GitHub Advanced Security (GHAS)

Documentation Roadmap

Enterprise Implementation Guide

Secret Scanning → Code Scanning → Dependabot

Prepared for:	*[Company Name]*
Program:	*[AppSec Program Name]*
Document Type:	Implementation Roadmap
Classification:	Internal Use Only

Version: 3.0
Date: January 19, 2026
Status: Active

Document Control

Version History

Version	Date	Changes	Author
1.0	2024-01-15	Initial document creation	AppSec Team
1.5	2024-03-01	Added governance framework and metrics	AppSec Team
2.0	2024-06-01	Comprehensive enhancement: added automation, API examples, troubleshooting, change management, and expanded all sections	AppSec Team
3.0	2024-09-01	Added official GitHub documentation cross-references and hyperlinks throughout; added Quick Reference appendix	AppSec Team

Documentation Reference Note

About Official Documentation Links

This document includes comprehensive cross-references to official GitHub documentation. All URLs reference **docs.github.com** and are current as of the document date. GitHub documentation is regularly updated; verify links if accessing this document significantly after publication.

Primary Documentation Portal: https://docs.github.com/en/code-security

Enterprise Cloud Documentation: https://docs.github.com/en/enterprise-cloud@latest/code-security

Reviewers and Approvers

Role	Name	Date	Status
AppSec Lead	[Name]		Pending
Engineering Director	[Name]		Pending
CISO	[Name]		Pending

Distribution List

- Application Security Team
- Engineering Leadership
- Platform Engineering
- Development Team Leads
- Compliance and Risk Management

Contents

1 Quick Reference: Official Documentation Links

This section provides a consolidated reference to all official GitHub documentation used throughout this roadmap. Use these links as your primary source for detailed technical guidance.

1.1 Core GHAS Documentation

GitHub Advanced Security Overview

- **About GHAS**: https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security
- **Security Features Overview**: https://docs.github.com/en/code-security/getting-started/github-security-features
- **Adopting GHAS at Scale**: https://docs.github.com/en/enterprise-cloud@latest/code-security/adopting-github-advanced-security-at-scale/introduction-to-adopting-github-advanced-security-at-scale
- **Code Security Portal**: https://docs.github.com/en/code-security

1.2 Security Configurations and Organization Settings

Enabling Security at Scale

- **About Enabling Security Features at Scale**: https://docs.github.com/en/code-security/securing-your-organization/introduction-to-securing-your-organization-at-scale/about-enabling-security-features-at-scale
- **Choosing Security Configurations**: https://docs.github.com/en/code-security/securing-your-organization/introduction-to-securing-your-organization-at-scale/choosing-a-security-configuration-for-your-repositories
- **Managing Org Security Settings**: https://docs.github.com/en/organizations/keeping-your-organization-secure/managing-security-settings-for-your-organization/managing-security-and-analysis-settings-for-your-organization

1.3 Security Overview and Reporting

Security Overview Dashboard

- **About Security Overview**: https://docs.github.com/en/code-security/security-overview/about-security-overview
- **Viewing Security Insights**: https://docs.github.com/en/code-security/security-overview/viewing-security-insights
- **Assessing Security Risk**: https://docs.github.com/en/code-security/security-overview/assessing-code-security-risk
- **Auditing Security Alerts**: https://docs.github.com/en/code-security/getting-started/auditing-security-alerts

1.4 Secret Scanning Documentation

Secret Scanning & Push Protection

- **Secret Scanning Portal**: https://docs.github.com/en/code-security/secret-scanning
- **About Push Protection**: https://docs.github.com/en/code-security/secret-scanning/introduction/about-push-protection
- **Enabling Secret Scanning**: https://docs.github.com/en/code-security/secret-scanning/enabling-secret-scanning-features
- **Enabling Push Protection**: https://docs.github.com/en/code-security/secret-scanning/enabling-secret-scanning-features/enabling-push-protection-for-your-repository
- **Push Protection CLI**: https://docs.github.com/en/code-security/secret-scanning/working-with-secret-scanning-and-push-protection/working-with-push-protection-from-the-command-line
- **Push Protection for Users**: https://docs.github.com/en/code-security/secret-scanning/working-with-secret-scanning-and-push-protection/push-protection-for-users
- **Working with Secret Scanning**: https://docs.github.com/en/code-security/secret-scanning/working-with-secret-scanning-and-push-protection

1.5 Code Scanning Documentation

Code Scanning & CodeQL

- **Code Scanning Portal**: https://docs.github.com/en/code-security/code-scanning
- **About CodeQL**: https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning-with-codeql
- **Default Setup**: https://docs.github.com/en/code-security/code-scanning/enabling-code-scanning/configuring-default-setup-for-code-scanning
- **Advanced Setup**: https://docs.github.com/en/code-security/code-scanning/creating-an-advanced-setup-for-code-scanning
- **Query Suites**: https://docs.github.com/en/code-security/code-scanning/managing-your-code-scanning-configuration/codeql-query-suites
- **CodeQL CLI**: https://docs.github.com/code-security/secure-coding/about-codeql-code-scanning-in-your-ci-system
- **Compiled Languages**: https://docs.github.com/en/code-security/code-scanning/creating-an-advanced-setup-for-code-scanning/codeql-code-scanning-for-compiled-languages

1.6 Dependabot Documentation

Dependabot Alerts & Updates

- **Dependabot Portal**: https://docs.github.com/en/code-security/dependabot
- **About Dependabot Alerts**: https://docs.github.com/code-security/dependabot/dependabot-alerts/about-dependabot-alerts
- **Viewing Alerts**: https://docs.github.com/en/code-security/dependabot/dependabot-alerts/viewing-and-updating-dependabot-alerts
- **Configuring Alerts**: https://docs.github.com/en/code-security/dependabot/dependabot-alerts/configuring-dependabot-alerts
- **About Security Updates**: https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/about-dependabot-security-updates
- **Configuring Security Updates**: https://docs.github.com/github/managing-security-vulnerabilities/configuring-dependabot-security-updates

1.7 Supply Chain Security Documentation

Dependency Graph & Review

- **About Dependency Graph**: https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph
- **Configuring Dependency Graph**: https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/configuring-the-dependency-graph
- **About Dependency Review**: https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-dependency-review
- **Dependency Review Action Config**: https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/customizing-your-dependency-review-action-config

2 Executive Summary

This document provides a comprehensive, documentation-first roadmap for implementing GitHub Advanced Security (GHAS) across *[Company Name]*. The implementation follows a structured, phased approach designed to maximize security value while minimizing disruption to development workflows.

Primary Documentation Reference

Start here for official GitHub guidance:

- **GHAS Overview**: https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security
- **Adopting at Scale**: https://docs.github.com/en/enterprise-cloud@latest/code-security/adopting-github-advanced-security-at-scale/introduction-to-adopting-github-advanced-security-at-scale

2.1 Strategic Objectives

1. **Credential Leakage Prevention**: Deploy secret scanning and push protection to prevent sensitive credentials from entering version control, reducing the primary attack vector for supply chain compromises. *See: Secret Scanning Documentation*
2. **Vulnerability Detection at Scale**: Implement CodeQL-based code scanning to identify security vulnerabilities during the development lifecycle, shifting security left. *See: About Code Scanning with CodeQL*
3. **Supply Chain Security**: Enable comprehensive dependency visibility and automated vulnerability remediation through Dependabot integration. *See: Dependabot Documentation*
4. **Certification Readiness**: Prepare team members to pass GitHub Advanced Security certification exams (e.g., GH-500) through hands-on implementation experience.
5. **Measurable Security Posture**: Establish quantifiable metrics and governance frameworks to demonstrate continuous security improvement. *See: About Security Overview*

2.2 Implementation Priority Matrix

Capability	Business Value	Priority	Timeline
Secret Scanning	Prevents credential exposure; immediate ROI	P0	Weeks 1–3
Push Protection	Blocks secrets before commit	P0	Weeks 2–3
Code Scanning (Default)	Broad vulnerability coverage	P1	Weeks 4–5
Code Scanning (Advanced)	Deep analysis for critical repos	P1	Weeks 5–6
Dependabot Alerts	Supply chain visibility	P2	Weeks 7–8
Dependabot Updates	Automated remediation	P2	Week 8+

2.3 Expected Outcomes

Upon completion of this roadmap, *[Company Name]* will achieve:

- 100% secret scanning coverage across all production repositories
- Push protection preventing credential commits at the source
- Automated vulnerability detection integrated into CI/CD pipelines
- Complete dependency inventory with known vulnerability tracking
- Documented governance framework with defined SLAs and escalation paths
- Weekly metrics reporting demonstrating security posture improvement
- Trained personnel capable of operating and optimizing GHAS capabilities

2.4 Resource Requirements

Resource	Description	Commitment
AppSec Engineer	Primary implementation lead	80% for 8 weeks
Platform Engineer	Infrastructure and automation support	20% for 8 weeks
Engineering Leads	Policy review and team coordination	10% for 8 weeks
Development Teams	Workflow integration and feedback	5% ongoing

3 Purpose and Outcomes

This document defines a comprehensive, documentation-first roadmap for deploying GitHub Advanced Security capabilities at enterprise scale.

3.1 Primary Goals

1. **Certification Preparation**: Map study activities to hands-on implementation, enabling team members to pass GitHub Advanced Security certification exams with practical experience rather than theoretical knowledge alone.
2. **Rapid Value Delivery**: Apply GHAS capabilities to *[Company Name]* in a sequence that maximizes immediate risk reduction while building toward comprehensive coverage.
3. **Sustainable Operations**: Establish governance frameworks, metrics, and operational cadences that enable long-term program success without requiring constant executive attention.
4. **Implementation Priority**:
   (a) **Secret Scanning** — Highest immediate business value through credential leakage prevention and rapid incident response when leaks occur.
   (b) **Code Scanning (CodeQL)** — High program impact for vulnerability detection; significant exam emphasis and developer experience improvement.
   (c) **Dependabot** — Supply chain visibility and automated vulnerability remediation for third-party dependencies.

3.2 Guiding Principles

3.2.1 Read → Configure → Measure → Enforce

For each capability area, follow this progression:

1. **Read**: Study the official documentation in a deliberate sequence, understanding both the "what" and "why" behind each feature.
2. **Configure**: Implement a minimum viable configuration in a controlled pilot cohort. Validate assumptions and gather feedback before broader rollout.
3. **Measure**: Use security reporting features to quantify adoption, alert volumes, remediation rates, and prevention effectiveness.
4. **Enforce**: Gradually introduce organizational governance controls (merge protection, required reviews, dismissal policies) as maturity allows.

3.2.2 Progressive Hardening

Security controls should be introduced progressively:

- **Phase 1 — Visibility**: Enable detection and alerting without blocking workflows.
- **Phase 2 — Guidance**: Provide clear remediation guidance and triage support.
- **Phase 3 — Soft Enforcement**: Introduce warnings and recommendations in PR workflows.
- **Phase 4 — Hard Enforcement**: Block merges and deployments based on policy violations.

3.2.3 Developer Experience Focus

Security tooling must enhance, not hinder, developer productivity:

- Alerts must be actionable with clear remediation guidance
- False positive rates must be actively managed and minimized
- Bypass mechanisms must exist for legitimate exceptions
- Feedback loops must enable continuous tool improvement

3.3 Success Criteria

Metric	Target	Timeframe
Secret scanning coverage	100% of in-scope repositories	Week 3
Push protection enabled	100% Tier 0/1 repositories	Week 3
Code scanning coverage	90%+ of in-scope repositories	Week 6
Mean time to remediate (Critical)	< 7 days	Ongoing
Mean time to remediate (High)	< 30 days	Ongoing
False positive rate	$< 10\%$ of total alerts	Ongoing
Developer satisfaction score	$> 3.5/5.0$	Quarterly

4 Scope, Assumptions, and Prerequisites

4.1 Scope Definition

4.1.1 In Scope

This roadmap covers the following capabilities and activities:

- **Organization-Level Configuration**:
  - Security configurations (policy-as-code)
  - Default security settings for new repositories
  - Organization-wide enablement patterns
  - Custom security policies
- **Secret Scanning**:
  - Repository and organization enablement
  - Push protection configuration and bypass policies
  - Custom pattern development for internal tokens
  - Alert triage and remediation workflows
  - Integration with incident response processes
- **Code Scanning**:
  - Default setup deployment at scale
  - Advanced setup for complex repositories
  - CodeQL query suite selection and customization
  - SARIF integration for third-party tools
  - Alert operations and triage workflows
  - Merge protection policies
- **Dependabot**:
  - Dependency graph enablement and accuracy
  - Vulnerability alert configuration
  - Security update automation
  - Version update policies
  - PR management and auto-merge strategies
- **Governance and Operations**:
  - Metrics and reporting frameworks
  - SLA definitions and enforcement
  - Audit and compliance evidence collection
  - Continuous improvement processes

4.1.2 Out of Scope

The following items are explicitly excluded from this roadmap:

- Third-party SAST/DAST tool integration (beyond SARIF import)
- Runtime application security monitoring (RASP)
- Container image scanning (Trivy, Snyk Container, etc.)
- Infrastructure-as-Code security scanning
- Penetration testing and red team activities
- Security awareness training program development
- Vendor security assessment processes

4.2 Assumptions

ID	Assumption	Risk if False
A1	*[Company Name]* uses GitHub Enterprise Cloud with GHAS licensed and enabled	High
A2	AppSec team has or will obtain organization admin permissions	High
A3	Engineering teams can modify repository workflows as needed	Medium
A4	Repository ownership is documented and current	Medium
A5	Pilot cohort repositories are representative of the broader portfolio	Low
A6	Development teams have capacity to respond to security alerts	Medium
A7	Leadership supports enforcement of security policies	High

4.3 Prerequisites Checklist

Complete the following prerequisites before beginning implementation:

Category	Prerequisite	Owner	Status
Inventory	Complete repository inventory with criticality tiers	AppSec	☐
	Repository ownership mapping (team, Slack, escalation)	Engineering	☐
	Technology stack documentation per repository	Engineering	☐
Access	Organization admin permissions for AppSec lead	IT/Admin	☐
	Security manager role assignments	IT/Admin	☐
	API access tokens for automation	Platform	☐
Policy	Risk tolerance thresholds defined and approved	Leadership	☐
	Triage SLA definitions approved	AppSec	☐
	Exception process documented	AppSec	☐
Infrastructure	Pilot cohort of 5–10 repositories selected	AppSec	☐
	Test organization available for experimentation	Platform	☐

4.4 Stakeholder RACI Matrix

Activity	AppSec	Platform	Dev Teams	Leadership
Security configuration design	R/A	C	C	I
Pilot implementation	R/A	C	C	I
Org-wide rollout	R/A	R	C	I
Alert triage (initial)	R/A	–	C	I
Alert remediation	C	–	R/A	I
Policy enforcement decisions	R	C	C	A
Metrics reporting	R/A	C	I	I
Exception approvals	R	–	C	A

Legend: R = Responsible, A = Accountable, C = Consulted, I = Informed

5 Roadmap Overview

5.1 Workstream Definitions

Workstream	Description	Lead
WS1: Foundation	Organization configuration, security configurations, reporting baseline, governance model	AppSec Lead
WS2: Secret Scanning	Push protection, custom patterns, remediation workflows, incident integration	AppSec Engineer
WS3: Code Scanning	CodeQL adoption, alert operations, query tuning, merge protection	AppSec Engineer
WS4: Dependabot	Dependency visibility, automated updates, PR management, version policies	Platform Engineer
WS5: Program Ops	Metrics, reporting, auditing, continuous improvement, training	AppSec Lead

5.2 Implementation Timeline

Phase	Duration	Primary Outcomes
Phase 0: Preparation	Week 0	Prerequisites complete; pilot cohort selected; test org configured
Phase 1: Foundation	Week 1	Security configurations (Baseline + Hardened) defined and tested; Reporting dashboards configured; Ownership model documented; Governance decisions recorded
Phase 2: Secrets	Weeks 2–3	Secret scanning enabled org-wide; Push protection active on Tier 0/1; Custom patterns deployed; Remediation runbook tested; Incident response integration complete
Phase 3: Code	Weeks 4–6	Default setup enabled broadly; Advanced setup on crown jewels; Alert triage workflow operational; Query tuning complete; Merge protection pilot
Phase 4: Supply Chain	Weeks 7–8	Dependency graph accurate; Alerts enabled org-wide; Security updates on priority repos; Version updates configured selectively; PR automation policies active
Phase 5: Optimize	Ongoing	Metrics collection and reporting; Policy tuning; Enforcement expansion; Training and certification; Continuous improvement

5.3 Dependency Map

Critical Dependencies

- **Foundation → All**: Security configurations must be defined before capability-specific rollout
- **Ownership Model → Alert Routing**: Alert routing requires current ownership data
- **Secret Scanning → Code Scanning**: Secret scanning validates alert workflow before higher-volume code scanning
- **Code Scanning Default → Advanced**: Default setup provides baseline before advanced customization
- **Dependabot Alerts → Updates**: Alerts establish baseline before enabling automated updates

5.4 Risk Register

ID	Risk	Impact	Prob.	Mitigation
R1	Alert fatigue overwhelms development teams	High	Medium	Phased rollout; tuning focus; SLA definitions
R2	Push protection causes development friction	Medium	Medium	Bypass policies; escalation paths; training
R3	Repository ownership data is stale	Medium	High	Ownership audit during Phase 0; automated sync
R4	Insufficient capacity for alert remediation	High	Medium	SLA tiers; prioritization framework; triage support
R5	Leadership deprioritizes enforcement	High	Low	Regular metrics reporting; risk quantification
R6	Custom patterns generate excessive false positives	Medium	Medium	Pattern testing in pilot; iterative tuning

6 Foundation: Learn Once, Apply Everywhere

The foundation phase establishes the organizational infrastructure that all subsequent capability deployments will leverage. Investing time here reduces friction and rework during later phases.

Official Documentation Portal

Primary References for Foundation Phase:

- **About GHAS**: https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security
- **Enabling Security at Scale**: https://docs.github.com/en/code-security/securing-your-organization/introduction-to-securing-your-organization-at-scale/about-enabling-security-features-at-scale
- **Security Overview**: https://docs.github.com/en/code-security/security-overview/about-security-overview

6.1 Documentation Reading Sequence

Complete this reading sequence before beginning configuration:

1. **GHAS Overview and Feature Landscape**
   - About GitHub Advanced Security
   - GitHub Security Features
   - Understand the three pillars: Secret Protection, Code Security, Supply Chain
   - Map features to business outcomes and risk reduction goals
   - Identify licensing requirements and feature availability
2. **Enabling Security Features at Scale**
   - About Enabling Security Features at Scale
   - Choosing Security Configurations
   - Managing Organization Security Settings
   - Organization-level enablement patterns
   - Default settings for new repositories
   - Security configurations (policy-as-configuration)

3. **Security Overview and Security Insights**

- About Security Overview
- Viewing Security Insights
- Assessing Security Risk
- Dashboard capabilities and limitations
- Available metrics and export options
- Custom reporting approaches

4. **Governance Controls**

- Auditing Security Alerts
- Alert dismissal policies
- Responsibility assignment
- Audit trail requirements

6.2 Security Configuration Design

6.2.1 Two-Tier Model

Implement a two-tier security configuration model to balance security rigor with operational flexibility:

Configuration	Purpose	Target Repositories
Baseline	Minimum required controls for all repositories. Enables visibility and detection without blocking workflows.	All repositories not assigned to Hardened tier
Hardened	Stricter controls for high-value assets. Includes preventive controls and enforcement policies.	Tier 0 (crown jewels) and Tier 1 (critical) repositories

6.2.2 Configuration Settings Matrix

Setting	Baseline	Hardened
Secret Scanning		
Secret scanning enabled	✓	✓
Push protection enabled	–	✓
Custom patterns enforced	–	✓
Validity checks enabled	✓	✓
Code Scanning		
Default setup enabled	✓	✓
Advanced setup required	–	✓
Security-extended queries	–	✓
Merge protection (critical)	–	✓
Dependabot		
Dependency graph enabled	✓	✓
Vulnerability alerts enabled	✓	✓
Security updates enabled	–	✓
Version updates enabled	–	Optional
Governance		
Delegated dismissal required	–	✓
Dismissal comments required	✓	✓
Alert notification routing	✓	✓

6.3 Ownership Model

6.3.1 Repository Ownership Requirements

Each repository must have documented:

- **Owning Team**: The team responsible for development and maintenance
- **Security Contact**: Individual or role for security-related communications
- **Slack Channel**: Primary communication channel for alerts and discussions
- **Escalation Path**: Management chain for unresolved issues
- **On-Call Rotation**: (For Tier 0/1) Who handles urgent security issues

6.3.2 Ownership Verification Process

1. Export current `CODEOWNERS` files from all repositories
2. Cross-reference with team directory and org chart
3. Identify orphaned repositories (no clear owner)
4. Assign interim owners for orphaned repositories
5. Establish quarterly ownership verification cadence

6.4 Metrics Baseline

Capture baseline metrics before enabling new capabilities to demonstrate program impact:

Metric	Description	Collection Method
Repository count by tier	Total repos per criticality tier	Manual/API
Existing alert backlog	Current open alerts (if any)	Security Overview
Feature enablement rates	% repos with each feature enabled	Security Overview
Mean time to remediate	Historical remediation velocity	Historical data
Developer satisfaction	Baseline satisfaction with security tooling	Survey

6.5 Governance Decisions

Document the following governance decisions before proceeding:

Decision Area	Question to Answer	Approver
Alert Dismissal	Who can dismiss alerts? Under what conditions?	AppSec Lead
Exception Process	How are policy exceptions requested and approved?	Security Director
Enforcement Timeline	When will blocking controls be introduced?	Engineering VP
SLA Definitions	What are remediation timeframes by severity?	AppSec Lead
Bypass Policy	Who can bypass push protection? What oversight exists?	CISO

6.6 Definition of Done: Foundation

Foundation Phase Complete When:

- ☐ Security configurations (Baseline and Hardened) are defined, tested, and documented
- ☐ Pilot repositories are assigned to appropriate security configuration tier
- ☐ Security overview dashboards provide usable adoption and backlog views
- ☐ Ownership model is documented with verified ownership for all pilot repositories
- ☐ Governance decisions are documented and approved
- ☐ Baseline metrics are captured and documented
- ☐ Weekly metrics report template is created and tested

7 Priority 1: Secret Scanning

Official Documentation Portal

Primary Reference: https://docs.github.com/en/code-security/secret-scanning
Secret scanning is available as part of GitHub Secret Protection. It detects secrets that have been checked into repositories and can block pushes containing secrets with push protection.

7.1 Strategic Rationale

Secret scanning addresses one of the highest-impact security risks: credential exposure in source code. Leaked credentials in public or semi-public repositories have been the root cause of numerous high-profile breaches.

Why Secret Scanning First

- **Immediate Risk Reduction**: Exposed credentials can be exploited within minutes of commit
- **Clear Remediation Path**: Rotate the credential, remove from history, done
- **Low False Positive Rate**: Pattern matching for known credential formats is highly accurate
- **Quick Wins**: Fast deployment, immediate value demonstration
- **Workflow Validation**: Tests alert routing and remediation workflows at manageable volume

7.2 Documentation Reading Sequence

1. **Secret Scanning Overview**
   - Secret Scanning Portal
   - Understanding the detection model
   - Push scanning vs. historical scanning
   - Partner program and validity checks
2. **Enabling Secret Scanning**
   - Enabling Secret Scanning Features
   - Repository vs. organization enablement
   - Default settings for new repositories
   - Enterprise-level configuration

3. **Push Protection**
   - About Push Protection
   - Enabling Push Protection for Your Repository
   - Working with Push Protection from the Command Line
   - Push Protection for Users
   - Preventive vs. detective controls
   - Bypass mechanisms and audit trails
   - Developer experience considerations
4. **Supported Secret Types**
   - Built-in patterns (100+ providers)
   - High-confidence vs. low-confidence patterns
   - Pattern matching algorithms
5. **Validity Checks**
   - How validity is determined
   - Supported providers
   - Using validity in triage
6. **Custom Patterns**
   - Pattern syntax (Hyperscan regex)
   - Testing and validation
   - Organization-wide deployment
7. **Alert Management**
   - Working with Secret Scanning and Push Protection
   - Alert lifecycle
   - Dismissal reasons and documentation
   - Bulk operations
8. **Remediation at Scale**
   - Credential rotation procedures
   - Git history considerations
   - Partner notification programs

7.3 Implementation Plan

7.3.1 Phase 2a: Pilot (5–10 Repositories)

Duration: 3–5 days

Activities:

1. Enable secret scanning on pilot repositories
2. Enable push protection (may initially be in "alert only" mode)
3. Review and triage any existing alerts
4. Validate alert routing and notification
5. Test remediation workflow end-to-end
6. Document learnings and adjust configuration

Validation Criteria:

- Alerts appear in Security Overview within expected timeframe
- Notifications reach designated owners
- At least one test secret is successfully detected and remediated
- Push protection correctly blocks a test commit (and bypass works as expected)

7.3.2 Phase 2b: Scale (All Repositories)

Duration: 5–7 days

Activities:

1. Apply Baseline security configuration to all repositories
2. Apply Hardened security configuration to Tier 0/1 repositories
3. Communicate changes to all development teams (see Communication Template)
4. Monitor alert volume and adjust as needed
5. Enable push protection organization-wide (if maturity allows)

Communication Template:

```
Subject: [Action Required] Secret Scanning Enabled for Your
   Repositories

Hi Team,

We have enabled GitHub Secret Scanning on your repositories as part of
our security improvement program. Here's what you need to know:

WHAT'S CHANGING:
- Secret scanning will detect committed credentials and API keys
- [For Tier 0/1] Push protection will block commits containing secrets

WHAT YOU NEED TO DO:
- Review any existing alerts in your repository's Security tab
- Follow the remediation runbook [link] for any detected secrets
- Contact #appsec-support with questions

RESOURCES:
- Remediation Runbook: [link]
- FAQ: [link]
- Support: #appsec-support

Questions? Reply to this message or reach out in #appsec-support.

Thanks,
AppSec Team
```

7.3.3 Phase 2c: Custom Patterns

Duration: 3–5 days (iterative)

Common Custom Pattern Categories:

- Internal API keys and tokens
- Database connection strings
- Internal service account credentials
- Encryption keys and certificates
- Environment-specific secrets

Pattern Development Process:

1. Inventory internal secret formats with engineering teams
2. Develop regex patterns using Hyperscan syntax
3. Test patterns in dry-run mode (if available) or test organization
4. Deploy to organization with monitoring
5. Tune based on false positive feedback

Custom Pattern Best Practices

- **Be specific**: Overly broad patterns generate false positives
- **Include anchors**: Use word boundaries and context markers
- **Test extensively**: Validate against real repository content
- **Document rationale**: Record why each pattern exists
- **Review periodically**: Remove obsolete patterns

7.4 Remediation Runbook

7.4.1 Standard Response Procedure

1. **Triage** (Target: <1 hour for valid secrets)
   - Confirm secret type, scope, and exposure level
   - Check validity status (active, inactive, unknown)
   - Determine if secret has been accessed externally (if possible)
   - Assess blast radius: what systems/data could be compromised

2. **Contain** (Target: <4 hours for critical credentials)
   - Revoke or rotate the credential immediately
   - If third-party service: follow provider's breach response procedure
   - If internal service: coordinate with service owner
   - Document rotation in ticket/issue
3. **Eradicate** (Target: <24 hours)
   - Remove secret from current codebase
   - Assess need for Git history rewrite (policy-dependent)
   - If history rewrite required: coordinate with repository owner
   - Update any dependent configurations
4. **Recover** (Target: <48 hours)
   - Deploy updated configuration with new credential
   - Verify application functionality
   - Monitor for unauthorized access attempts (if detection is possible)
5. **Prevent Recurrence** (Target: <1 week)
   - Analyze root cause: how did the secret enter the codebase?
   - Implement preventive measures (secret management, developer training)
   - Update custom patterns if needed
   - Document lessons learned

7.4.2 Severity Classification

Severity	Secret Type	Examples	SLA
Critical	Production credentials with broad access	AWS root keys, database admin passwords, API keys with admin scope	4 hours
High	Production credentials with limited scope	Service-specific API keys, read-only database credentials	24 hours
Medium	Non-production or expired credentials	Test environment keys, expired tokens	7 days
Low	False positives or explicitly allowed	Example credentials, test fixtures	30 days

7.5 Push Protection Configuration

7.5.1 Bypass Policy Framework

Bypass Reason	When Appropriate	Approval
False positive	Pattern matches non-sensitive content	Self (with justification)
Used in tests	Credential is intentionally for testing	Self (with justification)
Will fix later	Temporary bypass for urgent deployment	Manager + AppSec

7.5.2 Bypass Monitoring

Establish weekly review of bypass events:

- Identify patterns in bypass reasons
- Follow up on "will fix later" bypasses
- Tune patterns to reduce false positive bypasses
- Escalate concerning bypass patterns to management

7.6 Integration with Incident Response

Condition	Action	Responsible
Valid credential with external exposure	Trigger incident response process	Security Operations
AWS/GCP/Azure root credential	Immediate escalation to Cloud team	AppSec + Cloud
Database credentials for production data	Notify Data Protection team	AppSec + Data
Evidence of credential use	Full incident investigation	Security Operations

7.7 Definition of Done: Secret Scanning

Secret Scanning Phase Complete When:

- ☐ Secret scanning enabled for 100% of in-scope repositories
- ☐ Push protection enabled for Tier 0/1 repositories (minimum)
- ☐ Custom patterns deployed for all identified internal secret formats
- ☐ Remediation runbook documented, tested, and communicated
- ☐ Alert routing verified for all repository ownership groups
- ☐ Bypass policy documented and communicated
- ☐ Weekly bypass review process established
- ☐ Integration with incident response process documented
- ☐ Metrics baseline captured (alert volume, remediation time)

8 Priority 2: Code Scanning (CodeQL)

Official Documentation Portal

Primary Reference: https://docs.github.com/en/code-security/code-scanning
CodeQL Reference: https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning-with-codeql
CodeQL is GitHub's semantic code analysis engine. It treats code as data, enabling sophisticated vulnerability detection.

8.1 Strategic Rationale

Code scanning with CodeQL provides automated static analysis to identify security vulnerabilities during development. Unlike secret scanning (which detects known patterns), code scanning uses semantic analysis to find complex vulnerability patterns like SQL injection, XSS, and insecure deserialization.

Why Code Scanning Second

- **Higher Alert Volume**: Code scanning generates more alerts than secret scanning, requiring mature triage workflows
- **Complexity**: False positives require security expertise to evaluate
- **Build Integration**: Advanced setup requires CI/CD pipeline modifications
- **Developer Education**: Teams need to understand vulnerability categories
- **Foundation Required**: Benefits from established ownership and governance

8.2 Documentation Reading Sequence

1. **About Code Scanning**
   - About Code Scanning with CodeQL
   - Detection model and supported languages
   - Alert lifecycle and severity model
   - CodeQL vs. third-party tools
2. **Default Setup**
   - Configuring Default Setup for Code Scanning
   - Automatic language detection
   - Configuration options and limitations
   - When default setup is sufficient

3. **Advanced Setup**
   - Creating an Advanced Setup for Code Scanning
   - CodeQL Code Scanning for Compiled Languages
   - Workflow configuration options
   - Build process integration
   - Query suite selection
   - Scheduling and triggers
4. **Alert Operations**
   - Interpreting alert context
   - Triage and investigation
   - Resolution states and transitions
   - Bulk operations
5. **CodeQL Specifics**
   - CodeQL Query Suites
   - Query suites (default, security-extended, security-and-quality)
   - Query customization and creation
   - CodeQL packs and bundles
   - Performance optimization
6. **SARIF Import**
   - Integrating third-party tool results
   - SARIF format requirements
   - Deduplication and tracking
7. **Merge Protection**
   - Ruleset configuration
   - Severity thresholds
   - Override policies

8.3 Setup Types Comparison

Aspect	Default Setup	Advanced Setup
Configuration	Automatic; no workflow files	Requires workflow YAML
Build process	Uses inferred build commands	Custom build configuration
Query suite	Default or extended (configurable)	Fully customizable
Scheduling	Automatic on push/PR + weekly	Custom triggers
Multi-language	Automatic detection	Explicit configuration
Best for	Standard projects, broad coverage	Complex builds, custom queries

8.4 Implementation Plan

8.4.1 Phase 3a: Broad Enablement with Default Setup

Duration: 5–7 days

Activities:

1. Enable default setup across all repositories via security configuration
2. Monitor initial scan completion and language detection accuracy
3. Review and triage initial alert volume
4. Establish ownership assignment for new alerts
5. Document common false positive patterns

Expected Outcomes:

- 80%+ of repositories successfully scanning within 48 hours
- Initial alert backlog quantified and assigned to owners
- Common triage patterns identified

8.4.2 Phase 3b: Advanced Setup for Crown Jewels

Duration: 5–10 days
Target Repositories: All Tier 0 and critical Tier 1 repositories
Activities:

1. Assess each repository's build complexity and language mix
2. Create customized CodeQL workflow files
3. Configure appropriate query suites (security-extended recommended)
4. Set up scheduled scans aligned with release cadence
5. Validate scan results and tune as needed

Sample Advanced Workflow:

```yaml
name: "CodeQL Advanced Analysis"

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
  schedule:
    - cron: '0 6 * * 1'  # Weekly Monday 6am UTC

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

    strategy:
      fail-fast: false
      matrix:
        language: ['javascript', 'python']

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Initialize CodeQL
        uses: github/codeql-action/init@v3
        with:
          languages: ${{ matrix.language }}
          queries: security-extended

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

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

8.4.3 Phase 3c: Query Suite Selection

Suite	Coverage	Recommendation
`default`	Core security vulnerabilities with high precision	Baseline tier repos
`security-extended`	Additional medium-precision security queries	Hardened tier repos
`security-and-quality`	Security + code quality issues	Development/QA focus

8.5 Alert Triage Framework

8.5.1 Severity Definitions and SLAs

Severity	Description	Tier 0 SLA	Tier 1 SLA	Tier 2 SLA
Critical	Remote code execution, authentication bypass	7 days	14 days	30 days
High	SQL injection, XSS, SSRF	14 days	30 days	60 days
Medium	Information disclosure, weak crypto	30 days	60 days	90 days
Low	Best practice violations	90 days	120 days	Backlog

8.5.2 Triage Decision Tree

1. **Is the finding reachable?**
   - Review dataflow paths shown in alert
   - Check if vulnerable code path is exercised in production
   - Consider input validation and sanitization elsewhere
2. **Is exploitation feasible?**
   - Assess attack prerequisites (authentication, network access)
   - Evaluate compensating controls (WAF, rate limiting)
   - Consider actual vs. theoretical risk
3. **What is the impact?**
   - Data sensitivity of affected component
   - Blast radius of successful exploitation
   - Regulatory or compliance implications
4. **Resolution Options**:
   - **Fix**: Remediate the vulnerability in code
   - **Dismiss (False Positive)**: Detection is incorrect
   - **Dismiss (Won't Fix)**: Accepted risk with compensating controls
   - **Dismiss (Test Code)**: Vulnerability is in test fixtures only

8.5.3 Dismissal Governance

Dismissal Reason	Required Documentation	Approver
False positive	Technical explanation of why detection is wrong	Developer + Security Review
Won't fix	Risk assessment, compensating controls	Security Team + Risk Owner
Used in tests	Confirmation that code is test-only	Developer

8.6 Merge Protection Configuration

8.6.1 Phased Enforcement Rollout

1. **Phase 1: Observation** (Weeks 1–2)
   - Enable code scanning without merge protection
   - Measure alert volume and remediation capacity
   - Identify and address common false positives
2. **Phase 2: Warning** (Weeks 3–4)
   - Enable status checks that warn but don't block
   - Track alert acknowledgment rates
   - Refine severity thresholds based on feedback
3. **Phase 3: Soft Block** (Weeks 5–6)
   - Block merges on Critical findings (with bypass allowed)
   - Monitor bypass usage and reasons
   - Adjust thresholds as needed
4. **Phase 4: Hard Enforcement** (Week 7+)
   - Block merges on Critical and High findings
   - Require security review for bypasses on Critical
   - Establish SLA tracking for blocked PRs

8.6.2 Recommended Ruleset Configuration

```yaml
# Organization Ruleset: Code Scanning - Hardened Tier
name: "Code Scanning Enforcement"
target: branch
enforcement: active

conditions:
  ref_name:
    include:
      - refs/heads/main
      - refs/heads/release/*

rules:
  - type: code_scanning
    parameters:
      code_scanning_tool: codeql
      security_alerts_threshold: high_or_higher
      alerts_threshold: errors
```

8.7 Definition of Done: Code Scanning

Code Scanning Phase Complete When:

- ☐ Default setup enabled on 90%+ of in-scope repositories
- ☐ Advanced setup configured for all Tier 0 repositories
- ☐ Appropriate query suites selected and documented
- ☐ Alert triage workflow documented and operational
- ☐ Severity SLAs defined and communicated
- ☐ Dismissal governance process implemented
- ☐ Merge protection enabled for Tier 0 repositories (minimum)
- ☐ Bypass monitoring and review process established
- ☐ Initial alert backlog triaged and assigned

9 Priority 3: Dependabot (Supply Chain Security)

Official Documentation Portal

Primary Reference: https://docs.github.com/en/code-security/dependabot
Alerts: https://docs.github.com/code-security/dependabot/dependabot-alerts/about-dependabot-alerts
Security Updates: https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/about-dependabot-security-updates
Dependabot automatically keeps your dependencies up to date and alerts you to security vulnerabilities.

9.1 Strategic Rationale

Supply chain attacks have emerged as a critical threat vector. Dependabot addresses this by providing visibility into dependencies and automating vulnerability remediation through pull requests.

Why Dependabot Third

- **Volume Management**: Dependabot can generate significant PR volume
- **Team Readiness**: Requires established PR review capacity
- **Policy Maturity**: Benefits from established security policies
- **Complementary Coverage**: Addresses different risk vector than code/secrets
- **Automation Focus**: More automated than other capabilities

9.2 Documentation Reading Sequence

1. **Dependabot Quickstart**
   - Dependabot Documentation Portal
   - Feature overview and capabilities
   - Supported ecosystems
   - Getting started workflow
2. **Dependency Graph**
   - About the Dependency Graph
   - Configuring the Dependency Graph
   - How dependencies are detected
   - Supported manifest files
   - Accuracy considerations

3. **Dependabot Alerts**
   - About Dependabot Alerts
   - Viewing and Updating Dependabot Alerts
   - Configuring Dependabot Alerts
   - GitHub Advisory Database
   - Alert severity and scoring
   - Alert management
4. **Dependabot Security Updates**
   - About Dependabot Security Updates
   - Configuring Dependabot Security Updates
   - Automated PR generation
   - Update strategies
   - Configuration options
5. **Dependabot Version Updates**
   - Keeping dependencies current
   - Scheduling and grouping
   - Managing PR volume
6. **Dependency Review**
   - About Dependency Review
   - Customizing Dependency Review Action Configuration
   - PR-time dependency scanning
   - Blocking vulnerable dependencies
   - License compliance

9.3 Feature Comparison

Feature	Purpose	Output
Dependency Graph	Visibility into project dependencies	Inventory view
Dependabot Alerts	Notification of vulnerable dependencies	Security alerts
Security Updates	Automated PRs to fix vulnerable dependencies	Pull requests
Version Updates	Automated PRs to keep dependencies current	Pull requests
Dependency Review	Block introduction of vulnerable dependencies	PR check

9.4 Implementation Plan

9.4.1 Phase 4a: Visibility Foundation

Duration: 3–5 days

Activities:

1. Verify dependency graph is enabled and accurate across all repositories
2. Review detected ecosystems and languages
3. Identify repositories with incomplete or missing dependency detection
4. Enable Dependabot alerts organization-wide
5. Quantify initial vulnerability backlog

Dependency Graph Accuracy Checks:

- Verify manifest files are in standard locations
- Check for lockfile presence and accuracy
- Identify vendored dependencies that may not be detected
- Review private registry configurations

9.4.2 Phase 4b: Automated Remediation

Duration: 5–7 days

Activities:

1. Enable security updates on prioritized repositories
2. Establish PR triage norms and ownership
3. Configure auto-merge policies where appropriate
4. Monitor PR volume and team capacity
5. Tune settings to manage noise

Auto-Merge Considerations:

Criteria	Auto-Merge Candidate	Manual Review Required
Update Type	Patch version	Major/minor version
Test Coverage	High coverage, all tests pass	Low coverage or test failures
Dependency Type	Development dependencies	Production dependencies
Repository Tier	Tier 2	Tier 0/1
Changelog	No breaking changes noted	Breaking changes indicated

9.4.3 Phase 4c: Version Currency

Duration: Ongoing

Activities:

1. Enable version updates selectively on mature repositories
2. Configure scheduling to manage PR timing
3. Implement grouping to reduce PR volume
4. Establish version update policies by ecosystem

Sample Dependabot Configuration:

```yaml
# .github/dependabot.yml
version: 2
updates:
  # npm dependencies
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
      day: "monday"
      time: "06:00"
      timezone: "America/New_York"
    groups:
      development-dependencies:
        dependency-type: "development"
        patterns:
          - "*"
      production-dependencies:
        dependency-type: "production"
        patterns:
          - "*"
    open-pull-requests-limit: 10
    reviewers:
      - "org/platform-team"
    labels:
      - "dependencies"
      - "automated"

  # GitHub Actions
  - package-ecosystem: "github-actions"
    directory: "/"
    schedule:
      interval: "weekly"
    groups:
      actions:
        patterns:
          - "*"
```

9.5 Alert Triage and Prioritization

9.5.1 Severity Assessment

CVSS Score	GitHub Severity	Remediation SLA
9.0–10.0	Critical	7 days
7.0–8.9	High	30 days
4.0–6.9	Medium	60 days
0.1–3.9	Low	90 days

9.5.2 Exploitability Assessment

When prioritizing remediation, consider:

- **EPSS Score**: Probability of exploitation in the wild
- **Known Exploits**: Check CISA KEV catalog
- **Reachability**: Is the vulnerable function actually used?
- **Environment**: Is the dependency used in production or development only?
- **Network Exposure**: Can the vulnerability be exploited remotely?

9.6 PR Management Strategies

9.6.1 Reducing PR Volume

- **Grouping**: Combine related updates into single PRs
- **Scheduling**: Align updates with team capacity (e.g., Monday mornings)
- **Limits**: Set maximum open PRs per repository
- **Ignoring**: Exclude dependencies with known noisy update patterns
- **Allow Lists**: Only enable for specific dependencies

9.6.2 PR Review Workflow

1. **Automated Checks**:
   - CI pipeline passes
   - No security regressions (dependency review)
   - Changelog reviewed for breaking changes
2. **Human Review**:
   - Major version updates
   - Production dependencies
   - Repositories with low test coverage

3. **Merge Decision**:

- Auto-merge if all criteria met
- Defer if team lacks capacity
- Close if update causes issues

9.7 Dependency Review (PR Gate)

9.7.1 Configuration

```yaml
# .github/workflows/dependency-review.yml
name: 'Dependency Review'
on:
  pull_request:
    branches: [main]

jobs:
  dependency-review:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Dependency Review
        uses: actions/dependency-review-action@v4
        with:
          fail-on-severity: high
          deny-licenses: GPL-3.0, AGPL-3.0
          allow-ghsas: GHSA-xxxx-yyyy  # Known accepted risks
```

9.7.2 Policy Decisions

Document policies for:

- Severity threshold for blocking PRs
- License compatibility requirements
- Exception process for necessary but vulnerable dependencies

9.8 Definition of Done: Dependabot

Dependabot Phase Complete When:

- ☐ Dependency graph enabled and accurate for all in-scope repositories
- ☐ Dependabot alerts enabled organization-wide
- ☐ Security updates enabled for prioritized repositories
- ☐ Version updates configured with appropriate grouping and scheduling
- ☐ PR management policies documented and communicated
- ☐ Auto-merge policies defined and implemented where appropriate
- ☐ Dependency review enabled for Tier 0/1 repositories
- ☐ Alert SLAs defined and tracking established
- ☐ Initial vulnerability backlog quantified and prioritized

10 Governance, Reporting, and Evidence

Official Documentation References

- **Security Overview**: https://docs.github.com/en/code-security/security-overview/about-security-overview
- **Viewing Insights**: https://docs.github.com/en/code-security/security-overview/viewing-security-insights
- **Assessing Risk**: https://docs.github.com/en/code-security/security-overview/assessing-code-security-risk
- **Auditing Alerts**: https://docs.github.com/en/code-security/getting-started/auditing-security-alerts

10.1 Operational Cadence

10.1.1 Weekly Activities

Activity	Description	Owner	Output
Metrics collection	Export coverage and alert data	AppSec	Dashboard update
Alert review	Triage new alerts, update aging report	AppSec	Triage report
Bypass review	Review push protection and merge bypasses	AppSec	Exception log
Team sync	Coordinate with development teams on blockers	AppSec	Action items

10.1.2 Monthly Activities

Activity	Description	Owner	Output
Policy review	Assess policy effectiveness, propose changes	AppSec Lead	Policy updates
Trend analysis	Identify patterns in alerts and remediation	AppSec	Trend report
Leadership report	Executive summary of program health	AppSec Lead	Report
False positive review	Analyze dismissed alerts, tune detection	AppSec	Tuning changes

10.1.3 Quarterly Activities

Activity	Description	Owner	Output
Program assessment	Comprehensive health check against goals	AppSec Lead	Assessment doc
Roadmap refresh	Update priorities based on learnings	AppSec Lead	Updated roadmap
Developer survey	Gather feedback on security tooling	AppSec	Survey results
Training review	Assess team capability and training needs	AppSec Lead	Training plan

10.2 Metrics Framework

10.2.1 Coverage Metrics

Metric	Definition	Target
Secret scanning coverage	% repos with secret scanning enabled	100%
Push protection coverage	% repos with push protection enabled	100% Tier 0/1
Code scanning coverage	% repos with code scanning enabled	90%+
Advanced setup coverage	% Tier 0 repos with advanced CodeQL setup	100%
Dependabot alerts coverage	% repos with Dependabot alerts enabled	100%
Security updates coverage	% priority repos with security updates enabled	80%+

10.2.2 Backlog Metrics

Metric	Definition	Target
Open critical alerts	Count of unresolved critical alerts	<5
Open high alerts	Count of unresolved high alerts	<25
Aging alerts	Alerts older than SLA by severity	0
Alert backlog trend	Week-over-week change in backlog	Decreasing

10.2.3 Throughput Metrics

Metric	Definition	Target
Alerts resolved per week	Count of alerts closed weekly	Baseline + 10%
MTTR (Critical)	Mean time to remediate critical alerts	<7 days
MTTR (High)	Mean time to remediate high alerts	<30 days
Time to triage	Average time from alert to first action	<24 hours

10.2.4 Prevention Metrics

Metric	Definition	Target
Pushes blocked	Count of secret push protection blocks	Track trend
Secrets prevented	Unique secrets blocked before commit	Track trend
Vulnerable deps blocked	Dependencies blocked by dependency review	Track trend
PR blocks	Merges blocked by code scanning policy	Track trend

10.2.5 Quality Metrics

Metric	Definition	Target
False positive rate	% alerts dismissed as false positive	<10%
Bypass rate	% commits bypassing push protection	<5%
Reopen rate	% alerts reopened after closure	<2%
Developer satisfaction	Survey score on security tooling	>3.5/5

10.3 Reporting Templates

10.3.1 Weekly Metrics Report

```
GHAS Weekly Metrics Report
Week of: [DATE]

COVERAGE
- Secret Scanning: XX% (target: 100%)
- Push Protection: XX% Tier 0/1 (target: 100%)
- Code Scanning: XX% (target: 90%)
- Dependabot Alerts: XX% (target: 100%)

BACKLOG
- Critical: X (SLA: 0 overdue)
- High: X (SLA: X overdue)
- Medium: X
- Low: X
- Week-over-week change: +/-X

THROUGHPUT
- Alerts resolved this week: X
- MTTR (Critical): X days
- MTTR (High): X days

PREVENTION
- Pushes blocked: X
- Secrets prevented: X
- Vulnerable deps blocked: X

ACTIONS REQUIRED
1. [Action item]
2. [Action item]
```

10.3.2 Monthly Executive Report

```
GHAS Program Monthly Report
Month: [MONTH YEAR]

EXECUTIVE SUMMARY
[2-3 sentence summary of program health and key accomplishments]

KEY METRICS
| Metric                | Previous | Current | Target | Status |
|-----------------------|----------|---------|--------|--------|
| Coverage (overall)    | X%       | X%      | 95%    | [RAG]  |
| Critical backlog      | X        | X       | <5     | [RAG]  |
| MTTR (Critical)       | X days   | X days  | <7     | [RAG]  |
| Prevention events     | X        | X       | -      | -      |

ACCOMPLISHMENTS
- [Key accomplishment 1]
- [Key accomplishment 2]

CHALLENGES
- [Challenge and mitigation]

NEXT MONTH PRIORITIES
1. [Priority 1]
2. [Priority 2]
```

10.4 Evidence Artifacts

Maintain the following artifacts for audit and compliance purposes:

Artifact	Description	Retention
Security configuration exports	JSON/YAML of Baseline and Hardened configs	Permanent
Weekly metrics snapshots	Exported Security Overview data	2 years
Remediation runbooks	Current versions of all response procedures	Permanent
Policy documents	Approved SLAs, governance decisions, exception policies	Permanent
Incident postmortems	Root cause analysis for security incidents	7 years
Training records	Certification and training completion evidence	3 years
Exception log	Approved exceptions with justification	3 years
Audit reports	Third-party or internal audit findings	7 years

11 Automation and API Integration

11.1 GitHub API Overview

GHAS capabilities can be automated and extended through GitHub's REST and GraphQL APIs. This section provides examples for common automation scenarios.

11.2 Authentication Setup

```
# Create a Personal Access Token (PAT) with these scopes:
# - repo (full control of private repositories)
# - security_events (read and write security events)
# - admin:org (read and write organization settings)

# Store token securely
export GITHUB_TOKEN="ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
export GITHUB_ORG="your-organization"
```

11.3 Common Automation Scripts

11.3.1 Enable Secret Scanning Organization-Wide

```
#!/usr/bin/env python3
"""Enable secret scanning for all repositories in an organization."""

import requests
import os

GITHUB_TOKEN = os.environ["GITHUB_TOKEN"]
GITHUB_ORG = os.environ["GITHUB_ORG"]
BASE_URL = "https://api.github.com"

headers = {
    "Authorization": f"Bearer {GITHUB_TOKEN}",
    "Accept": "application/vnd.github+json",
    "X-GitHub-Api-Version": "2022-11-28"
}

def get_repos(org):
    """Fetch all repositories in the organization."""
    repos = []
    page = 1
    while True:
        response = requests.get(
            f"{BASE_URL}/orgs/{org}/repos",
            headers=headers,
            params={"per_page": 100, "page": page}
        )
        response.raise_for_status()
        data = response.json()
        if not data:
            break
        repos.extend(data)
        page += 1
    return repos

def enable_secret_scanning(org, repo_name):
    """Enable secret scanning for a repository."""
    response = requests.patch(
        f"{BASE_URL}/repos/{org}/{repo_name}",
        headers=headers,
        json={
            "security_and_analysis": {
                "secret_scanning": {"status": "enabled"},
                "secret_scanning_push_protection": {"status":
                    "enabled"}
            }
        }
    )
    return response.status_code == 200

if __name__ == "__main__":
```

```
    repos = get_repos(GITHUB_ORG)
    for repo in repos:
        if not repo["archived"]:
            success = enable_secret_scanning(GITHUB_ORG, repo["name"])
            status = "enabled" if success else "FAILED"
            print(f"{repo['name']}: {status}")
```

11.3.2 Export Security Alerts

```
#!/usr/bin/env python3
"""Export all security alerts to CSV for reporting."""

import requests
import csv
import os
from datetime import datetime

GITHUB_TOKEN = os.environ["GITHUB_TOKEN"]
GITHUB_ORG = os.environ["GITHUB_ORG"]
BASE_URL = "https://api.github.com"

headers = {
    "Authorization": f"Bearer {GITHUB_TOKEN}",
    "Accept": "application/vnd.github+json",
    "X-GitHub-Api-Version": "2022-11-28"
}

def get_code_scanning_alerts(org, repo):
    """Fetch code scanning alerts for a repository."""
    alerts = []
    page = 1
    while True:
        response = requests.get(
            f"{BASE_URL}/repos/{org}/{repo}/code-scanning/alerts",
            headers=headers,
            params={"per_page": 100, "page": page, "state": "open"}
        )
        if response.status_code == 404:
            break  # No alerts or feature not enabled
        response.raise_for_status()
        data = response.json()
        if not data:
            break
        alerts.extend(data)
        page += 1
    return alerts

def export_alerts_to_csv(org, output_file):
    """Export all alerts to CSV."""
    repos = get_repos(org)  # Reuse function from previous example

    with open(output_file, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow([
            "Repository", "Alert Number", "Rule ID", "Severity",
            "State", "Created At", "URL"
        ])

        for repo in repos:
            alerts = get_code_scanning_alerts(org, repo["name"])
            for alert in alerts:
```

```
                writer.writerow([
                    repo["name"],
                    alert["number"],
                    alert["rule"]["id"],
                    alert["rule"]["security_severity_level"],
                    alert["state"],
                    alert["created_at"],
                    alert["html_url"]
                ])

    print(f"Exported alerts to {output_file}")
```

11.3.3 Bulk Apply Security Configuration

```
#!/bin/bash
# Apply security configuration to repositories by tier

# Tier 0 repositories (crown jewels)
TIER0_REPOS=(
    "auth-service"
    "payment-gateway"
    "user-data-service"
)

# Apply Hardened configuration
for repo in "${TIER0_REPOS[@]}"; do
    echo "Applying Hardened configuration to $repo..."
    gh api \
        --method PUT \
        "/repos/$GITHUB_ORG/$repo/security-configurations" \
        -f configuration_id="hardened-config-id"
done
```

11.4 GitHub Actions for Automated Governance

11.4.1 Alert Aging Notification

```yaml
# .github/workflows/alert-aging-check.yml
name: Check for Aging Security Alerts

on:
  schedule:
    - cron: '0 9 * * 1'  # Every Monday at 9am

jobs:
  check-alerts:
    runs-on: ubuntu-latest
    steps:
      - name: Check for aging critical alerts
        uses: actions/github-script@v7
        with:
          github-token: ${{ secrets.SECURITY_TOKEN }}
          script: |
            const { owner, repo } = context.repo;
            const alerts = await
                github.rest.codeScanning.listAlertsForRepo({
              owner,
              repo,
              state: 'open'
            });

            const criticalAging = alerts.data.filter(alert => {
              const age = Date.now() - new Date(alert.created_at);
              const days = age / (1000 * 60 * 60 * 24);
              return alert.rule.security_severity_level === 'critical'
                     && days > 7;
            });

            if (criticalAging.length > 0) {
              // Send notification (Slack, email, etc.)
              console.log(`Found ${criticalAging.length} aging
                 critical alerts`);
            }
```

11.5 Webhook Integration

Configure webhooks to receive real-time notifications for security events:

Event	Use Case	Target
`secret_scanning_alert`	Trigger incident response for valid secrets	SIEM/SOAR
`code_scanning_alert`	Update security dashboard	Metrics system
`dependabot_alert`	Create tracking ticket	Issue tracker
`repository_vulnerability_alert`	Notify security team	Slack/Teams

12 Troubleshooting Guide

12.1 Secret Scanning Issues

Issue	Possible Causes	Resolution
Alerts not appearing	Feature not enabled; repository archived; secret type not supported	Verify enablement in settings; check supported patterns
Push protection not blocking	Feature not enabled; secret not in supported patterns; bypass in effect	Check org settings; review bypass audit log
High false positive rate	Custom patterns too broad; test data triggering alerts	Refine pattern regex; use allow lists
Delayed alert appearance	Large repository initial scan; API rate limiting	Wait for scan completion; check scan status

12.2 Code Scanning Issues

Issue	Possible Causes	Resolution
Default setup failing	Language not supported; build issues; resource limits	Check supported languages; review workflow logs
No alerts generated	No vulnerabilities found; queries not matching; build incomplete	Verify build captured all code; check query suite
Analysis timeout	Large codebase; resource constraints; complex dependencies	Increase timeout; optimize build; use Advanced setup
Missing languages	Auto-detection incomplete; polyglot repository	Use Advanced setup with explicit language matrix

12.3 Dependabot Issues

Issue	Possible Causes	Resolution
Incomplete dependency graph	Manifest not detected; private registry auth issues	Verify manifest location; configure registry access
PRs not created	Feature disabled; dependency not in vulnerability database	Enable security updates; check ecosystem support
Too many PRs	Version updates on all deps; no grouping configured	Add grouping; limit PR count; adjust schedule
PR merge failures	CI failures; merge conflicts; stale branches	Fix tests; rebase PRs; configure auto-rebase

12.4 Performance Optimization

12.4.1 Code Scanning Performance

- **Build Caching**: Use GitHub Actions cache for dependencies
- **Incremental Analysis**: Configure CodeQL to analyze only changed files
- **Resource Scaling**: Use larger runners for complex repositories
- **Query Selection**: Use default suite instead of security-and-quality for faster scans

12.4.2 Reducing Alert Noise

- **False Positive Suppression**: Use inline comments or configuration files
- **Path Exclusions**: Exclude vendor, test, and generated directories
- **Severity Filtering**: Focus on high and critical alerts first
- **Query Tuning**: Remove or adjust queries with high false positive rates

13 Exam Readiness Checklist

13.1 Hands-On Competency Requirements

Demonstrate each capability in a test organization before attempting certification:

Domain	Competency	Verified
Secret Scanning	Enable and manage secret scanning at repository and org level	☐
	Configure push protection with appropriate bypass policies	☐
	Create and tune custom patterns for internal secret formats	☐
	Interpret alert context and execute remediation workflow	☐
Code Scanning	Enable default setup and verify successful analysis	☐
	Configure advanced setup with custom workflow	☐
	Select and justify query suite choices	☐
	Triage alerts using dataflow analysis	☐
	Configure merge protection with severity thresholds	☐
Dependabot	Enable and verify dependency graph accuracy	☐
	Configure Dependabot alerts and security updates	☐
	Set up version updates with grouping and scheduling	☐
	Configure dependency review in PR workflow	☐
Governance	Create and apply security configurations	☐
	Configure dismissal policies and governance controls	☐
	Generate and interpret security metrics reports	☐

13.2 Recommended Preparation Path

1. **Baseline Assessment**: Take official practice exam to identify knowledge gaps
2. **Hands-On Implementation**: Complete this roadmap in a test organization, focusing on weak areas identified in assessment
3. **Documentation Review**: Re-read official documentation for each capability area
4. **Validation**: Retake practice assessment and ensure all areas meet passing threshold
5. **Final Review**: Review edge cases, advanced configurations, and troubleshooting scenarios

13.3 Key Exam Topics

- Security configuration design and application patterns
- Push protection bypass policies and audit mechanisms
- CodeQL query suite selection criteria
- SARIF format and third-party tool integration
- Merge protection ruleset configuration
- Dependabot grouping and scheduling strategies
- Security overview dashboard interpretation
- Alert lifecycle and dismissal governance

A Implementation Backlog Template

Use this structure to convert the roadmap into your internal backlog:

Epic	Capability	Definition of Done	Evidence
Foundation	Security configurations defined	Baseline and Hardened configurations created, tested, documented; applied to pilot repos	Configuration exports; test results
Foundation	Ownership model documented	All repos have verified owners; escalation paths documented	Ownership matrix; CODEOWNERS audit
Secret Scanning	Org-wide enablement	100% coverage; push protection on Tier 0/1	Security overview export; coverage metrics
Secret Scanning	Custom patterns deployed	All internal token formats covered; <10% false positive rate	Pattern definitions; FP metrics
Secret Scanning	Remediation workflow	Runbook documented, tested; incident integration complete	Runbook document; test records
Code Scanning	Default setup coverage	90%+ repos with default setup enabled	Security overview export
Code Scanning	Advanced setup for crown jewels	All Tier 0 repos with advanced CodeQL; security-extended queries	Workflow files; scan results
Code Scanning	Merge protection	Tier 0 repos blocking on critical/high findings	Ruleset configuration; bypass logs
Dependabot	Visibility foundation	Dependency graph accurate; alerts enabled org-wide	Dependency review; alert coverage
Dependabot	Automated remediation	Security updates on priority repos; PR policies documented	dependabot.yml files; PR metrics
Program Ops	Metrics and reporting	Weekly reports operational; leadership reporting established	Report templates; distribution list

B Repository Tiering Model

B.1 Tier Definitions

Tier	Name	Criteria
Tier 0	Crown Jewels	Identity/authentication services; payment processing; key infrastructure; PII/PHI data services; core security services
Tier 1	Critical	Customer-facing applications; core business logic; revenue-generating services; regulatory compliance systems
Tier 2	Standard	Internal tools; low-risk services; prototypes; documentation repositories

B.2 Control Requirements by Tier

Control	Tier 0	Tier 1	Tier 2
Secret Scanning	Required	Required	Required
Push Protection	Required	Required	Recommended
Custom Patterns	Required	Required	Optional
Code Scanning (Default)	Required	Required	Recommended
Code Scanning (Advanced)	Required	Recommended	Optional
Security-Extended Queries	Required	Recommended	Optional
Merge Protection	Required	Recommended	Optional
Dependabot Alerts	Required	Required	Required
Security Updates	Required	Required	Recommended
Version Updates	Recommended	Optional	Optional
Dependency Review	Required	Required	Optional
Delegated Dismissal	Required	Recommended	Optional

C SLA Reference

C.1 Alert Remediation SLAs

Severity	Tier 0	Tier 1	Tier 2
Critical	3 calendar days	7 calendar days	14 calendar days
High	7 calendar days	14 calendar days	30 calendar days
Medium	30 calendar days	60 calendar days	90 calendar days
Low	90 calendar days	120 calendar days	Best effort

C.2 Triage SLAs

Alert Type	Definition	SLA
Valid secret (external exposure)	Active credential committed to public/fork-visible repo	1 hour
Valid secret (internal)	Active credential in private repository	4 hours
Critical code scanning alert	RCE, auth bypass, critical injection	24 hours
High code scanning alert	SQL injection, XSS, SSRF	48 hours
Critical dependency alert	CVE with known exploits, EPSS > 0.7	24 hours

D Communication Templates

D.1 Rollout Announcement

```
Subject: [Announcement] GitHub Advanced Security Now Enabled

Hi Engineering Team,

We are pleased to announce the rollout of GitHub Advanced Security
    (GHAS)
across our repositories. This initiative will significantly strengthen
our security posture by:

- Detecting leaked credentials before they cause incidents
- Identifying security vulnerabilities in our code
- Providing visibility into vulnerable dependencies

WHAT'S ENABLED:
- Secret scanning: Detects committed credentials
- Code scanning: Identifies security vulnerabilities via CodeQL
- Dependabot: Alerts on vulnerable dependencies

WHAT YOU NEED TO DO:
1. Review any existing alerts in your repository's Security tab
2. Follow the remediation guides linked in each alert
3. Contact #appsec-support with questions

RESOURCES:
- Internal Wiki: [link]
- Remediation Guides: [link]
- FAQ: [link]

Questions? Reach out in #appsec-support.

Best,
Application Security Team
```

D.2 Enforcement Notification

```
Subject: [Action Required] Security Merge Protection Enabled on [Repo]

Hi [Team],

Starting [DATE], merge protection for security alerts will be enforced
on the [repository-name] repository.

WHAT THIS MEANS:
- Pull requests with Critical or High severity code scanning alerts
  will be blocked from merging
- You must either fix the vulnerability or request an exception

HOW TO RESOLVE BLOCKED PRs:
1. Review the alert details in the PR's Security tab
2. Fix the vulnerability and push updated code
3. If false positive, request security team review
4. For accepted risks, follow the exception process

EXCEPTION PROCESS:
[Link to exception request form]

This change helps us maintain security standards for our most critical
systems. Thank you for your cooperation.

Questions? Contact #appsec-support.

AppSec Team
```

E Glossary

Term	Definition	Documentation
CodeQL	GitHub's semantic code analysis engine used for code scanning	Link
CVSS	Common Vulnerability Scoring System; standardized severity scoring	–
Dependabot	GitHub's automated dependency update and vulnerability alerting service	Link
Dependency Graph	Summary of project dependencies	Link
Dependency Review	PR-time dependency analysis	Link
EPSS	Exploit Prediction Scoring System; probability of exploitation	–
GHAS	GitHub Advanced Security; umbrella term for security features	Link
KEV	Known Exploited Vulnerabilities catalog maintained by CISA	–
MTTR	Mean Time to Remediate; average time to fix a vulnerability	–
Push Protection	Feature that blocks commits containing secrets	Link
SARIF	Static Analysis Results Interchange Format; standard for tool results	–
Security Configuration	GitHub's policy-as-code for applying security settings	Link
Security Overview	Dashboard showing security status across an organization	Link
SLA	Service Level Agreement; committed remediation timeframes	–
Validity Check	Verification that a detected secret is currently active	–

A Complete Documentation Reference

This appendix provides a comprehensive, organized reference to all official GitHub documentation relevant to GHAS implementation.

A.1 GitHub Advanced Security Core

Topic	URL
About GitHub Advanced Security	https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security
GitHub Security Features	https://docs.github.com/en/code-security/getting-started/github-security-features
Adopting GHAS at Scale	https://docs.github.com/en/enterprise-cloud@latest/code-security/adopting-github-advanced-security-at-scale/introduction-to-adopting-github-advanced-security-at-scale
Code Security Portal	https://docs.github.com/en/code-security

A.2 Security Configurations and Organization Settings

Topic	URL
Enabling Security at Scale	https://docs.github.com/en/code-security/securing-your-organization/introduction-to-securing-your-organization-at-scale/about-enabling-security-features-at-scale
Choosing Security Configurations	https://docs.github.com/en/code-security/securing-your-organization/introduction-to-securing-your-organization-at-scale/choosing-a-security-configuration-for-your-repositories
Managing Org Security Settings	https://docs.github.com/en/organizations/keeping-your-organization-secure/managing-security-settings-for-your-organization/managing-security-and-analysis-settings-for-your-organization

A.3 Security Overview and Reporting

Topic	URL
About Security Overview	https://docs.github.com/en/code-security/security-overview/about-security-overview
Viewing Security Insights	https://docs.github.com/en/code-security/security-overview/viewing-security-insights
Assessing Security Risk	https://docs.github.com/en/code-security/security-overview/assessing-code-security-risk
Auditing Security Alerts	https://docs.github.com/en/code-security/getting-started/auditing-security-alerts

A.4 Secret Scanning

Topic	URL
Secret Scanning Portal	https://docs.github.com/en/code-security/secret-scanning
About Push Protection	https://docs.github.com/en/code-security/secret-scanning/introduction/about-push-protection
Enabling Secret Scanning	https://docs.github.com/en/code-security/secret-scanning/enabling-secret-scanning-features
Enabling Push Protection	https://docs.github.com/en/code-security/secret-scanning/enabling-secret-scanning-features/enabling-push-protection-for-your-repository
Push Protection CLI	https://docs.github.com/en/code-security/secret-scanning/working-with-secret-scanning-and-push-protection/working-with-push-protection-from-the-command-line
Push Protection for Users	https://docs.github.com/en/code-security/secret-scanning/working-with-secret-scanning-and-push-protection/push-protection-for-users
Working with Secret Scanning	https://docs.github.com/en/code-security/secret-scanning/working-with-secret-scanning-and-push-protection

A.5 Code Scanning

Topic	URL
Code Scanning Portal	https://docs.github.com/en/code-security/code-scanning
About CodeQL	https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning-with-codeql
Default Setup	https://docs.github.com/en/code-security/code-scanning/enabling-code-scanning/configuring-default-setup-for-code-scanning
Advanced Setup	https://docs.github.com/en/code-security/code-scanning/creating-an-advanced-setup-for-code-scanning
Query Suites	https://docs.github.com/en/code-security/code-scanning/managing-your-code-scanning-configuration/codeql-query-suites
CodeQL CLI	https://docs.github.com/code-security/secure-coding/about-codeql-code-scanning-in-your-ci-system
Compiled Languages	https://docs.github.com/en/code-security/code-scanning/creating-an-advanced-setup-for-code-scanning/codeql-code-scanning-for-compiled-languages

A.6 Dependabot

Topic	URL
Dependabot Portal	https://docs.github.com/en/code-security/dependabot
About Dependabot Alerts	https://docs.github.com/code-security/dependabot/dependabot-alerts/about-dependabot-alerts
Viewing Alerts	https://docs.github.com/en/code-security/dependabot/dependabot-alerts/viewing-and-updating-dependabot-alerts
Configuring Alerts	https://docs.github.com/en/code-security/dependabot/dependabot-alerts/configuring-dependabot-alerts
About Security Updates	https://docs.github.com/en/code-security/dependabot/dependabot-security-updates/about-dependabot-security-updates
Configuring Security Updates	https://docs.github.com/github/managing-security-vulnerabilities/configuring-dependabot-security-updates

A.7 Supply Chain Security

Topic	URL
About Dependency Graph	https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph
Configuring Dependency Graph	https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/configuring-the-dependency-graph
About Dependency Review	https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-dependency-review
Dependency Review Action	https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/customizing-your-dependency-review-action-configuration

End of Document

Version 3.0 — January 19, 2026

All documentation links reference `docs.github.com` and were verified at publication.
Validate certification scope against the current official exam guide.
Align enforcement thresholds to *[Company Name]* risk tolerance and engineering capacity.