/semgrep

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

---

name: semgrep description: Run Semgrep static analysis scan on a codebase using parallel subagents. Automatically detects and uses Semgrep Pro for cross-file analysis when available. Use when asked to scan code for vulnerabilities, run a security audit with Semgrep, find bugs, or perform static analysis. Spawns parallel workers for multi-language codebases and triage. allowed-tools:

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

Semgrep Security Scan

Run a complete Semgrep scan with automatic language detection, parallel execution via Task subagents, and parallel triage. Automatically uses Semgrep Pro for cross-file taint analysis when available.

Prerequisites

Required: Semgrep CLI

```
semgrep --version
```

If not installed, see Semgrep installation docs.

Optional: Semgrep Pro (for cross-file analysis and Pro languages)

```
# Check if Semgrep Pro engine is installed
semgrep --pro --validate --config p/default 2>/dev/null && echo "Pro available" || echo "OSS only"

# If logged in, install/update Pro Engine
semgrep install-semgrep-pro
```

Pro enables: cross-file taint tracking, inter-procedural analysis, and additional languages (Apex, C#, Elixir).

When to Use

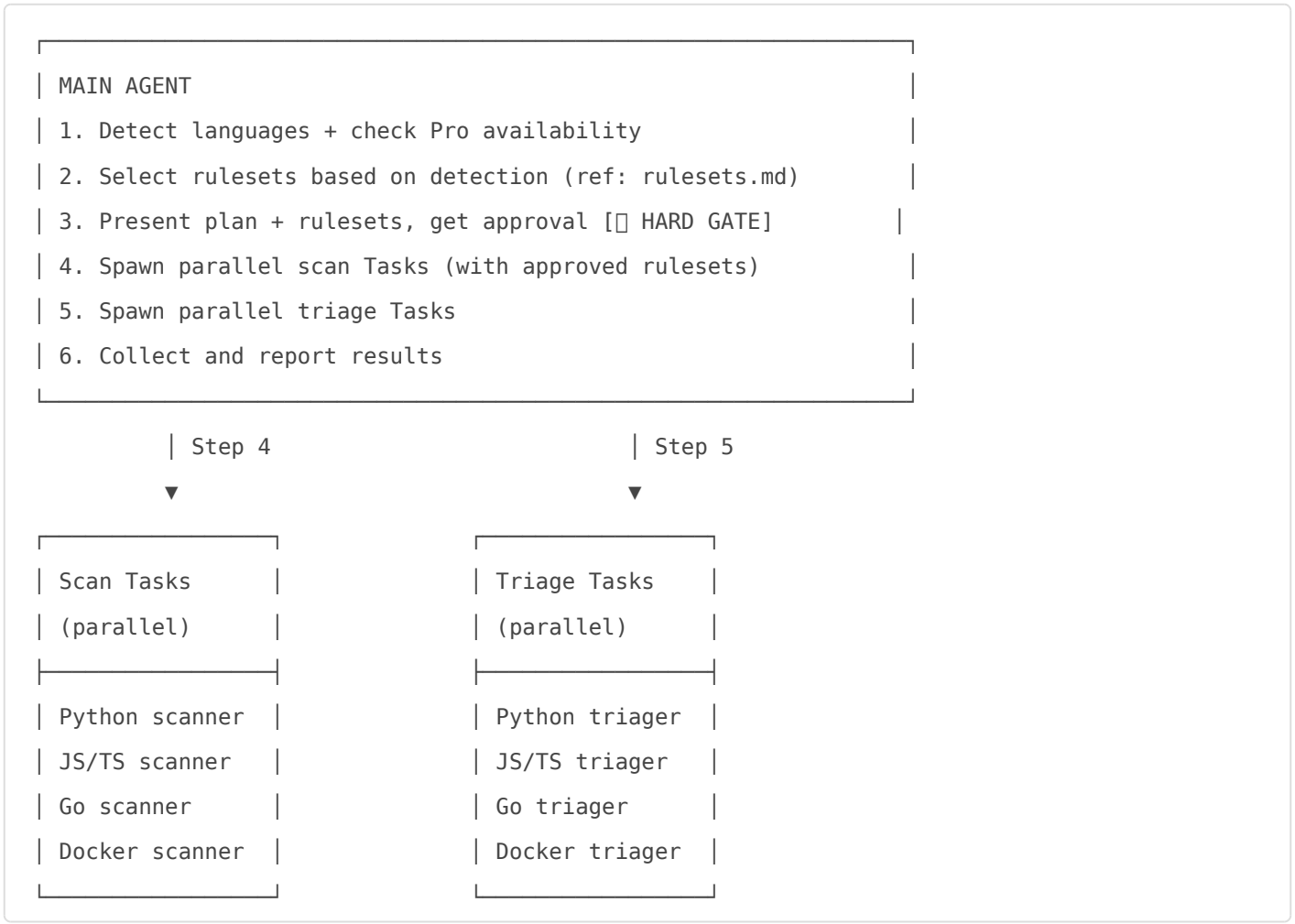
- Security audit of a codebase
- Finding vulnerabilities before code review
- Scanning for known bug patterns
- First-pass static analysis

When NOT to Use

- Binary analysis → Use binary analysis tools
- Already have Semgrep CI configured → Use existing pipeline
- Need cross-file analysis but no Pro license → Consider CodeQL as alternative
- Creating custom Semgrep rules → Use `semgrep-rule-creator` skill
- Porting existing rules to other languages → Use `semgrep-rule-variant-creator` skill

Orchestration Architecture

This skill uses **parallel Task subagents** for maximum efficiency:



Workflow Enforcement via Task System

This skill uses the **Task system** to enforce workflow compliance. On invocation, create these tasks:

```
TaskCreate: "Detect languages and Pro availability" (Step 1)
TaskCreate: "Select rulesets based on detection" (Step 2) - blockedBy: Step 1
TaskCreate: "Present plan with rulesets, get approval" (Step 3) - blockedBy: Step 2
TaskCreate: "Execute scans with approved rulesets" (Step 4) - blockedBy: Step 3
TaskCreate: "Triage findings" (Step 5) - blockedBy: Step 4
TaskCreate: "Report results" (Step 6) - blockedBy: Step 5
```

Mandatory Gates

Task	Gate Type	Cannot Proceed Until
Step 3: Get approval	**HARD GATE**	User explicitly approves rulesets + plan

Task	Gate Type	Cannot Proceed Until
Step 5: Triage	**SOFT GATE**	All scan JSON files exist

Step 3 is a HARD GATE: Mark as `completed` ONLY after user says "yes", "proceed", "approved", or equivalent.

Task Flow Example

```
1. Create all 6 tasks with dependencies
2. TaskUpdate Step 1 → in_progress, execute detection
3. TaskUpdate Step 1 → completed
4. TaskUpdate Step 2 → in_progress, select rulesets
5. TaskUpdate Step 2 → completed
6. TaskUpdate Step 3 → in_progress, present plan with rulesets
7. STOP: Wait for user response (may modify rulesets)
8. User approves → TaskUpdate Step 3 → completed
9. TaskUpdate Step 4 → in_progress (now unblocked)
... continue workflow
```

Workflow

Step 1: Detect Languages and Pro Availability (Main Agent)

```
# Check if Semgrep Pro is available (non-destructive check)
SEMGREP_PRO=false
if semgrep --pro --validate --config p/default 2>/dev/null; then
  SEMGREP_PRO=true
  echo "Semgrep Pro: AVAILABLE (cross-file analysis enabled)"
else
  echo "Semgrep Pro: NOT AVAILABLE (OSS mode, single-file analysis)"
fi

# Find languages by file extension
fd -t f -e py -e js -e ts -e jsx -e tsx -e go -e rb -e java -e php -e c -e cpp -e rs | \
  sed 's/.*\.//' | sort | uniq -c | sort -rn
```

```
# Check for frameworks/technologies
ls -la package.json pyproject.toml Gemfile go.mod Cargo.toml pom.xml 2>/dev/null
fd -t f "Dockerfile" "docker-compose" ".tf" "*.yaml" "*.yml" | head -20
```

Map findings to categories:

Detection	Category
`.py`, `pyproject.toml`	Python
`.js`, `.ts`, `package.json`	JavaScript/TypeScript
`.go`, `go.mod`	Go
`.rb`, `Gemfile`	Ruby
`.java`, `pom.xml`	Java
`.php`	PHP
`.c`, `.cpp`	C/C++
`.rs`, `Cargo.toml`	Rust
`Dockerfile`	Docker
`.tf`	Terraform
k8s manifests	Kubernetes

Step 2: Select Rulesets Based on Detection

Using the detected languages and frameworks from Step 1, select rulesets by following the **Ruleset Selection Algorithm** in rulesets.md.

The algorithm covers:

1. Security baseline (always included)
2. Language-specific rulesets
3. Framework rulesets (if detected)
4. Infrastructure rulesets
5. **Required** third-party rulesets (Trail of Bits, 0xdea, Decurity - NOT optional)
6. Registry verification

Output: Structured JSON passed to Step 3 for user review:

```
{
  "baseline": ["p/security-audit", "p/secrets"],
  "python": ["p/python", "p/django"],
```

```
  "javascript": ["p/javascript", "p/react", "p/nodejs"],
  "docker": ["p/dockerfile"],
  "third_party": ["https://github.com/trailofbits/semgrep-rules"]
}
```

Step 3: CRITICAL GATE - Present Plan and Get Approval

> **MANDATORY CHECKPOINT - DO NOT SKIP**
>
> This step requires explicit user approval before proceeding. User may modify rulesets before approving.

Present plan to user with **explicit ruleset listing**:

```
## Semgrep Scan Plan

**Target:** /path/to/codebase
**Output directory:** ./semgrep-results-001/
**Engine:** Semgrep Pro (cross-file analysis) | Semgrep OSS (single-file)

### Detected Languages/Technologies:
- Python (1,234 files) - Django framework detected
- JavaScript (567 files) - React detected
- Dockerfile (3 files)

### Rulesets to Run:

**Security Baseline (always included):**
- [x] `p/security-audit` - Comprehensive security rules
- [x] `p/secrets` - Hardcoded credentials, API keys

**Python (1,234 files):**
- [x] `p/python` - Python security patterns
- [x] `p/django` - Django-specific vulnerabilities

**JavaScript (567 files):**
- [x] `p/javascript` - JavaScript security patterns
```

```
- [x] `p/react` - React-specific issues
- [x] `p/nodejs` - Node.js server-side patterns

**Docker (3 files):**
- [x] `p/dockerfile` - Dockerfile best practices

**Third-party (auto-included for detected languages):**
- [x] Trail of Bits rules - https://github.com/trailofbits/semgrep-rules

**Available but not selected:**
- [ ] `p/owasp-top-ten` - OWASP Top 10 (overlaps with security-audit)

### Execution Strategy:
- Spawn 3 parallel scan Tasks (Python, JavaScript, Docker)
- Total rulesets: 9
- [If Pro] Cross-file taint tracking enabled

**Want to modify rulesets?** Tell me which to add or remove.
**Ready to scan?** Say "proceed" or "yes".
```

STOP: Await explicit user approval

After presenting the plan:

1. **If user wants to modify rulesets:**
   - Add requested rulesets to the appropriate category
   - Remove requested rulesets
   - Re-present the updated plan
   - Return to waiting for approval
2. **Use AskUserQuestion** if user hasn't responded:

   ```
   "I've prepared the scan plan with 9 rulesets (including Trail of Bits). Proceed with scanning?"
   Options: ["Yes, run scan", "Modify rulesets first"]
   ```

3. **Valid approval responses:**
   - "yes", "proceed", "approved", "go ahead", "looks good", "run it"
4. **Mark task completed** only after approval with final rulesets confirmed
5. **Do NOT treat as approval:**
   - User's original request ("scan this codebase")
   - Silence / no response
   - Questions about the plan

Pre-Scan Checklist

Before marking Step 3 complete, verify:

- ☐ Target directory shown to user
- ☐ Engine type (Pro/OSS) displayed
- ☐ Languages detected and listed
- ☐ **All rulesets explicitly listed with checkboxes**
- ☐ User given opportunity to modify rulesets
- ☐ User explicitly approved (quote their confirmation)
- ☐ **Final ruleset list captured for Step 4**

Step 4: Spawn Parallel Scan Tasks

Create output directory with run number to avoid collisions, then spawn Tasks with **approved rulesets from Step 3**:

```
# Find next available run number
LAST=$(ls -d semgrep-results-[0-9][0-9][0-9] 2>/dev/null | sort | tail -1 | grep -o '[0-9]*$'
|| true)
NEXT_NUM=$(printf "%03d" $(( ${LAST:-0} + 1 )))
OUTPUT_DIR="semgrep-results-${NEXT_NUM}"
mkdir -p "$OUTPUT_DIR"
echo "Output directory: $OUTPUT_DIR"
```

Spawn N Tasks in a SINGLE message (one per language category) using `subagent_type: Bash`.

Use the scanner task prompt template from scanner-task-prompt.md.

Example - 3 Language Scan (with approved rulesets):

Spawn these 3 Tasks in a SINGLE message:

1. **Task: Python Scanner**
   - Approved rulesets: p/python, p/django, p/security-audit, p/secrets, https://github.com/trailofbits/semgrep-rules
   - Output: semgrep-results-001/python-*.json
2. **Task: JavaScript Scanner**
   - Approved rulesets: p/javascript, p/react, p/nodejs, p/security-audit, p/secrets, https://github.com/trailofbits/semgrep-rules
   - Output: semgrep-results-001/js-*.json

3. **Task: Docker Scanner**
   - Approved rulesets: p/dockerfile
   - Output: semgrep-results-001/docker-*.json

Step 5: Spawn Parallel Triage Tasks

After scan Tasks complete, spawn triage Tasks using `subagent_type: general-purpose` (triage requires reading code context, not just running commands).

Use the triage task prompt template from triage-task-prompt.md.

Step 6: Collect Results (Main Agent)

After all Tasks complete, generate merged SARIF and report:

Generate merged SARIF with only triaged true positives:

```
uv run {baseDir}/scripts/merge_triaged_sarif.py [OUTPUT_DIR]
```

This script:

1. Attempts to use SARIF Multitool for merging (if `npx` is available)
2. Falls back to pure Python merge if Multitool unavailable
3. Reads all `*-triage.json` files to extract true positive findings
4. Filters merged SARIF to include only triaged true positives
5. Writes output to `[OUTPUT_DIR]/findings-triaged.sarif`

Optional: Install SARIF Multitool for better merge quality:

```
npm install -g @microsoft/sarif-multitool
```

Report to user:

```
## Semgrep Scan Complete

**Scanned:** 1,804 files
**Rulesets used:** 9 (including Trail of Bits)
**Total raw findings:** 156
**After triage:** 32 true positives

### By Severity:
- ERROR: 5
```

```
- WARNING: 18
- INFO: 9

### By Category:
- SQL Injection: 3
- XSS: 7
- Hardcoded secrets: 2
- Insecure configuration: 12
- Code quality: 8

Results written to:
- semgrep-results-001/findings-triaged.sarif (SARIF, true positives only)
- semgrep-results-001/*-triage.json (triage details per language)
- semgrep-results-001/*.json (raw scan results)
- semgrep-results-001/*.sarif (raw SARIF per ruleset)
```

Common Mistakes

Mistake	Correct Approach
Running without `--metrics=off`	Always use `--metrics=off` to prevent telemetry
Running rulesets sequentially	Run in parallel with `&` and `wait`
Not scoping rulesets to languages	Use `--include="*.py"` for language-specific rules
Reporting raw findings without triage	Always triage to remove false positives
Single-threaded for multi-lang	Spawn parallel Tasks per language
Sequential Tasks	Spawn all Tasks in SINGLE message for parallelism
Using OSS when Pro is available	Check login status; use `--pro` for deeper analysis
Assuming Pro is unavailable	Always check with login detection before scanning

Limitations

1. **OSS mode:** Cannot track data flow across files (login with `semgrep login` and run `semgrep install-semgrep-pro` to enable)
2. **Pro mode:** Cross-file analysis uses `-j 1` (single job) which is slower per ruleset, but parallel rulesets compensate
3. Triage requires reading code context - parallelized via Tasks

4. Some false positive patterns require human judgment

Rationalizations to Reject

Shortcut	Why It's Wrong
"User asked for scan, that's approval"	Original request ≠ plan approval; user must confirm specific parameters. Present plan, use AskUserQuestion, await explicit "yes"
"Step 3 task is blocking, just mark complete"	Lying about task status defeats enforcement. Only mark complete after real approval
"I already know what they want"	Assumptions cause scanning wrong directories/rulesets. Present plan with all parameters for verification
"Just use default rulesets"	User must see and approve exact rulesets before scan
"Add extra rulesets without asking"	Modifying approved list without consent breaks trust
"Skip showing ruleset list"	User can't make informed decision without seeing what will run
"Third-party rulesets are optional"	Trail of Bits, 0xdea, Decurity rules catch vulnerabilities not in official registry - they are REQUIRED when language matches
"Skip triage, report everything"	Floods user with noise; true issues get lost
"Run one ruleset at a time"	Wastes time; parallel execution is faster
"Use --config auto"	Sends metrics; less control over rulesets
"Triage later"	Findings without context are harder to evaluate
"One Task at a time"	Defeats parallelism; spawn all Tasks together
"Pro is too slow, skip --pro"	Cross-file analysis catches 250% more true positives; worth the time
"Don't bother checking for Pro"	Missing Pro = missing critical cross-file vulnerabilities
"OSS is good enough"	OSS misses inter-file taint flows; always prefer Pro when available

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