CodeQL Triage Playbook

Practical SOP for GitHub Advanced Security

Version v1.0 • October 30, 2025

Build note: This document uses the `minted` package for syntax highlighting. Compile with `pdflatex -shell-escape` (or `xelatex -shell-escape`).

Contents

1 How to Use This SOP

This playbook distills a lightweight, repeatable triage process for CodeQL findings in GitHub. It is optimized for speed, auditability, and clear decisions. Use it as a runbook during daily triage or while debugging failed code scanning runs.

2 Where to Look First

2.1 GitHub Security UI

- Navigate to **Repository** → **Security** → **Code scanning alerts**.
- Open an alert, review the **rule documentation**, and click **Show paths** to trace the source-to-sink flow.

2.2 Actions Run vs. Query Failure

When a scan fails, open the failed **Actions** run and expand the job steps.

- If the failure is in workflow or YAML (checkout, setup, permissions), fix the pipeline first.
- If the failure originates from the CodeQL action or a specific query pack, inspect annotations and logs to isolate the step.

2.3 SARIF In/Out

- You can **upload** third-party SARIF to *Code scanning alerts*.
- You can **export** CodeQL SARIF for downstream processing and dashboards.

3 Minimum Triage Flow (10 Minutes)

1. **Confirm failure location:** workflow/YAML vs. CodeQL analysis.
2. **Open alert** and use **Show paths** to identify root cause (source, propagators, sink).
3. **Read the rule doc:** impact, examples, and suggested fixes (often links to the query).
4. **Decide:** fix now, file an issue, or dismiss with justification.
5. **Log evidence:** link to path trace, commit, PR, and any compensating controls.

4 Dismissal Reasons (Use Sparingly)

If you dismiss, always provide a clear justification and schedule a review.

- **False positive / Not exploitable** in our context (explain why the path is unreachable).
- **Mitigated by design** (e.g., centralized sanitization or enforcement); add references.
- **Test or sample code** (excluded from build/runtime).
- **Legacy with compensating control** (ticket opened, control ID listed, timeline set).

5 Known CodeQL Limitations

- **Language coverage varies;** some languages and frameworks have richer modeling than others.
- **AST/DB modeling is not a full compiler:** framework indirection or meta-programming can obscure flows.
- **Signal vs. noise:** narrower queries reduce false positives but may increase scan time; broader queries raise noise.

6 Speed and Cost Tips

- **Scope the code:** analyze only relevant directories using workflow `paths/paths-ignore`.
- **Cache:** use `actions/cache` for databases and dependencies.
- **Parallelize:** consider matrix builds or larger runners when this meaningfully cuts wall-clock.
- **Prefer built-in suites/packs;** keep custom queries targeted.

7 Handy Commands and Snippets

7.1 gh CLI: List and Inspect Alerts

```
# Set a default repo (once per shell)
gh repo set-default ORG/REPO

# List CodeQL alerts (IDs, rule, state)
gh api repos/:owner/:repo/code-scanning/alerts \
  --jq '.[] | "\(.number)  \(.rule.id)  \(.state)"'

# Get one alert's JSON (replace 123)
gh api repos/:owner/:repo/code-scanning/alerts/123

# Tip: From the Actions UI you can download a run's log archive when debugging.
```

7.2 CodeQL CLI: Local Repro and SARIF

```
# Create a CodeQL database (example: Python)
codeql database create db-python --language=python --source-root .

# Analyze with a suite; emit SARIF
codeql database analyze db-python \
  codeql/python-queries:codeql-suites/python-code-scanning.qls \
  --format=sarifv2 --output=codeql-results.sarif

# Upgrade the database as code changes
codeql database upgrade db-python
```

7.3 GitHub Actions: Add Caching Around CodeQL

```
# Excerpt from a CodeQL job
- name: Cache CodeQL database
  uses: actions/cache@v4
  with:
    path: |
      ~/.codeql
      ./.codeql-db
    key: ${{ runner.os }}-codeql-${{ hashFiles('**/requirements*.txt',
      '**/package-lock.json', '**/poetry.lock') }}

# Typical CodeQL init/analyze
- uses: github/codeql-action/init@v3
  with:
    languages: python, javascript
- uses: github/codeql-action/analyze@v3
```

8 Done-Done Checklist for Each Alert

- Root cause identified via **Show paths**; exploitability assessed.
- Decision recorded (fix/issue/dismiss) with clear justification.

- Evidence captured: rule doc link, query reference if applicable, and commit/PR.
- If dismissed: ticket to re-review later and compensating control noted.
- Any runtime learnings (cache/scope/matrix tweaks) captured in the workflow PR.

Acknowledgments

This SOP is meant to be adapted to your repository conventions and governance. Keep the document close to the code: propose improvements via pull requests as you learn from real alerts.