Attack Vectors, Risk Levels, and Security Mitigation Strategies

Critical Risk Attack Vectors:

1. GitHub Token/Credential Theft

- **Risk Level: Critical**
- **Attack Vector:** Exposure of GitHub private key or access tokens stored in environment variables
- **Impact:** Complete compromise of the GitHub integration, allowing attackers to access repositories, merge code, or impersonate the bot
- **Mitigation Strategies:**
    - Store credentials in a secure environment variable management system
    - Implement key rotation policies (30-90 day intervals)
    - Use secret scanning tools to detect accidental credential commits
    - Apply the principle of least privilege to GitHub App permissions

2. Webhook Endpoint Exploitation

- **Risk Level: Critical**
- **Attack Vector:** Attackers sending malicious or spoofed webhook payloads
- **Impact:** Unauthorized system actions, potential server compromise
- **Mitigation Strategies:**
    - Implement strict webhook signature validation using HMAC
    - Set up rate limiting to prevent brute force attacks
    - Apply input validation on all webhook payloads
    - Use a dedicated webhook path with low predictability

High Risk Attack Vectors:

3. LLM Prompt Injection

- **Risk Level: High**
- **Attack Vector:** Malicious code in PRs designed to manipulate the LLM
- **Impact:** Misleading reviews, potential data exfiltration, or system manipulation
- **Mitigation Strategies:**
    - Implement prompt hardening techniques with clear boundaries
    - Add system instructions that explicitly prevent prompt injection
    - Sanitize code before sending to LLM
    - Review and filter LLM outputs before posting to GitHub

4. Man-in-the-Middle (MITM) on Development Tunnel

- **Risk Level: High**
- **Attack Vector:** Interception of traffic through NGROK or similar tunneling services
- **Impact:** Exposure of sensitive code or credentials
- **Mitigation Strategies:**
    - Use HTTPS for all communications
    - Limit NGROK to development environments only
    - Move to a production-grade hosting solution for deployment
    - Implement additional application-level encryption for sensitive data

5. Environment Variable Leakage

- **Risk Level: High**
- **Attack Vector:** Accidental exposure of secrets through logs, error messages, or stack traces
- **Impact:** Credential theft and system compromise
- **Mitigation Strategies:**
    - Implement secure logging practices that redact sensitive data
    - Use a dedicated secret management solution
    - Configure proper error handling that doesn't reveal system internals
    - Regular audits of logs and error outputs

Medium Risk Attack Vectors:

6. Denial of Service (DoS)

- **Risk Level: Medium**
- **Attack Vector:** Overwhelming the system with numerous PR events or webhook calls
- **Impact:** Service unavailability, potential resource exhaustion
- **Mitigation Strategies:**
    - Implement rate limiting at the API level
    - Set concurrency limits on PR processing
    - Configure resource quotas for LLM API calls
    - Design the system to gracefully handle high loads

7. Data Exposure via Storage

- **Risk Level: Medium**
- **Attack Vector:** Unauthorized access to stored PR review data or model information
- **Impact:** Exposure of proprietary code or internal feedback
- **Mitigation Strategies:**
    - Encrypt sensitive data at rest

    - Implement proper access controls for database
    - Consider data anonymization for stored code snippets
    - Regular security audits of storage systems

8. Dependency Supply Chain Attacks

- **Risk Level: Medium**
- **Attack Vector:** Compromised npm packages or other dependencies
- **Impact:** Introduction of malicious code into your system
- **Mitigation Strategies:**
    - Regularly update dependencies
    - Use dependency scanning tools
    - Implement lockfiles and integrity checks
    - Consider using a private registry for critical dependencies

Low Risk Attack Vectors:

9. Insecure Direct Object References

- **Risk Level: Low**
- **Attack Vector:** Improper access controls on database objects or API endpoints
- **Impact:** Unauthorized access to data
- **Mitigation Strategies:**
    - Implement proper authentication and authorization
    - Use indirect references where possible
    - Validate all user inputs against expected patterns
    - Apply the principle of least privilege

10. Information Disclosure via Verbose Error Messages

- **Risk Level: Low**
- **Attack Vector:** Detailed error messages revealing system internals
- **Impact:** Information gathering for more targeted attacks
- **Mitigation Strategies:**
    - Custom error handling that limits information disclosure
    - Different error handling for production vs. development
    - Centralized logging with proper sanitization
    - Regular review of error messages

Security Implementation Priorities

For your Secure Software Design project, I recommend this implementation order:

1. **Webhook Security (Critical)**
    - Implement signature validation
    - Add rate limiting
    - Validate webhook payloads
2. **Credential Protection (Critical)**
    - Secure environment variable handling
    - Implement least privilege access
    - Set up secret rotation
3. **LLM Security (High)**
    - Implement prompt hardening
    - Add input sanitization
    - Filter LLM outputs
4. **Infrastructure Security (Medium)**
    - Move from NGROK to production hosting
    - Secure database access
    - Implement proper logging
5. **Ongoing Security Practices (Medium)**
    - Regular dependency updates
    - Security code reviews
    - System monitoring