/semgrep-rule-creator

Source: `~/.claude/skills/tob-semgrep-rule-creator/skills/semgrep-rule-creator/SKILL.md`

---

name: semgrep-rule-creator description: Creates custom Semgrep rules for detecting security vulnerabilities, bug patterns, and code patterns. Use when writing Semgrep rules or building custom static analysis detections. allowed-tools:

- Bash
- Read
- Write
- Edit
- Glob
- Grep
- WebFetch

Semgrep Rule Creator

Create production-quality Semgrep rules with proper testing and validation.

When to Use

Ideal scenarios:

- Writing Semgrep rules for specific bug patterns
- Writing rules to detect security vulnerabilities in your codebase
- Writing taint mode rules for data flow vulnerabilities

- Writing rules to enforce coding standards

When NOT to Use

Do NOT use this skill for:

- Running existing Semgrep rulesets
- General static analysis without custom rules (use `static-analysis` skill)

Rationalizations to Reject

When writing Semgrep rules, reject these common shortcuts:

- **"The pattern looks complete"** → Still run `semgrep --test --config <rule-id>.yaml <rule-id>.<ext>` to verify. Untested rules have hidden false positives/negatives.
- **"It matches the vulnerable case"** → Matching vulnerabilities is half the job. Verify safe cases don't match (false positives break trust).
- **"Taint mode is overkill for this"** → If data flows from user input to a dangerous sink, taint mode gives better precision than pattern matching.
- **"One test is enough"** → Include edge cases: different coding styles, sanitized inputs, safe alternatives, and boundary conditions.
- **"I'll optimize the patterns first"** → Write correct patterns first, optimize after all tests pass. Premature optimization causes regressions.
- **"The AST dump is too complex"** → The AST reveals exactly how Semgrep sees code. Skipping it leads to patterns that miss syntactic variations.

Anti-Patterns

Too broad - matches everything, useless for detection:

```
# BAD: Matches any function call
pattern: $FUNC(...)

# GOOD: Specific dangerous function
pattern: eval(...)
```

Missing safe cases in tests - leads to undetected false positives:

```
# BAD: Only tests vulnerable case
# ruleid: my-rule
```

```
dangerous(user_input)

# GOOD: Include safe cases to verify no false positives
# ruleid: my-rule
dangerous(user_input)

# ok: my-rule
dangerous(sanitize(user_input))

# ok: my-rule
dangerous("hardcoded_safe_value")
```

Overly specific patterns - misses variations:

```
# BAD: Only matches exact format
pattern: os.system("rm " + $VAR)

# GOOD: Matches all os.system calls with taint tracking
mode: taint
pattern-sinks:
  - pattern: os.system(...)
```

Strictness Level

This workflow is **strict** - do not skip steps:

- **Read documentation first**: See Documentation before writing Semgrep rules
- **Test-first is mandatory**: Never write a rule without tests
- **100% test pass is required**: "Most tests pass" is not acceptable
- **Optimization comes last**: Only simplify patterns after all tests pass
- **Avoid generic patterns**: Rules must be specific, not match broad patterns
- **Prioritize taint mode**: For data flow vulnerabilities
- **One YAML file - one Semgrep rule**: Each YAML file must contain only one Semgrep rule; don't combine multiple rules in a single file
- **No generic rules**: When targeting a specific language for Semgrep rules - avoid generic pattern matching (`languages: generic`)
- **Forbidden `todook` and `todoruleid` test annotations**: `todoruleid: <rule-id>` and `todook: <rule-id>` annotations in tests files for future rule improvements are forbidden

Overview

This skill guides creation of Semgrep rules that detect security vulnerabilities and code patterns. Rules are created iteratively: analyze the problem, write tests first, analyze AST structure, write the rule, iterate until all tests pass, optimize the rule.

Approach selection:

- **Taint mode** (prioritize): Data flow issues where untrusted input reaches dangerous sinks
- **Pattern matching**: Simple syntactic patterns without data flow requirements

Why prioritize taint mode? Pattern matching finds syntax but misses context. A pattern `eval($X)` matches both `eval(user_input)` (vulnerable) and `eval("safe_literal")` (safe). Taint mode tracks data flow, so it only alerts when untrusted data actually reaches the sink—dramatically reducing false positives for injection vulnerabilities.

Iterating between approaches: It's okay to experiment. If you start with taint mode and it's not working well (e.g., taint doesn't propagate as expected, too many false positives/negatives), switch to pattern matching. Conversely, if pattern matching produces too many false positives on safe cases, try taint mode instead. The goal is a working rule—not rigid adherence to one approach.

Output structure - exactly 2 files in a directory named after the rule-id:

```
<rule-id>/
├── <rule-id>.yaml     # Semgrep rule
└── <rule-id>.<ext>    # Test file with ruleid/ok annotations
```

Quick Start

```
rules:
  - id: insecure-eval
    languages: [python]
    severity: HIGH
    message: User input passed to eval() allows code execution
    mode: taint
    pattern-sources:
      - pattern: request.args.get(...)
    pattern-sinks:
      - pattern: eval(...)
```

Test file (`insecure-eval.py`):

```
# ruleid: insecure-eval
eval(request.args.get('code'))

# ok: insecure-eval
eval("print('safe')")
```

Run tests (from rule directory): `semgrep --test --config <rule-id>.yaml <rule-id>.<ext>`

Quick Reference

- For commands, pattern operators, and taint mode syntax, see quick-reference.md.
- For detailed workflow and examples, you MUST see workflow.md

Workflow

Copy this checklist and track progress:

```
Semgrep Rule Progress:
- [ ] Step 1: Analyze the Problem
- [ ] Step 2: Write Tests First
- [ ] Step 3: Analyze AST structure
- [ ] Step 4: Write the rule
- [ ] Step 5: Iterate until all tests pass (semgrep --test)
- [ ] Step 6: Optimize the rule (remove redundancies, re-test)
- [ ] Step 7: Final Run
```

Documentation

REQUIRED: Before writing any rule, use WebFetch to read **all** of these 4 links with Semgrep documentation:

1. Rule Syntax
2. Pattern Syntax
3. ToB Testing Handbook - Semgrep
4. Constant propagation
5. Writing Rules Index

Revision #4
Created 2026-02-18 08:40:08 UTC by John
Updated 2026-05-31 20:01:51 UTC by John