Coding & Development / Code Reviews

Flag common security issues (SQL injection, XSS, hardcoded secrets, unsafe deserialization) with fix recommendations.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Security Audits, Vulnerability Detection, Code Reviews
Updated: May 2026
Why This Prompt Exists
Most security vulnerabilities are introduced in code reviews that miss them.

You get:

- SQL injection risks from string concatenation
- XSS vulnerabilities from unescaped user input
- hardcoded API keys and passwords in source code
- unsafe deserialization of untrusted data
- missing authentication or authorization checks

But security is not optional.

It is a code review requirement.

- SQL injection: use parameterized queries, not string concatenation
- XSS: escape output, use CSP headers, sanitize input
- Secrets: never hardcode — use environment variables or secret manager
- Deserialization: validate input, avoid unsafe deserializers
- Auth: check authentication and authorization on every request

Without security review, vulnerabilities ship to production.

This framework forces AI to flag security issues.

The Prompt

```
Assume the role of a security engineer who reviews code for
vulnerabilities.

Your task is to flag security issues and recommend fixes.

Generate:

1. VULNERABILITY LIST
   - Type of vulnerability (SQL Injection, XSS, Hardcoded Secret,
etc.)
   - Location (file and line)

2. FOR EACH VULNERABILITY:
   - Why it's dangerous
   - Exploit scenario (how an attacker could use it)

3. FIX RECOMMENDATION
   - Specific code change
   - Secure alternative

4. SEVERITY RATING
   - CRITICAL / HIGH / MEDIUM / LOW

5. EVIDENCE (why you flagged it)
```

```
INPUTS:

Code (paste):
[PASTE CODE]

Language/Framework:
[PYTHON / JAVASCRIPT / JAVA / C# / PHP / OTHER]

Context:
[WEB APP / API / CLI / DESKTOP / MOBILE]

Known Threat Model (if any):
[E.G., "Authenticated users only," "Public-facing API"]

RULES:
- Flag string concatenation in SQL queries (SQL injection)
- Flag unescaped user input in HTML (XSS)
- Flag hardcoded secrets (API keys, passwords, tokens)
- Flag unsafe deserialization (pickle, eval, unserialize)
- Flag missing authentication/authorization checks
- Flag use of deprecated or known-vulnerable functions
- Provide specific fix code, not general advice
- Rate severity based on exploitability and impact
```

How To Use It

- Run this on every pull request that touches authentication, database, or user input.
- CRITICAL vulnerabilities must be fixed before merging.
- Never hardcode secrets — use environment variables or a secret manager.
- Use parameterized queries for all database operations (never string concatenation).

- Escape all user-generated content before rendering in HTML.

Example Input

Code:
def get_user(request):
user_id = request.GET.get('id')
query = f"SELECT * FROM users WHERE id = {user_id}"
cursor.execute(query)
return cursor.fetchone()

Language/Framework: PYTHON / Django

Context: WEB API (public-facing)

Known Threat Model: Public API, no authentication on this endpoint

Why It Works
Most security vulnerabilities are missed in code reviews.

This framework improves outcomes by forcing:

- vulnerability identification (detection)
- exploit scenario explanation (understanding)
- specific fix code (action)
- severity rating (prioritization)
- evidence (justification)

Great security reviews don't just find bugs — they prevent breaches.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, code review frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The Code Quality Reviewer