

Application Security Program

Powered by GitHub Advanced Security (GHAS)

Comprehensive Framework for Enterprise Security

Integrating Security into the Developer Workflow

Version 1.0

December 17, 2025

Contents

Document Metadata **ii**

1 Executive Summary **1**

1.1 Program Mission 1

1.2 Strategic Objectives 1

1.3 GHAS Value Proposition 1

2 GitHub Advanced Security Components **1**

2.1 Code Scanning (CodeQL SAST) 1

2.2 Secret Scanning 2

2.3 Dependency Review & Dependabot (SCA) 2

2.4 Security Overview Dashboard 2

3 Governance Framework **2**

3.1 Organizational Structure 2

3.2 Application Risk Classification 3

3.3 Core Security Policies 3

4 CI/CD Pipeline Integration (16-Gate Model) **3**

4.1 GHAS Gate Mapping 3

4.2 Developer Workflow Integration 4

5 Implementation Roadmap **4**

5.1 Phase 1: Foundation (Months 1-6) 4

5.1.1 Governance Activities 4

5.1.2 Technical Implementation 5

5.1.3 Phase 1 Success Criteria 5

5.2 Phase 2: Core Build (Months 7-12) 5

5.2.1 Expansion Activities 5

5.2.2 Developer Enablement 5

5.2.3 Phase 2 Success Criteria 5

5.3 Phase 3: Optimization (Months 13-18) 6

5.3.1 Comprehensive Coverage 6

5.3.2	Process Optimization	6
5.3.3	Phase 3 Success Criteria	6
6	Metrics and Reporting	6
6.1	Key Performance Indicators	6
6.2	Reporting Cadence	6
7	Appendices	7
7.1	Vulnerability Remediation SLAs	7
7.2	GHAS Configuration Checklist	7
7.3	Reference Standards	7
8	Conclusion	7
8.1	Program Success Factors	8

Document Metadata

Document Title	Application Security Program
Version	1.0
Effective Date	December 17, 2025
Primary Tooling	GitHub Advanced Security (GHAS)
Purpose	Comprehensive framework for integrating application security controls into the developer workflow.

1 Executive Summary

This Application Security Program establishes a comprehensive framework for protecting software applications throughout their lifecycle by leveraging GitHub Advanced Security (GHAS) as the foundational security tooling platform. The program integrates security directly into the developer workflow, enabling shift-left security practices while maintaining development velocity.

1.1 Program Mission

To systematically reduce application security risk across the enterprise by embedding GHAS-powered security into every phase of the software development lifecycle, fostering a security-conscious culture, and establishing measurable controls that protect organizational assets from application-layer threats.

1.2 Strategic Objectives

- Shift-Left Security: Find and fix vulnerabilities early using CodeQL SAST, Secret Scanning, and Dependency Review
- Developer Enablement: Provide security feedback in pull requests where developers already work
- Supply Chain Protection: Secure third-party dependencies with Dependabot and SCA capabilities
- Automated Governance: Enforce security policies through CI/CD pipeline integration
- Measurable Progress: Track security posture with Security Overview dashboards

1.3 GHAS Value Proposition

GitHub Advanced Security provides native security capabilities integrated directly into GitHub's developer platform, eliminating the friction of external security tools and enabling security-at-speed for modern development teams.

2 GitHub Advanced Security Components

GHAS delivers three core security capabilities that form the foundation of this program, available as two purchasable products for private repositories: GitHub Code Security and GitHub Secret Protection.

2.1 Code Scanning (CodeQL SAST)

CodeQL is a powerful semantic code analysis engine that queries code as data to find security vulnerabilities and code quality issues.

- Analyzes source code for vulnerabilities without execution
- Supports 10+ languages including JavaScript, Python, Java, C/C++, Go, Ruby
- Runs automatically on commits, pull requests, and scheduled scans
- Findings displayed inline in PRs with severity, CWE, and remediation guidance

- Customizable query suites for organization-specific threat models

2.2 Secret Scanning

Detects hardcoded credentials, API keys, tokens, and other secrets in code and git history before they can be exploited.

- Partner patterns detect secrets from major cloud providers and SaaS vendors
- Custom patterns enable organization-specific secret detection
- Push Protection blocks commits containing secrets before they enter the repository
- Historical scanning detects secrets already in git history
- Alerts integrate with incident response workflows

2.3 Dependency Review & Dependabot (SCA)

Software Composition Analysis capabilities secure the software supply chain by identifying and remediating vulnerable dependencies.

- Dependency Review: Shows vulnerability impact of dependency changes at PR time
- Dependabot Alerts: Notifies about known vulnerabilities (CVEs) in dependencies
- Dependabot Security Updates: Creates automated PRs to remediate vulnerable dependencies
- Dependabot Version Updates: Keeps dependencies current to reduce exposure window
- Supports all major package ecosystems: npm, pip, Maven, NuGet, etc.

2.4 Security Overview Dashboard

Provides organization-wide visibility into security posture with aggregated metrics, filtering by repository/severity/ecosystem, and trend analysis for leadership reporting and audit evidence.

3 Governance Framework

3.1 Organizational Structure

The Application Security Program operates under a tiered governance model with clear accountability and decision-making authority.

Table 1: Organizational Roles and Responsibilities

Role	Responsibilities	GHAS Focus
Executive Sponsor	Program authority, budget, risk acceptance	Security Overview review, policy approval
AppSec Program Lead	Strategy, operations, team leadership	GHAS configuration, rollout, metrics
Security Champions	Team liaison, first-line triage	Alert triage, developer coaching

Role	Responsibilities	GHAS Focus
Development Teams	Secure coding, remediation	Fix alerts in PRs, merge Dependabot PRs

3.2 Application Risk Classification

Applications are classified into risk tiers to enable appropriate security investment based on business criticality and data sensitivity.

Table 2: Application Risk Classification Tiers

Tier	Criteria	GHAS Requirements
Tier 1 (Critical)	Customer-facing, PII/PCI data, regulatory scope, high revenue impact	All GHAS features enabled, blocking gates, mandatory PR checks
Tier 2 (High)	Internal sensitive data, moderate business impact, partner integrations	All GHAS features, advisory gates, critical findings block
Tier 3 (Medium)	Internal tools, limited data access, low business impact	CodeQL default, Dependabot alerts, informational gates
Tier 4 (Low)	Test/dev environments, no sensitive data, minimal impact	Basic scanning, no blocking gates, periodic review

3.3 Core Security Policies

- GHAS Enablement Policy: All repositories must have appropriate GHAS features enabled based on tier
- Branch Protection Policy: Protected branches require passing GHAS checks before merge
- Vulnerability Remediation Policy: SLAs based on severity (Critical: 7 days, High: 30 days, Medium: 90 days)
- Secret Exposure Policy: Push Protection enabled; exposed secrets require immediate rotation
- Dependency Management Policy: Dependabot PRs must be reviewed within 14 days

4 CI/CD Pipeline Integration (16-Gate Model)

GHAS integrates security checks throughout the CI/CD pipeline, serving as the enforcement mechanism for multiple quality and security gates. This section maps GHAS capabilities to a comprehensive 16-gate pipeline model.

4.1 GHAS Gate Mapping

Table 3: GHAS Mapping to 16-Gate CI/CD Model

Gate	Gate Name	GHAS Feature	Blocking Behavior
1	Source Code Version Control	Secret Scanning (historical)	GHAS enabled on repo creation
2	Branching Strategy	Branch Protection + GHAS	PR required with passing checks
3	Static Analysis (SAST)	Code Scanning (CodeQL)	Blocks on high/critical
4	Code Coverage (80%+)	External (combined check)	Both coverage + CodeQL pass
5	Vulnerability Scan	Code + Secret Scanning	Blocks on secrets/vulns
6	Dependency Scan (SCA)	Dependency Review	Blocks vulnerable deps
7	Artifact Version Control	Dependabot + Security Overview	Clean status for promotion
8-11	Infra & Testing Gates	GHAS status informs risk	Promotion requires clean status
12	Full Automation	GHAS as required check	Pipeline fails if GHAS fails
13-15	Rollback & Release Gates	Security Overview status	Green status for prod deploy
16	Feature Toggle Governance	Per-feature GHAS status	Block toggle if alerts open

4.2 Developer Workflow Integration

GHAS seamlessly integrates into existing GitHub-based development workflows:

- Developer Commit: Code pushed to feature branch triggers CodeQL workflow
- Pull Request Created: CodeQL, Dependency Review, and Secret Scanning run automatically
- Inline Feedback: Findings appear as annotations directly in the PR diff view
- Status Checks: Branch protection prevents merge until GHAS checks pass
- Merge to Main: Clean merge triggers scheduled scans and updates Security Overview
- Continuous Monitoring: Scheduled scans detect new vulnerabilities in existing code

5 Implementation Roadmap

The GHAS-powered security program is implemented in three phases over 18 months, progressing from foundation to full maturity.

5.1 Phase 1: Foundation (Months 1-6)

Objective: Establish governance, enable GHAS for critical applications, achieve quick wins.

5.1.1 Governance Activities

- Secure executive sponsorship and budget approval for GHAS licenses
- Establish AppSec governance committee with monthly cadence

- Define and publish GHAS enablement policy and vulnerability SLAs
- Complete application portfolio inventory and tier classification

5.1.2 Technical Implementation

- Enable GitHub Code Security and Secret Protection for organization
- Configure CodeQL default setup for all Tier 1 repositories
- Enable Push Protection for Secret Scanning organization-wide
- Configure Dependabot alerts and security updates for Tier 1-2
- Implement branch protection rules requiring GHAS status checks

5.1.3 Phase 1 Success Criteria

- 100% of Tier 1 applications under CodeQL scanning
- Push Protection enabled for all repositories
- Security Champion identified for each development team
- Baseline vulnerability metrics established
- All policies approved and published

5.2 Phase 2: Core Build (Months 7-12)

Objective: Expand coverage, implement blocking gates, mature developer experience.

5.2.1 Expansion Activities

- Extend CodeQL coverage to all Tier 2 applications
- Configure advanced CodeQL queries for custom security requirements
- Implement custom secret patterns for organization-specific tokens
- Enable Dependency Review action with blocking for vulnerable dependencies
- Configure required status checks for protected branches

5.2.2 Developer Enablement

- Launch security awareness training program (100% developer completion)
- Create internal documentation hub for GHAS usage and triage guidance
- Establish Security Champion advanced training curriculum
- Implement feedback mechanism for false positive reporting

5.2.3 Phase 2 Success Criteria

- 100% of Tier 1-2 applications under full GHAS coverage
- Blocking gates enforced for all Tier 1 applications
- SLA compliance $\geq 85\%$ for vulnerability remediation
- Developer security training completion 100%
- Mean time to remediate Critical findings <10 days

5.3 Phase 3: Optimization (Months 13-18)

Objective: Achieve full coverage, optimize processes, establish continuous improvement.

5.3.1 Comprehensive Coverage

- Extend GHAS to all Tier 3-4 applications
- Implement organization-wide security policies via GitHub Enterprise
- Configure automated SBOM generation using Dependency Graph
- Integrate GHAS data with enterprise SIEM/SOAR platforms

5.3.2 Process Optimization

- Tune CodeQL queries to reduce false positive rate below 15%
- Implement automated triage rules for common finding patterns
- Establish automated compliance evidence collection from Security Overview
- Develop custom dashboards for leadership and audit reporting

5.3.3 Phase 3 Success Criteria

- 100% application portfolio coverage achieved
- SLA compliance $\geq 95\%$ across all tiers
- False positive rate $< 15\%$
- Developer satisfaction with security tooling $\geq 4.0/5.0$
- Automated compliance reporting operational

6 Metrics and Reporting

6.1 Key Performance Indicators

Table 4: Key Performance Indicators

Metric	Data Source	Target
Open Critical/High Findings	Security Overview dashboard	Decreasing trend
Mean Time to Remediate (MTTR)	Code scanning alerts timeline	Critical: < 7 days
SLA Compliance Rate	Calculated from alert ages	$> 95\%$
Secret Scanning Coverage	Organization settings	100%
Dependabot PR Merge Rate	Dependabot activity	$> 90\%$ within 14 days
Push Protection Bypass Rate	Secret scanning audit log	$< 5\%$
False Positive Rate	Dismissed as false positive	$< 15\%$

6.2 Reporting Cadence

- Weekly: Security Champion sync - review new alerts, triage support

- Monthly: Executive dashboard - Security Overview metrics, trend analysis
- Quarterly: Steering committee - program progress, roadmap updates, resource needs
- Annually: Maturity assessment - capability evaluation against OWASP SAMM

7 Appendices

7.1 Vulnerability Remediation SLAs

Table 5: Vulnerability Remediation SLAs by Tier

Severity	Tier 1 SLA	Tier 2 SLA	Tier 3-4 SLA
Critical	7 days	14 days	30 days
High	30 days	45 days	90 days
Medium	90 days	120 days	180 days
Low	180 days	180 days	Best effort

7.2 GHAS Configuration Checklist

- Enable GHAS at organization level
- Configure default CodeQL setup for all new repositories
- Enable Secret Scanning with Push Protection
- Configure custom secret patterns for internal tokens
- Enable Dependabot alerts and security updates
- Configure Dependency Review GitHub Action
- Set up branch protection rules with required status checks
- Configure Security Overview access for security team
- Set up alert notification routing to appropriate teams
- Configure audit log streaming for compliance

7.3 Reference Standards

- OWASP ASVS - Application Security Verification Standard
- OWASP SAMM - Software Assurance Maturity Model
- NIST SSDF - Secure Software Development Framework (SP 800-218)
- CIS Controls - Center for Internet Security Controls
- BSIMM - Building Security In Maturity Model

8 Conclusion

This Application Security Program, powered by GitHub Advanced Security, provides a comprehensive framework for systematically reducing application security risk across the enterprise. By embedding security directly into the developer workflow through CodeQL SAST, Secret Scanning, and Dependency Review, organizations can achieve shift-left security without sacrificing development velocity.

8.1 Program Success Factors

- Executive Commitment: Sustained sponsorship, adequate licensing, and organizational authority
- Developer Experience: Security integrated into existing workflows, not bolted on as friction
- Risk-Based Approach: Prioritize coverage and gates based on application criticality
- Measurable Progress: Leverage Security Overview for continuous visibility and improvement
- Cultural Change: Security Champions and training build organization-wide capability

Application security is not a destination but a journey.

The goal is not perfection but continuous improvement—systematically reducing risk while enabling the business to innovate.