/codeql

Source: `~/.claude/skills/tob-static-analysis/skills/codeql/SKILL.md`

---

name: codeql description: >- Runs CodeQL static analysis for security vulnerability detection using interprocedural data flow and taint tracking. Applicable when finding vulnerabilities, running a security scan, performing a security audit, running CodeQL, building a CodeQL database, selecting query rulesets, creating data extension models, or processing CodeQL SARIF output. NOT for writing custom QL queries or CI/CD pipeline setup. allowed-tools:

- Bash
- Read
- Write
- Glob
- Grep
- AskUserQuestion
- Task
- TaskCreate
- TaskList
- TaskUpdate

CodeQL Analysis

Supported languages: Python, JavaScript/TypeScript, Go, Java/Kotlin, C/C++, C#, Ruby, Swift.

Skill resources: Reference files and templates are located at `{baseDir}/references/` and `{baseDir}/workflows/`. Use `{baseDir}` to resolve paths to these files at runtime.

Quick Start

For the common case ("scan this codebase for vulnerabilities"):

```
# 1. Verify CodeQL is installed
command -v codeql >/dev/null 2>&1 && codeql --version || echo "NOT INSTALLED"

# 2. Check for existing database
ls -dt codeql_*.db 2>/dev/null | head -1
```

Then execute the full pipeline: **build database → create data extensions → run analysis** using the workflows below.

When to Use

- Scanning a codebase for security vulnerabilities with deep data flow analysis
- Building a CodeQL database from source code (with build capability for compiled languages)
- Finding complex vulnerabilities that require interprocedural taint tracking or AST/CFG analysis
- Performing comprehensive security audits with multiple query packs

When NOT to Use

- **Writing custom queries** - Use a dedicated query development skill
- **CI/CD integration** - Use GitHub Actions documentation directly
- **Quick pattern searches** - Use Semgrep or grep for speed
- **No build capability** for compiled languages - Consider Semgrep instead
- **Single-file or lightweight analysis** - Semgrep is faster for simple pattern matching

Rationalizations to Reject

These shortcuts lead to missed findings. Do not accept them:

- **"security-extended is enough"** - It is the baseline. Always check if Trail of Bits packs and Community Packs are available for the language. They catch categories `security-extended` misses entirely.
- **"The database built, so it's good"** - A database that builds does not mean it extracted well. Always run Step 4 (quality assessment) and check file counts against expected source files. A cached build produces zero useful extraction.
- **"Data extensions aren't needed for standard frameworks"** - Even Django/Spring apps have custom wrappers around ORM calls, request parsing, or shell execution that

CodeQL does not model. Skipping the extensions workflow means missing vulnerabilities in project-specific code.

- **"build-mode=none is fine for compiled languages"** - It produces severely incomplete analysis. No interprocedural data flow through compiled code is traced. Only use as an absolute last resort and clearly flag the limitation.
- **"No findings means the code is secure"** - Zero findings can indicate poor database quality, missing models, or wrong query packs. Investigate before reporting clean results.
- **"I'll just run the default suite"** - The default suite varies by how CodeQL is invoked. Always explicitly specify the suite (e.g., `security-extended`) so results are reproducible.

Workflow Selection

This skill has three workflows:

Workflow	Purpose
build-database	Create CodeQL database using 3 build methods in sequence
create-data-extensions	Detect or generate data extension models for project APIs
run-analysis	Select rulesets, execute queries, process results

Auto-Detection Logic

If user explicitly specifies what to do (e.g., "build a database", "run analysis"), execute that workflow.

Default pipeline for "test", "scan", "analyze", or similar: Execute all three workflows sequentially: build → extensions → analysis. The create-data-extensions step is critical for finding vulnerabilities in projects with custom frameworks or annotations that CodeQL doesn't model by default.

```
# Check if database exists
DB=$(ls -dt codeql_*.db 2>/dev/null | head -1)
if [ -n "$DB" ] && codeql resolve database -- "$DB" >/dev/null 2>&1; then
  echo "DATABASE EXISTS ($DB) - can run analysis"
else
  echo "NO DATABASE - need to build first"
fi
```

Condition	Action

No database exists	Execute build → extensions → analysis (full pipeline)
Database exists, no extensions	Execute extensions → analysis
Database exists, extensions exist	Ask user: run analysis on existing DB, or rebuild?
User says "just run analysis" or "skip extensions"	Run analysis only

Decision Prompt

If unclear, ask user:

```
I can help with CodeQL analysis. What would you like to do?

1. **Full scan (Recommended)** - Build database, create extensions, then run analysis
2. **Build database** - Create a new CodeQL database from this codebase
3. **Create data extensions** - Generate custom source/sink models for project APIs
4. **Run analysis** - Run security queries on existing database

[If database exists: "I found an existing database at <DB_NAME>"]
```

Revision #5
Created 2026-02-18 08:40:09 UTC by John
Updated 2026-06-21 20:01:16 UTC by John