

# GitHub Advanced Security (GHAS) and Its Role in a 16-Gate CI/CD Pipeline

December 6, 2025

## 1 Overview

GitHub Advanced Security (GHAS) secures applications by integrating security directly into the developer workflow using Code Scanning (CodeQL SAST) for code flaws, Secret Scanning for leaked credentials, and Dependency Review (Dependabot) for supply chain risks, all automated within GitHub Actions, providing early, developer-friendly alerts and fixes to shift security left in the SDLC [1, 2, 3, 4, 5].

A concise “elevator pitch” description is:

**GitHub Advanced Security builds SAST, secret detection, and software supply-chain security directly into the GitHub developer workflow using Code Scanning (CodeQL), Secret Scanning, and Dependency Review/Dependabot, surfacing PR-centric alerts and fixes so vulnerabilities are caught and remediated early in the SDLC.**

*Note: AI-generated summaries and mappings may include mistakes. Review and adapt to your organization’s standards, controls, and terminology before publishing as policy or architecture documentation.*

## 2 Product Naming and Scope

GitHub now presents GHAS capabilities as two purchasable products for organizations (in addition to features that are free for public repositories):

- **GitHub Code Security**

- Code scanning (CodeQL and supported third-party scanners).
- Premium Dependabot capabilities (security and version updates).
- Dependency Review.
- Security Overview dashboards.

- **GitHub Secret Protection**

- Secret Scanning for repositories and organization scope.
- Push Protection to block commits containing high-confidence secrets.

Typical enterprise policies:

- Public repositories benefit from a free tier of some security features.
- Private and internal repositories at scale generally require the paid GHAS capabilities to enforce AppSec gates and enterprise reporting.

## 3 Key Components for Application Security

### 3.1 Code Scanning (SAST)

- Uses **CodeQL**, a powerful query engine, to find vulnerabilities and quality issues in your code.
- Scans happen automatically on commits, pull requests (PRs), or schedules, providing near real-time feedback to developers.
- Rules can be tuned by language, repository, or organization to target your threat model.
- Findings are surfaced directly in PRs and in the repository's *Security* tab, with severity, CWE, and recommended remediation guidance [2, 3, 4, 6, 7].

### 3.2 Secret Scanning

- Detects hardcoded credentials (API keys, tokens, passwords) in code, configuration files, and git history.
- **Push Protection** can prevent commits with secrets from entering the repository by blocking the push at the server side.
- Supports:
  - Partner patterns (e.g., major cloud providers, SaaS vendors).
  - Custom patterns for organization-specific tokens and secrets.
- Alerts can be triaged centrally and integrated with incident response workflows.

### 3.3 Dependency Review and Dependabot (SCA)

- Analyzes your project's dependencies (open-source libraries and transitive dependencies) for known vulnerabilities (CVEs).
- **Dependency Review** shows, at PR time, exactly which dependencies were added, upgraded, or removed, along with associated vulnerabilities.
- **Dependabot**:
  - Creates automated PRs to remediate vulnerable or outdated dependencies.
  - Suggests safe update ranges, reducing supply-chain risk.
- Combined, these features help secure the software supply chain and reduce the attack surface of third-party code [1, 2, 3, 4, 6, 7].

## 4 How GHAS Works in the Developer Workflow

### 4.1 Integrated & Automated

- Scans run within existing GitHub Actions and CI/CD pipelines, without forcing developers to switch tools.
- Typical triggers:
  - `push` to any branch or selected branches.
  - `pull_request` events for PR validation.
  - Scheduled scans (e.g., nightly or weekly) for baseline analysis.
- GHAS jobs are simply additional workflows and checks in the same CI/CD system used for builds and tests.

### 4.2 Developer-Focused Feedback Loop

- Alerts appear directly in PRs, next to the code being changed, where developers already collaborate.
- Fixes can be suggested automatically or linked to secure coding guidance, reducing remediation time from months to minutes.
- Developers can:
  - See the impact of a dependency change via Dependency Review.
  - Understand why a CodeQL finding is being raised.
  - Triage false positives (within policies defined by the security team).

### 4.3 Security Overview and Vulnerability Management

- The **Security Overview** provides:
  - Organization-wide view of code scanning alerts, secret scanning alerts, and dependency vulnerabilities.
  - Filtering by repository, severity, ecosystem, and more.
  - Aggregated metrics for tracking security posture over time.
- Security and compliance teams can use this as a data source for:
  - Risk reporting to leadership.
  - Control evidence for audits.
  - Prioritization of remediation work [2, 3, 8, 9, 10, 11, 12, 13].

## 5 Benefits for Application Security

- **Shift Left:** Finds and fixes vulnerabilities early in the SDLC, close to where the code is written.
- **Reduced Friction:** Security is integrated into the same platform developers use for source control and collaboration, reducing reliance on separate security tools.
- **Faster Remediation:** Quick, actionable insights empower developers to fix issues immediately and keep PRs moving.
- **Comprehensive Protection:** Covers
  - Proprietary code via CodeQL SAST.
  - Secrets and configuration via Secret Scanning.
  - Software supply chain via Dependabot and Dependency Review.
- **Better Governance:** Central dashboards, policies, and metrics support regulatory compliance and internal security standards [2, 3, 9, 11, 12, 13].

## 6 GHAS in a 16-Gate CI/CD Pipeline View

This section provides a detailed description of how GHAS fits into a 16-gate CI/CD pipeline. The exact gate names may vary between organizations, but the mapping below assumes a common 16-gate pattern along these lines:

1. Source code version control
2. Optimum branching strategy
3. Static analysis
4. 80% (or higher) code coverage
5. Vulnerability scan
6. Open source / dependency scan
7. Artifact version control
8. Auto-provision infrastructure
9. Immutable servers / images
10. Integration testing
11. Performance testing
12. Build, deploy, testing automated for every commit
13. Automated rollback
14. Automated change order / change management
15. Zero-downtime release

## 16. Feature toggles

GHAS primarily implements and enforces the security aspects of several gates (especially 3, 5, and 6), and supports governance around others (1, 2, 7, 12).

### 6.1 Gate-Level Mapping of GHAS Capabilities

Table 1 shows an example mapping of GHAS features to the 16 gates.

Table 1: Example Mapping of GHAS to a 16-Gate CI/CD Pipeline

Gate	Gate Name	GHAS Feature(s)	Trigger / Integration Point	Blocking / Governance Behavior
1	Source code version control	GitHub repositories with branch protection rules; Secret Scanning (historical)	On every commit and push to GitHub	Requires PRs for changes to protected branches; security policies can mandate that no repositories are created without GHAS enabled for eligible projects.
2	Optimum branching strategy	Code Scanning and Dependency Review tied to PR-based development	<code>pull_request</code> events trigger CodeQL and Dependency Review workflows	Branch protection requires GHAS checks to pass before merging; enforces PR reviews and security checks as standard practice.
3	Static analysis	Code Scanning (CodeQL SAST)	On PRs, pushes to main branches, and scheduled scans	Gate fails if high/critical CodeQL findings remain open; PR cannot be merged until issues are fixed or formally waived under policy.
4	80% code coverage	Not directly GHAS (handled by test coverage tools), but CodeQL results can be combined with test coverage metrics in dashboards	Test workflows run in parallel with CodeQL workflows	Security policy may require that a PR satisfies both coverage thresholds and passes CodeQL checks to satisfy “quality and security” gates together.

Gate	Gate Name	GHAS Feature(s)	Trigger / Integration Point	Blocking / Governance Behavior
5	Vulnerability scan (application)	Code Scanning (security queries), Secret Scanning for configuration files	Same as Gate 3, plus scheduled full scans on default branch	Gate fails when high/critical security vulnerabilities are detected in application code or configuration; organizational policy defines remediation SLAs by severity.
6	Open source / dependency scan (SCA)	Dependency Review, Dependabot security updates, Dependabot alerts	On PRs that modify dependency manifests; scheduled dependency checks; Dependabot PR creation	PRs cannot merge if they introduce new vulnerable dependencies or downgrade to vulnerable versions; Dependabot PRs provide approved remediation paths.
7	Artifact version control	Dependabot and Dependency Review provide SBOM-like visibility into dependency versions; Security Overview tracks vulns by repo/version	At build time and during dependency updates	Security governance can require that only artifacts built from repositories with a “clean” GHAS status (no open critical vulns) are promoted to higher environments.
8–11	Infra & testing gates (auto-provision, immutable servers, integration, performance)	Not directly GHAS features, but GHAS findings inform risk for promotions between environments	CI/CD stages in infrastructure and testing pipelines	Promotion can be conditioned on a GHAS status check (e.g., no new high-severity findings since last deploy); security sign-off uses GHAS dashboards as evidence.

Gate	Gate Name	GHAS Feature(s)	Trigger / Integration Point	Blocking / Governance Behavior
12	Full automation on commit (build, deploy, testing)	GHAS workflows are first-class CI checks alongside build and test jobs	On every commit/PR to mainline branches	A fully automated pipeline includes GHAS scans as required checks; if GHAS fails, the pipeline fails and no deployment is made, preserving “secure by default” automation.
13–15	Rollback, change management, zero-downtime release	Security posture provided by GHAS informs go/no-go decisions and rollback triggers	Pre-deployment governance checks; post-deployment monitoring	Change advisory boards and automated policies can require GHAS status to be “green” (no unresolved high severity alerts) as a precondition for production deploys.
16	Feature toggle governance	GHAS status can be used as an input to decisions about enabling new features	When toggling features or performing canary rollouts	If a feature branch or feature toggle is associated with new code that still has open GHAS alerts, policies may restrict enabling the feature in production.

## 6.2 Narrative Flow: GHAS Across the 16 Gates

Below is a narrative describing how GHAS fits into the end-to-end CI/CD flow:

- Developer commit (Gates 1–2).** Developers work in feature branches in GitHub. Branch protection and GHAS enforcement (CodeQL, Dependency Review) are required for merges to main branches, aligning with source control standards and branching strategies.
- Pull request creation (Gates 2–3–6).** When a PR is opened:
  - Code Scanning (CodeQL) runs as a GitHub Actions workflow.
  - Dependency Review runs automatically, showing any new or upgraded dependencies and their vulnerabilities.
  - Optional: secret scanning can be configured to run as part of CI to quickly highlight any newly introduced secrets.

The PR displays GHAS findings inline, and status checks are configured so that PRs with outstanding high-severity findings cannot be merged.

3. **Quality gates (Gates 3–6).** The CI/CD pipeline treats GHAS checks as first-class gates alongside:

- Unit and integration test results.
- Coverage thresholds.
- Linting and style checks.

If GHAS checks fail (e.g., new critical CodeQL finding or introduction of a vulnerable dependency), the gate fails and the PR must be updated.

4. **Artifact build and promotion (Gates 7–12).** Once gates 3–6 pass, the pipeline builds versioned artifacts. Before promoting artifacts to higher environments:

- GHAS status on the default branch is queried (implicitly via required checks or explicitly via APIs).
- Policies can mandate “no open critical alerts” for the repository before deployment continues.

This ensures artifacts flowing through automated deployment (Gate 12) are not only functionally correct but also meet security standards.

5. **Pre-production and production release (Gates 10–16).** For integration, performance, and production release stages:

- Security teams use the Security Overview dashboards to verify that no new high-severity alerts have appeared since the last release.
- Change management processes (Gate 14) reference GHAS status as a required field or artifact in the change record.
- Zero-downtime and feature toggle strategies (Gates 15–16) can incorporate GHAS data: if a new feature is associated with unresolved GHAS alerts, toggling it on in production may be delayed or blocked.

6. **Ongoing governance and continuous improvement.** Scheduled GHAS scans run on the default branch (and optionally release branches), ensuring new vulnerabilities discovered in the ecosystem or rule sets are surfaced even without new code changes. Metrics from GHAS feed security KPIs, risk reports, and audit evidence.

## References

1. <https://www.youtube.com/watch?v=mtxlvWYjIxc>
2. [https://arctiq.com/hubfs/Arctiq%20Collateral%202024/GitHub%20Advanced%20Security%20\(GHAS\).pdf?hsLang=en](https://arctiq.com/hubfs/Arctiq%20Collateral%202024/GitHub%20Advanced%20Security%20(GHAS).pdf?hsLang=en)
3. <https://opsera.ai/blog/how-github-advanced-security-auto-remediation-transforms-appsec-and->
4. <https://resources.github.com/learn/pathways/security/essentials/application-security-testing>

5. <https://github.com/security/advanced-security>
6. <https://azurecodingarchitect.com/posts/ghas-starting-with-codeql/>
7. <https://www.coveros.com/unleashing-the-power-of-github-advanced-security-ghas/>
8. <https://github.blog/news-insights/product-news/introducing-github-vulnerability-management->
9. <https://github.com/resources/articles/application-security-testing>
10. <https://github.blog/security/application-security/application-security-orchestration-with-g>
11. <https://blog.cellenza.com/en/security/retrospective-analysis-on-github-advanced-security-gha>
12. <https://resources.github.com/webcasts/github-at-black-hat/>
13. <https://www.increment.inc/post/github-advanced-security>