A Practical Overview of GitHub Advanced Security (GHAS)

Build note: This document uses the `minted` package. Compile with *shell-escape* enabled, for example: `pdflatex -shell-escape ghas-overview.tex`

1 What is GHAS?

GitHub Advanced Security (GHAS) adds first-party application security features on top of GitHub:

- **Dependabot** for Software Composition Analysis (SCA) and automated update PRs.
- **Secret scanning** to detect exposed credentials, with optional push protection.
- **Code scanning** powered by **CodeQL** for static analysis and data-flow security checks.

GHAS is free for public repositories and typically requires enterprise licensing for private/internal repositories. It integrates natively with pull requests, security alerts, and repository insights.

2 Enablement checklist

1. **Organization-level:** In *Settings →Code security and analysis*, enable GHAS features for the org and choose default policies (e.g., default enablement for new repos).
2. **Repository-level:** In each repo's *Settings →Code security and analysis*, enable:
   - Dependabot (*Dependency graph*, *Dependabot alerts*).
   - Secret scanning (plus *Push protection* if available).
   - Code scanning (add CodeQL or upload SARIF from a third-party scanner).
3. **Permissions & branches:** Protect your default branch, require status checks for CodeQL and tests, and restrict who can dismiss alerts.

3 Dependabot (SCA + automated updates)

Dependabot alerts surface known vulnerabilities in dependencies. Dependabot updates can automatically open pull requests to bump versions.

3.1 Recommended practices

- Centralize configuration in `.github/dependabot.yml` (case-sensitive).
- Run on a predictable schedule and pin time zone.
- Use `allow`/`ignore` rules to control scope.
- Label PRs and auto-assign reviewers to streamline triage.
- Consider `versioning-strategy` per ecosystem (pin, widen, increase, lockfile-only).

Listing 1: `.github/dependabot.yml` (Node.js example)

```yaml
version: 2
updates:
  - package-ecosystem: npm
    directory: "/"
    schedule:
      interval: weekly
      time: "09:00"                # HH:MM (UTC unless timezone set)
      timezone: "America/New_York"
    allow:
      - dependency-name: lodash
      - dependency-name: "react*"
    ignore:
      - dependency-name: express
        versions: ["4.x", "5.x"]
    reviewers: ["org/sec-team"]
    assignees: ["octocat"]
    labels: ["deps", "automated"]
    versioning-strategy: increase  # pin | widen | increase | lockfile-only
    rebase-strategy: auto          # auto | disabled
```

4 Secret scanning (detection & protection)

Secret scanning detects credentials (API keys, tokens, etc.) in git history, PRs, and issues. With push protection, contributors are blocked from pushing known secrets unless they explicitly bypass with a justification.

4.1 Recommended practices

- Turn on secret scanning and push protection for all repos (public and private).
- Use repository-level exclusions for documentation paths or test fixtures that may contain false positives.
- Triage new alerts quickly; revoke exposed credentials at the source.

Listing 2: `.github/secret_scanning.yml` (ignore paths)

```yaml
# File must be on the default branch to take effect
paths-ignore:
  - "docs/**"
  - "examples/**"
  - "website/static/**"
```

Tip: Keep ignore lists as narrow as possible to avoid blind spots. If you must store fake tokens for demos, mark them clearly and prefer generated test credentials.

5 Code scanning with CodeQL

CodeQL builds a semantic database of your code and evaluates it with security query packs. It supports many languages (e.g., C/C++, C#, Go, Java/Kotlin, JavaScript/TypeScript, Python, Ruby). Results appear in the repository's *Security* tab and as PR annotations.

5.1 Quick-start workflow

Add a standard CodeQL workflow to `.github/workflows/codeql.yml`. Use matrix builds for multi-language repos. Require it as a PR check.

Listing 3: `.github/workflows/codeql.yml`

```yaml
name: "CodeQL"

on:
  push:
    branches: ["main"]
  pull_request:
    branches: ["main"]
  schedule:
    - cron: "0 7 * * 1"   # weekly Monday 07:00 UTC

jobs:
  analyze:
    permissions:
      security-events: write
      contents: read
      actions: read
    runs-on: ubuntu-latest

    strategy:
      matrix:
        language: [ "javascript-typescript", "python" ]
        # Other options: "cpp", "csharp", "go", "java-kotlin", "ruby"
    steps:
      - uses: actions/checkout@v4

      - name: Initialize CodeQL
        uses: github/codeql-action/init@v3
        with:
          languages: ${{ matrix.language }}
          config-file: ./.github/codeql/codeql-config.yml

      - name: Autobuild
        uses: github/codeql-action/autobuild@v3

      - name: Perform CodeQL Analysis
        uses: github/codeql-action/analyze@v3
        with:
          category: "/language:${{ matrix.language }}"
```

5.2 Repository CodeQL configuration

Use a config file to select query packs, exclude low-signal results, and control path filters.

Listing 4: `.github/codeql/codeql-config.yml`

```yaml
name: "repo codeql config"

# Keep defaults; extend with additional packs as needed.
disable-default-queries: false

queries:
  - uses: security-extended
  # - uses: security-experimental  # Opt-in for bleeding-edge rules

query-filters:
  - exclude:
      severity: warning

paths:
  - src/
paths-ignore:
  - node_modules/
  - "**/*test*"
```

Builds matter: For compiled languages, ensure CodeQL sees a faithful build (consider replacing `autobuild` with explicit build steps). Mismatched or partial builds reduce findings quality.

6 Uploading third-party scanner results (SARIF)

You can ingest SARIF output from other SAST tools into the GitHub Code Scanning UI.

Listing 5: Minimal SARIF upload workflow

```yaml
name: "Upload third-party SARIF"

on:
  workflow_dispatch:
  push:
    branches: ["main"]

jobs:
  upload-sarif:
    permissions:
      security-events: write
      contents: read
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      # Run your third-party scanner here; produce results.sarif
      # - name: Run Scanner
      #   run: ./scanner --format sarif -o results.sarif

      - name: Upload SARIF to Code Scanning
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: results.sarif
```

7 Triage workflow & SLAs

- **Routing:** Use labels (e.g., `security`, `deps`), CODEOWNERS, and auto-assigns to direct alerts to service owners.
- **PR gating:** Require passing CodeQL and tests on protected branches. Consider blocking merges on High/Critical findings or on active secret leaks.
- **Suggested SLAs:**
  - Critical: 24–48 hours
  - High: 3–5 days
  - Medium: 10 business days
  - Low/Informational: backlog with periodic review
- **Suppressions:** Prefer fixing. When not feasible, document risk acceptance in the PR or project tracker; use *dismiss with reason* so institutional knowledge is preserved.

8 Operational guardrails

- **Scale with templates:** Keep reusable workflows in a central `.github` repo and consume via `workflow_call`.
- **Environment parity:** Build and analyze the code paths you run in production; otherwise, you miss issues.
- **Least privilege:** Grant the minimal `GITHUB_TOKEN` permissions needed; set Action job `permissions` explicitly.
- **Noise control:** Start with `security-extended` and tune path filters; review experimental packs in a staging repo first.
- **Dependency hygiene:** Merge low-risk upgrades quickly; batch PRs when ecosystems are noisy.

9 Appendix: quick copy-paste

Enable Code Scanning (one-liner for new repos)

```
gh api \
  -X PUT \
  -H "Accept: application/vnd.github+json" \
  /repos/OWNER/REPO/code-scanning/default-setup
```

Query false positives across repos (CodeQL)

```
gh codeql database analyze --help
# Use "dismiss with reason" in the UI to capture audit context.
```

This overview is designed as a drop-in, working baseline. Tune schedules, query packs, and branch protections to match your risk appetite and developer workflow.