/testing-handbook-generator

Source: `~/.claude/skills/tob-testing-handbook-skills/skills/testing-handbook-generator/SKILL.md`

---

name: testing-handbook-generator description: > Meta-skill that analyzes the Trail of Bits Testing Handbook (appsec.guide) and generates Claude Code skills for security testing tools and techniques. Use when creating new skills based on handbook content.

Testing Handbook Skill Generator

Generate and maintain Claude Code skills from the Trail of Bits Testing Handbook.

When to Use

Invoke this skill when:

- Creating new security testing skills from handbook content
- User mentions "testing handbook", "appsec.guide", or asks about generating skills
- Bulk skill generation or refresh is needed

Do NOT use for:

- General security testing questions (use the generated skills)
- Non-handbook skill creation

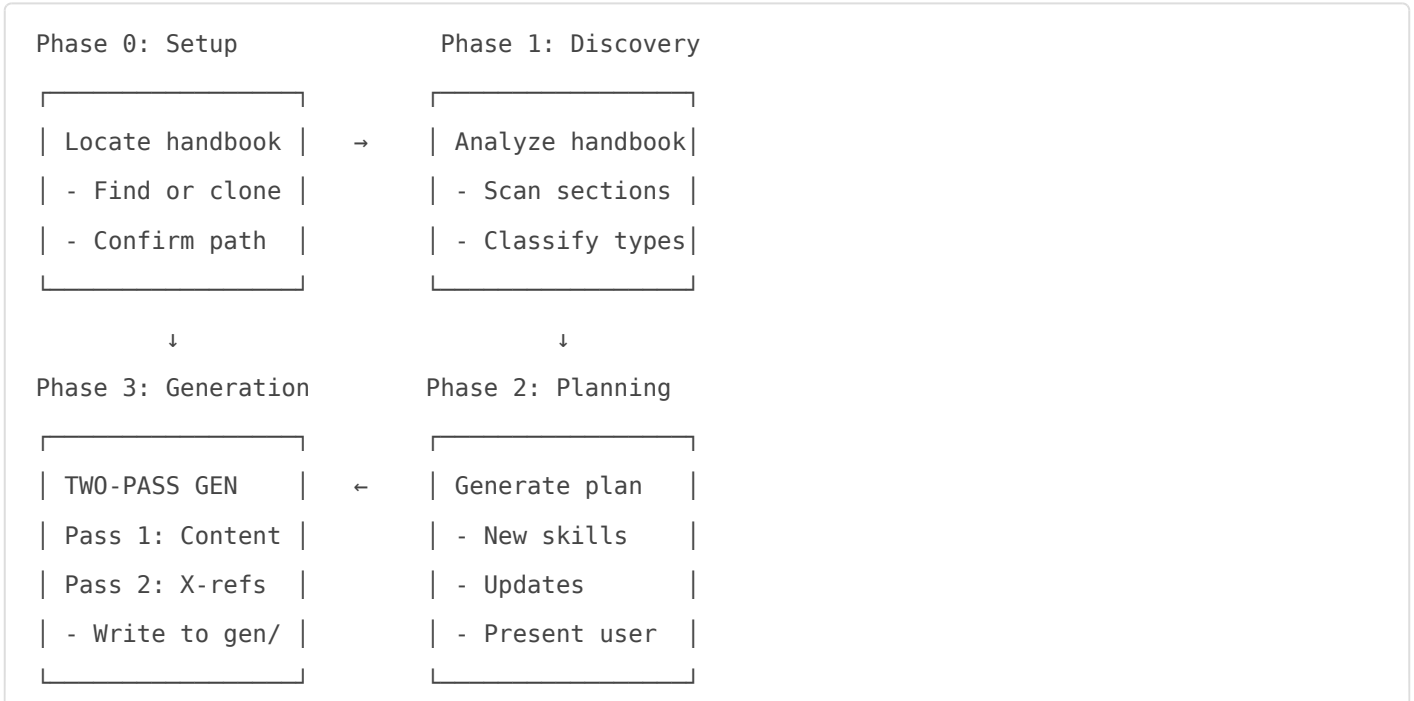
Handbook Location

The skill needs the Testing Handbook repository. See discovery.md for full details.

Quick reference: Check `./testing-handbook`, `../testing-handbook`, `~/testing-handbook` → ask user → clone as last resort.

Repository: `https://github.com/trailofbits/testing-handbook`

Workflow Overview



```
         ↓
Phase 4: Testing           Phase 5: Finalize
┌──────────────────┐       ┌──────────────────┐
│ Validate skills  │   →   │ Post-generation  │
│ - Run validator  │       │ - Update README  │
│ - Test activation│       │ - Update X-refs  │
│ - Fix issues     │       │ - Self-improve   │
└──────────────────┘       └──────────────────┘
```

Scope Restrictions

ONLY modify these locations:

- `plugins/testing-handbook-skills/skills/[skill-name]/*` - Generated skills (as siblings to testing-handbook-generator)
- `plugins/testing-handbook-skills/skills/testing-handbook-generator/*` - Self-improvement
- Repository root `README.md` - Add generated skills to table

NEVER modify or analyze:

- Other plugins (`plugins/property-based-testing/`, `plugins/static-analysis/`, etc.)
- Other skills outside this plugin

Do not scan or pull into context any skills outside of `testing-handbook-skills/`. Generate skills based solely on handbook content and resources referenced from it.

Quick Reference

Section ? Skill Type Mapping

Handbook Section	Skill Type	Template
`/static-analysis/[tool]/`	Tool Skill	tool-skill.md
`/fuzzing/[lang]/[fuzzer]/`	Fuzzer Skill	fuzzer-skill.md
`/fuzzing/techniques/`	Technique Skill	technique-skill.md
`/crypto/[tool]/`	Domain Skill	domain-skill.md
`/web/[tool]/`	Tool Skill	tool-skill.md

Skill Candidate Signals

Signal	Indicates
`_index.md` with `bookCollapseSection: true`	Major tool/topic
Numbered files (00-, 10-, 20-)	Structured content
`techniques/` subsection	Methodology content
`99-resources.md` or `91-resources.md`	Has external links

Exclusion Signals

Signal	Action
`draft: true` in frontmatter	Skip section
Empty directory	Skip section
Template/placeholder file	Skip section
GUI-only tool (e.g., `web/burp/`)	Skip section (Claude cannot operate GUI tools)

Decision Tree

Starting skill generation?

```
├─ Need to analyze handbook and build plan?
│  └─ Read: discovery.md
│     (Handbook analysis methodology, plan format)
│
├─ Spawning skill generation agents?
│  └─ Read: agent-prompt.md
│     (Full prompt template, variable reference, validation checklist)
│
├─ Generating a specific skill type?
│  └─ Read appropriate template:
│     ├─ Tool (Semgrep, CodeQL) → templates/tool-skill.md
│     ├─ Fuzzer (libFuzzer, AFL++) → templates/fuzzer-skill.md
│     ├─ Technique (harness, coverage) → templates/technique-skill.md
│     └─ Domain (crypto, web) → templates/domain-skill.md
│
```

```
├─ Validating generated skills?
│  └─ Run: scripts/validate-skills.py
│     Then read: testing.md for activation testing
│
├─ Finalizing after generation?
│  └─ See: Post-Generation Tasks below
│     (Update main README, update Skills Cross-Reference, self-improvement)
│
└─ Quick generation from specific section?
   └─ Use Quick Reference above, apply template directly
```

Two-Pass Generation (Phase 3)

Generation uses a **two-pass approach** to solve forward reference problems (skills referencing other skills that don't exist yet).

Pass 1: Content Generation (Parallel)

Generate all skills in parallel **without** the Related Skills section:

```
Pass 1 - Generating 5 skills in parallel:
├─ Agent 1: libfuzzer (fuzzer) → skills/libfuzzer/SKILL.md
├─ Agent 2: aflpp (fuzzer) → skills/aflpp/SKILL.md
├─ Agent 3: semgrep (tool) → skills/semgrep/SKILL.md
├─ Agent 4: harness-writing (technique) → skills/harness-writing/SKILL.md
└─ Agent 5: wycheproof (domain) → skills/wycheproof/SKILL.md

Each agent uses: pass=1 (content only, Related Skills left empty)
```

Pass 1 agents:

- Generate all sections EXCEPT Related Skills
- Leave a placeholder: `## Related Skills\n\n<!-- PASS2: populate after all skills exist -->`
- Output report includes `references: DEFERRED`

Pass 2: Cross-Reference Population (Sequential)

After all Pass 1 agents complete, run Pass 2 to populate Related Skills:

```
Pass 2 - Populating cross-references:
├─ Read all generated skill names from skills/*/SKILL.md
├─ For each skill, determine related skills based on:
│   ├─ related_sections from discovery (handbook structure)
│   ├─ Skill type relationships (fuzzers → techniques)
│   └─ Explicit mentions in content
└─ Update each SKILL.md's Related Skills section
```

Pass 2 process:

1. Collect all generated skill names: `ls -d skills/*/SKILL.md`
2. For each skill, identify related skills using the mapping from discovery
3. Edit each SKILL.md to replace the placeholder with actual links
4. Validate cross-references exist (no broken links)

Agent Prompt Template

See **agent-prompt.md** for the full prompt template with:

- Variable substitution reference (including `pass` variable)
- Pre-write validation checklist
- Hugo shortcode conversion rules
- Line count splitting rules
- Error handling guidance
- Output report format

Collecting Results

After Pass 1: Aggregate output reports, verify all skills generated. After Pass 2: Run validator to check cross-references.

Handling Agent Failures

If an agent fails or produces invalid output:

Failure Type	Detection	Recovery Action
Agent crashed	No output report	Re-run single agent with same inputs
Validation failed	Output report shows errors	Check gaps/warnings, manually patch or re-run

Failure Type	Detection	Recovery Action
Wrong skill type	Content doesn't match template	Re-run with corrected `type` parameter
Missing content	Output report lists gaps	Accept if minor, or provide additional `related_sections`
Pass 2 broken ref	Validator shows missing skill	Check if skill was skipped, update reference

Important: Do NOT re-run the entire parallel batch for a single agent failure. Fix individual failures independently.

Single-Skill Regeneration

To regenerate a single skill without re-running the entire batch:

```
# Regenerate single skill (Pass 1 - content only)
"Use testing-handbook-generator to regenerate the {skill-name} skill from section
{section_path}"

# Example:
"Use testing-handbook-generator to regenerate the libfuzzer skill from section fuzzing/c-
cpp/10-libfuzzer"
```

Regeneration workflow:

1. Re-read the handbook section for fresh content
2. Apply the appropriate template
3. Write to `skills/{skill-name}/SKILL.md` (overwrites existing)
4. Re-run Pass 2 for that skill only to update cross-references
5. Run validator on the single skill: `uv run scripts/validate-skills.py --skill {skill-name}`

Output Location

Generated skills are written to:

```
skills/[skill-name]/SKILL.md
```

Each skill gets its own directory for potential supporting files (as siblings to testing-handbook-generator).

Quality Checklist

Before delivering generated skills:

- [ ] All handbook sections analyzed (Phase 1)
- [ ] Plan presented to user before generation (Phase 2)
- [ ] Parallel agents launched - one per skill (Phase 3)
- [ ] Templates applied correctly per skill type
- [ ] Validator passes: `uv run scripts/validate-skills.py`
- [ ] Activation testing passed - see testing.md
- [ ] Main `README.md` updated with generated skills table
- [ ] `README.md` Skills Cross-Reference graph updated
- [ ] Self-improvement notes captured
- [ ] User notified with summary

Post-Generation Tasks

1. Update Main README

After generating skills, update the repository's main `README.md` to list them.

Format: Add generated skills to the same "Available Plugins" table, directly after `testing-handbook-skills`. Use plain text `testing-handbook-generator` as the author (no link).

Example:

```
| Plugin | Description | Author |
|--------|-------------|--------|
| ... other plugins ... |
| [testing-handbook-skills](plugins/testing-handbook-skills/) | Meta-skill that generates skills from the Testing Handbook | Paweł Płatek |
| [libfuzzer](plugins/testing-handbook-skills/skills/libfuzzer/) | Coverage-guided fuzzing with libFuzzer for C/C++ | testing-handbook-generator |
| [aflpp](plugins/testing-handbook-skills/skills/aflpp/) | Multi-core fuzzing with AFL++ | testing-handbook-generator |
| [semgrep](plugins/testing-handbook-skills/skills/semgrep/) | Fast static analysis for finding bugs | testing-handbook-generator |
```

2. Update Skills Cross-Reference

After generating skills, update the `README.md`'s **Skills Cross-Reference** section with the mermaid graph showing skill relationships.

Process:

1. Read each generated skill's `SKILL.md` and extract its `## Related Skills` section
2. Build the mermaid graph with nodes grouped by skill type (Fuzzers, Techniques, Tools, Domain)
3. Add edges based on the Related Skills relationships:
   - Solid arrows (`-->`) for primary technique dependencies
   - Dashed arrows (`-.->`) for alternative tool suggestions
4. Replace the existing mermaid code block in README.md

Edge classification:

Relationship	Arrow Style	Example
Fuzzer → Technique	`-->`	`libfuzzer --> harness-writing`
Tool → Tool (alternative)	`-.->`	`semgrep -.-> codeql`
Fuzzer → Fuzzer (alternative)	`-.->`	`libfuzzer -.-> aflpp`
Technique → Technique	`-->`	`harness-writing --> coverage-analysis`

Validation: After updating, run `validate-skills.py` to verify all referenced skills exist.

3. Self-Improvement

After each generation run, reflect on what could improve future runs.

Capture improvements to:

- Templates (missing sections, better structure)
- Discovery logic (missed patterns, false positives)
- Content extraction (shortcodes not handled, formatting issues)

Update process:

1. Note issues encountered during generation
2. Identify patterns that caused problems
3. Update relevant files:
   - `SKILL.md` - Workflow, decision tree, quick reference updates
   - `templates/*.md` - Template improvements
   - `discovery.md` - Detection logic updates

- `testing.md` - New validation checks

4. Document the improvement in commit message

Example self-improvement:

```
Issue: libFuzzer skill missing sanitizer flags table
Fix: Updated templates/fuzzer-skill.md to include ## Compiler Flags section
```

Example Usage

Full Discovery and Generation

```
User: "Generate skills from the testing handbook"


1. Locate handbook (check common locations, ask user, or clone)
2. Read discovery.md for methodology
3. Scan handbook at {handbook_path}/content/docs/
4. Build candidate list with types
5. Present plan to user
6. On approval, generate each skill using appropriate template
7. Validate generated skills
8. Update main README.md with generated skills table
9. Update README.md Skills Cross-Reference graph from Related Skills sections
10. Self-improve: note any template/discovery issues for future runs
11. Report results
```

Single Section Generation

```
User: "Create a skill for the libFuzzer section"


1. Read /testing-handbook/content/docs/fuzzing/c-cpp/10-libfuzzer/
2. Identify type: Fuzzer Skill
3. Read templates/fuzzer-skill.md
4. Extract content, apply template
5. Write to skills/libfuzzer/SKILL.md
6. Validate and report
```

Tips

Do:

- Always present plan before generating
- Use appropriate template for skill type
- Preserve code blocks exactly
- Validate after generation

Don't:

- Generate without user approval
- Skip fetching non-video external resources (use WebFetch)
- Fetch video URLs (YouTube, Vimeo - titles only)
- Include handbook images directly
- Skip validation step
- Exceed 500 lines per SKILL.md

For first-time use: Start with discovery.md to understand the handbook analysis process.

For template reference: See templates/ directory for skill type templates.

For validation: See testing.md for quality assurance methodology.

Revision #5
Created 2026-02-18 08:40:13 UTC by John
Updated 2026-06-21 20:01:30 UTC by John