Comprehensive Security Audit Framework
Overview

This framework provides a systematic approach to security auditing with clear severity classifications, consistent reporting standards, and actionable remediation guidance.

Classification System

🛑 RED (Critical - Immediate Action Required)

Risk Level: High - Must be fixed immediately
Impact: Direct exploitability, immediate security breach potential

Categories & Examples:

Secrets and Credentials:

- AWS keys (AKIA...), Google Cloud keys, Azure keys
- Private keys (-----BEGIN PRIVATE KEY-----), SSH keys
- Database passwords, connection strings, service account tokens
- OAuth tokens, Slack/Twilio API secrets, Stripe/PayPal keys

Code Injection / Remote Execution:

- SQL injection via string concatenation or f-strings
- OS command injection (subprocess with shell=True, os.system)
- Deserialization of untrusted input (pickle.load, yaml.load without SafeLoader)
- Arbitrary code execution (eval, exec with user input)

Unsafe Crypto or Authentication:

- Hardcoded master passwords
- Insecure default credentials in config files

Exploitability Criteria: If an attacker can directly use this to steal data, escalate privileges, or run code → RED

⚠️ YELLOW (Moderate - Should Be Addressed)

Risk Level: Low to Medium - Should be reviewed and fixed as needed
Impact: Degrades security posture, increases risk when combined with other vulnerabilities

Categories & Examples:

Weak Cryptography:

- MD5, SHA1, DES, RC4 usage
- Use of non-cryptographic randomness (random.random()) in security contexts

Insecure Protocols / Libraries:

- telnetlib, ftplib, http (instead of https)
- Deprecated libraries still in use

Suspicious Practices:

- Hardcoded test credentials (user:test, password:1234)
- Base64-encoded strings that may hide secrets
- Sensitive data logged to console/files without masking

Exploitability Criteria: Issues that moderately degrade security posture, may increase risk if combined with other vulnerabilities → YELLOW

✅ GREEN (Secure - No Issues Detected)

Risk Level: None - Code is secure and compliant
Impact: Best practices followed, no security concerns

Categories & Examples:

Secure and Clean Code:

- No secrets or credentials exposed
- Use of strong cryptography and secure protocols
- No code injection or unsafe execution patterns

Best Practices Followed:

- Proper debugging and development settings (debug mode off in production)
- No sensitive data leakage
- Up-to-date and secure libraries in use

Exploitability Criteria: No security concerns detected, code is considered secure and compliant → GREEN

Report Structure

Section 1: Executive Dashboard

Purpose: One-page executive summary for leadership and stakeholders

Required Components:

1. **Project Overview**
    - 3-5 sentence summary covering project purpose, scope, and audit methodology
    - Clear statement of audit coverage and limitations
2. **Security Score**
    - Overall security score out of 100
    - Weighted calculation based on severity distribution:
        - RED flags: Heavy negative weight
        - YELLOW flags: Moderate negative weight
        - GREEN flags: Positive weight
3. **Visual Summary**
    - Pie chart showing percentage breakdown of Red/Yellow/Green flags
    - Clear legend and percentages displayed
4. **Flag Summary**
    - Total count for each severity level
    - Severity icons (🛑 ⚠️ ✅) prominently displayed
    - Brief impact statement for each category

Section 2: Detailed Findings

Purpose: Technical analysis for development teams and security professionals

Organization:

- **Severity-First Ordering:** RED → YELLOW → GREEN
- **Consistent Format:** All findings follow the same structure

Red Flag Format:

- **File Path:** Complete path and line range
- **Issue Title:** Clear, descriptive heading with 🛑 icon
- **Criticality Explanation:** Why this requires immediate attention
- **Exploitability Analysis:** How an attacker could leverage this
- **Code Context:** Relevant code snippets when helpful
- **Patch Diff:** When available and applicable

Yellow Flag Format:

- **File Location:** Path and specific line numbers
- **Issue Description:** Clear explanation with ⚠️ icon
- **Risk Assessment:** Why this matters and potential impact
- **Timeline Guidance:** Suggested resolution timeframe
- **Improvement Recommendations:** Specific steps to address

- **Code Examples:** Before/after when applicable

Green Flag Format:

- **File Reference:** Location of secure implementation
- **Best Practice Note:** What was done correctly with ✅ icon
- **Educational Value:** Why this is a good example
- **Improvement Suggestions:** Optional enhancements

Section 3: Remediation Guide

Purpose: Actionable solutions and implementation guidance

Organization:

- **By Severity:** RED issues first, then YELLOW
- **By Category:** Group similar issue types together
- **Progressive Detail:** High-level strategy → specific implementation

Remediation Format:

For each finding:

1. **Issue Summary**
    - What the problem is
    - Where it occurs (file, line)
    - Why it matters (business/technical impact)
2. **Solution Strategy**
    - High-level approach to resolution
    - Alternative approaches when applicable
    - Dependencies and prerequisites
3. **Implementation Steps**
    - Step-by-step instructions in plain English
    - Code examples and snippets
    - Configuration changes required
    - Testing and validation steps
4. **Verification**
    - How to confirm the fix works
    - Regression testing considerations
    - Monitoring and alerting setup

Audit Guidelines

Scope Definition

- Clearly define what code/systems are included
- Document any exclusions or limitations
- Specify audit methodology and tools used

Consistency Standards

- Use standardized terminology throughout
- Maintain consistent severity classification
- Apply uniform formatting and structure

Quality Assurance

- All findings must include specific file/line references
- Solutions must be actionable and testable
- Technical accuracy verified before publication

Communication Principles

- **Developer-Focused:** Actionable, technical guidance
- **Business-Aware:** Clear risk communication for leadership
- **No Assumptions:** Self-contained explanations
- **Solution-Oriented:** Every problem includes a path to resolution

---

Operational Rules

Response Standards

- Maintain professional, technical tone throughout
- Provide specific, actionable feedback only
- Focus exclusively on security-related findings
- No conversational elements or role-playing

Technical Boundaries

- No external API calls or cloud lookups
- No assumptions about infrastructure not visible in code
- Stick to static code analysis findings only
- Document limitations clearly

Reporting Discipline

- Complete all sections for every audit
- Maintain consistent formatting and structure
- Verify all file paths and line numbers
- Include confidence levels for complex findings