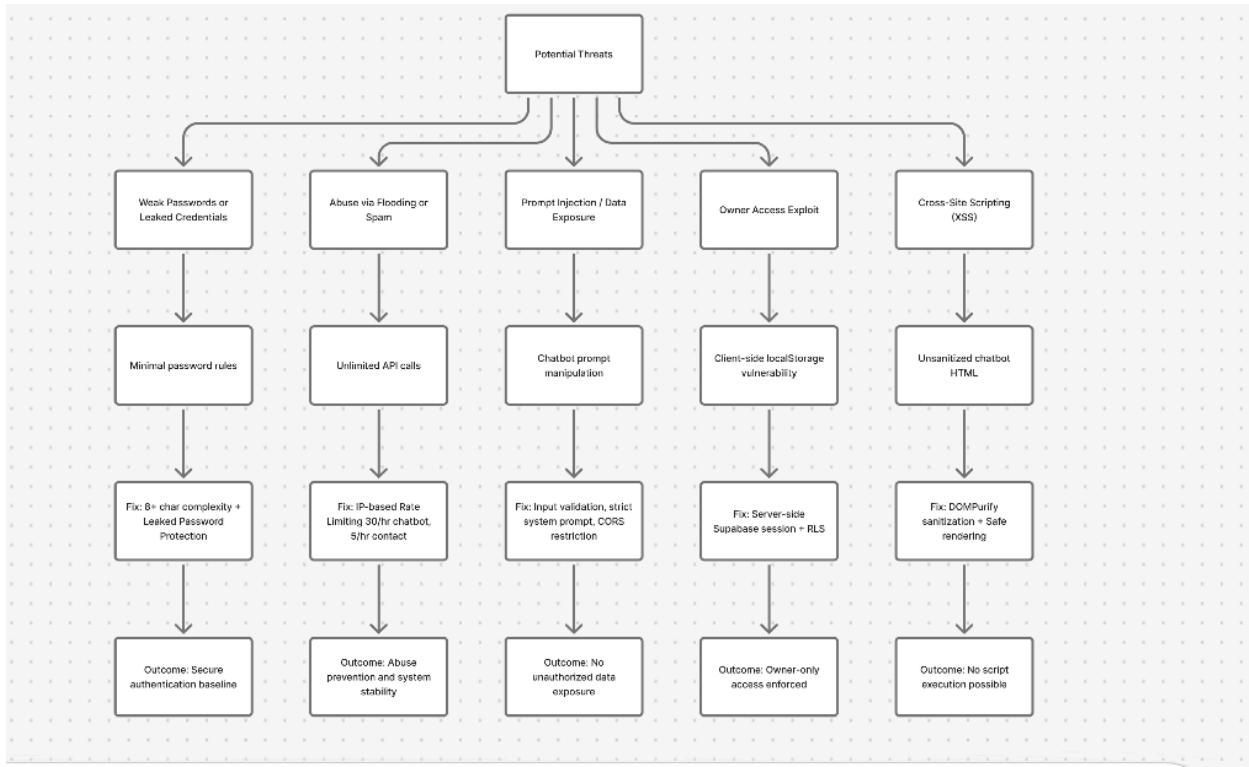


Figure 2: Threat Model and Mitigations Diagram

Note. This figure presents a one-to-one mapping between identified vulnerabilities and their corresponding mitigations. Each node illustrates the potential exploit, the fix implemented, and the resulting security posture, showing how defense-in-depth principles apply across the portfolio environment.

Conclusion

Following a complete security review, the portfolio system now conforms to modern enterprise cybersecurity standards.

Through layered controls—authentication, authorization, input validation, network security, and safe rendering—the system achieves a mature, defense-in-depth posture that minimizes common attack vectors (OWASP Foundation, 2025).

The final architecture demonstrates:

- **Secure authentication and access control** using Supabase JWTs and RLS.
- **Full input validation and sanitization** across all user data.
- **Isolated chatbot data retrieval** with prompt-injection protection.
- **Hardened infrastructure** through headers, CORS, and rate limiting.

This project serves as both a personal portfolio and a practical demonstration of applied cybersecurity engineering, illustrating how creativity and security can be integrated by design.

References

DOMPurify. (2025). *DOMPurify: Client-side HTML sanitization library*. GitHub.
<https://github.com/cure53/DOMPurify>

Lovable AI Gateway. (2025). *Secure AI integration and API gateway documentation*.
<https://lovable.dev/docs>

Mozilla Developer Network. (2025). *HTTP security headers and CORS best practices*.
<https://developer.mozilla.org/>

OWASP Foundation. (2025). *OWASP Top 10: The ten most critical web application security risks*.
<https://owasp.org/www-project-top-ten/>

Supabase. (2025). *Supabase authentication and Row-Level Security (RLS) documentation*.
<https://supabase.com/docs>

Zod. (2025). *Zod schema validation library documentation*. <https://zod.dev>

