/audit-context-building

Source: `~/.claude/skills/tob-audit-context-building/skills/audit-context-building/SKILL.md`

---

name: audit-context-building
description: Enables ultra-granular, line-by-line code analysis to build deep architectural context before vulnerability or bug finding.

Deep Context Builder Skill (Ultra-Granular Pure Context Mode)

1. Purpose

This skill governs **how Claude thinks** during the context-building phase of an audit.

When active, Claude will:

- Perform **line-by-line / block-by-block** code analysis by default.
- Apply **First Principles**, **5 Whys**, and **5 Hows** at micro scale.
- Continuously link insights → functions → modules → entire system.
- Maintain a stable, explicit mental model that evolves with new evidence.
- Identify invariants, assumptions, flows, and reasoning hazards.

This skill defines a structured analysis format (see Example: Function Micro-Analysis below) and runs **before** the vulnerability-hunting phase.

---

2. When to Use This Skill

Use when:

- Deep comprehension is needed before bug or vulnerability discovery.
- You want bottom-up understanding instead of high-level guessing.
- Reducing hallucinations, contradictions, and context loss is critical.
- Preparing for security auditing, architecture review, or threat modeling.

Do **not** use for:

- Vulnerability findings
- Fix recommendations
- Exploit reasoning
- Severity/impact rating

---

3. How This Skill Behaves

When active, Claude will:

- Default to **ultra-granular analysis** of each block and line.
- Apply micro-level First Principles, 5 Whys, and 5 Hows.
- Build and refine a persistent global mental model.
- Update earlier assumptions when contradicted ("Earlier I thought X; now Y.").
- Periodically anchor summaries to maintain stable context.
- Avoid speculation; express uncertainty explicitly when needed.

Goal: **deep, accurate understanding**, not conclusions.

Rationalizations (Do Not Skip)

Rationalization	Why It's Wrong	Required Action
"I get the gist"	Gist-level understanding misses edge cases	Line-by-line analysis required
"This function is simple"	Simple functions compose into complex bugs	Apply 5 Whys anyway
"I'll remember this invariant"	You won't. Context degrades.	Write it down explicitly
"External call is probably fine"	External = adversarial until proven otherwise	Jump into code or model as hostile
"I can skip this helper"	Helpers contain assumptions that propagate	Trace the full call chain
"This is taking too long"	Rushed context = hallucinated vulnerabilities later	Slow is fast

---

4. Phase 1 — Initial Orientation (Bottom-Up Scan)

Before deep analysis, Claude performs a minimal mapping:

1. Identify major modules/files/contracts.
2. Note obvious public/external entrypoints.
3. Identify likely actors (users, owners, relayers, oracles, other contracts).
4. Identify important storage variables, dicts, state structs, or cells.
5. Build a preliminary structure without assuming behavior.

This establishes anchors for detailed analysis.

---

5. Phase 2 — Ultra-Granular Function Analysis (Default Mode)

Every non-trivial function receives full micro analysis.

5.1 Per-Function Microstructure Checklist

For each function:

1. **Purpose**
   - Why the function exists and its role in the system.
2. **Inputs & Assumptions**
   - Parameters and implicit inputs (state, sender, env).
   - Preconditions and constraints.
3. **Outputs & Effects**
   - Return values.
   - State/storage writes.
   - Events/messages.
   - External interactions.
4. **Block-by-Block / Line-by-Line Analysis** For each logical block:
   - What it does.
   - Why it appears here (ordering logic).
   - What assumptions it relies on.
   - What invariants it establishes or maintains.
   - What later logic depends on it.

   Apply per-block:
   - **First Principles**
   - **5 Whys**
   - **5 Hows**

5.2 Cross-Function & External Flow Analysis

(Full Integration of Jump-Into-External-Code Rule)

When encountering calls, **continue the same micro-first analysis across boundaries.**

Internal Calls

- Jump into the callee immediately.
- Perform block-by-block analysis of relevant code.
- Track flow of data, assumptions, and invariants: caller → callee → return → caller.
- Note if callee logic behaves differently in this specific call context.

External Calls — Two Cases

Case A — External Call to a Contract Whose Code Exists in the Codebase Treat as an internal call:

- Jump into the target contract/function.
- Continue block-by-block micro-analysis.
- Propagate invariants and assumptions seamlessly.
- Consider edge cases based on the *actual* code, not a black-box guess.

Case B — External Call Without Available Code (True External / Black Box) Analyze as adversarial:

- Describe payload/value/gas or parameters sent.
- Identify assumptions about the target.
- Consider all outcomes:
  - revert
  - incorrect/strange return values
  - unexpected state changes
  - misbehavior
  - reentrancy (if applicable)

Continuity Rule

Treat the entire call chain as **one continuous execution flow**. Never reset context. All invariants, assumptions, and data dependencies must propagate across calls.

5.3 Complete Analysis Example

See FUNCTION_MICRO_ANALYSIS_EXAMPLE.md for a complete walkthrough demonstrating:

- Full micro-analysis of a DEX swap function
- Application of First Principles, 5 Whys, and 5 Hows
- Block-by-block analysis with invariants and assumptions
- Cross-function dependency mapping
- Risk analysis for external interactions

This example demonstrates the level of depth and structure required for all analyzed functions.

5.4 Output Requirements

When performing ultra-granular analysis, Claude MUST structure output following the format defined in OUTPUT_REQUIREMENTS.md.

Key requirements:

- **Purpose** (2-3 sentences minimum)
- **Inputs & Assumptions** (all parameters, preconditions, trust assumptions)
- **Outputs & Effects** (returns, state writes, external calls, events, postconditions)
- **Block-by-Block Analysis** (What, Why here, Assumptions, First Principles/5 Whys/5 Hows)
- **Cross-Function Dependencies** (internal calls, external calls with risk analysis, shared state)

Quality thresholds:

- Minimum 3 invariants per function
- Minimum 5 assumptions documented
- Minimum 3 risk considerations for external interactions
- At least 1 First Principles application
- At least 3 combined 5 Whys/5 Hows applications

5.5 Completeness Checklist

Before concluding micro-analysis of a function, verify against the COMPLETENESS_CHECKLIST.md:

- **Structural Completeness**: All required sections present (Purpose, Inputs, Outputs, Block-by-Block, Dependencies)
- **Content Depth**: Minimum thresholds met (invariants, assumptions, risk analysis, First Principles)
- **Continuity & Integration**: Cross-references, propagated assumptions, invariant couplings
- **Anti-Hallucination**: Line number citations, no vague statements, evidence-based claims

Analysis is complete when all checklist items are satisfied and no unresolved "unclear" items remain.

---

6. Phase 3 — Global System Understanding

After sufficient micro-analysis:

1. **State & Invariant Reconstruction**
   - Map reads/writes of each state variable.
   - Derive multi-function and multi-module invariants.
2. **Workflow Reconstruction**
   - Identify end-to-end flows (deposit, withdraw, lifecycle, upgrades).
   - Track how state transforms across these flows.
   - Record assumptions that persist across steps.
3. **Trust Boundary Mapping**
   - Actor → entrypoint → behavior.
   - Identify untrusted input paths.
   - Privilege changes and implicit role expectations.
4. **Complexity & Fragility Clustering**

- Functions with many assumptions.
- High branching logic.
- Multi-step dependencies.
- Coupled state changes across modules.

These clusters help guide the vulnerability-hunting phase.

---

7. Stability & Consistency Rules

(Anti-Hallucination, Anti-Contradiction)

Claude must:

- **Never reshape evidence to fit earlier assumptions.** When contradicted:
  - Update the model.
  - State the correction explicitly.
- **Periodically anchor key facts** Summarize core:
  - invariants
  - state relationships
  - actor roles
  - workflows
- **Avoid vague guesses** Use:
  - "Unclear; need to inspect X." instead of:
  - "It probably…"
- **Cross-reference constantly** Connect new insights to previous state, flows, and invariants to maintain global coherence.

---

8. Subagent Usage

Claude may spawn subagents for:

- Dense or complex functions.
- Long data-flow or control-flow chains.
- Cryptographic / mathematical logic.
- Complex state machines.
- Multi-module workflow reconstruction.

Subagents must:

- Follow the same micro-first rules.
- Return summaries that Claude integrates into its global model.

9. Relationship to Other Phases

This skill runs **before**:

- Vulnerability discovery
- Classification / triage
- Report writing
- Impact modeling
- Exploit reasoning

It exists solely to build:

- Deep understanding
- Stable context
- System-level clarity

---

10. Non-Goals

While active, Claude should NOT:

- Identify vulnerabilities
- Propose fixes
- Generate proofs-of-concept
- Model exploits
- Assign severity or impact

This is **pure context building** only.

Revision #5
Created 2026-02-18 08:40:02 UTC by John
Updated 2026-06-21 20:01:10 UTC by John