Event-Driven Integration Blueprint

ServiceNow + GitHub Secret Protection & Code Security

Dependabot, Code Scanning, Secret Scanning, and
Security Campaign Automation

Prepared for Internal AppSec / SecOps Enablement

Architecture & Engineering Reference

Version 2.0 — Updated: January 14, 2026

Abstract

This document provides a comprehensive architectural blueprint for integrating GitHub's security products (Secret Protection and Code Security, formerly bundled as GitHub Advanced Security) with ServiceNow's Vulnerability Response and Application Vulnerability Response modules. The integration enables automated intake, triage, assignment, SLA enforcement, and remediation tracking for Dependabot alerts, Code Scanning alerts, and Secret Scanning alerts through event-driven webhook processing, REST API reconciliation, and native ServiceNow integration patterns.

Key Updates (2025–2026): This revision incorporates GitHub's April 2025 product unbundling into GitHub Secret Protection and GitHub Code Security, the November 2025 GA release of alert assignees and security campaigns with corresponding webhook and REST API support, enhanced Copilot Autofix integration, and updated ServiceNow GitHub Application Vulnerability Integration capabilities.

Contents

1 Executive Summary **4**

1.1 Recommended Strategy: Hybrid Architecture 4

2 Target Outcomes and Non-Functional Requirements **5**

2.1 Operational Outcomes 5

2.2 Non-Functional Requirements (NFRs) 5

3 Reference Architecture **6**

3.1 High-Level Component Diagram 6

3.2 Data Flow: Event Path (Near Real-Time) 7

3.3 Data Flow: Baseline Path (Authoritative Ingestion) 8

4 Integration Patterns and Selection Criteria **8**

4.1 Pattern A: OOTB Import-Driven (Scheduled) 8

4.2 Pattern B: Webhook-First Event Processing 9

4.3 Pattern C: Webhooks + Queue + Reconciliation (Recommended) 9

5 Data Model Mapping **10**
5.1 ServiceNow Domain Objects 10
5.2 Canonical Alert Identifier 10
5.3 State Mapping Matrix 10
5.4 Alert Assignee Mapping (New 2025) 11

6 GitHub Configuration **11**
6.1 Webhook Events to Enable 11
6.2 Webhook Security Controls 12
6.3 GitHub App Authentication (Recommended) 12
6.4 REST APIs for Reconciliation 13

7 ServiceNow Configuration **13**
7.1 Baseline: GitHub Application Vulnerability Integration 13
7.2 Event-Driven Inbound Options 13
7.2.1 Option 1: External Receiver (Recommended) 13
7.2.2 Option 2: ServiceNow Scripted REST API 14
7.2.3 Option 3: IntegrationHub with External Trigger 14
7.3 Recommended ServiceNow Workflow Components 14

8 Implementation: Webhook Receiver **15**
8.1 Node.js/TypeScript Implementation 15
8.2 Go Implementation 20
8.3 ServiceNow Scripted REST API Implementation 26

9 Implementation: Reconciliation Worker **31**
9.1 Python Reconciliation Service 31

10 Implementation: GitHub Actions Integration **43**
10.1 Security Results to ServiceNow DevOps 43

11 Automation Design: Playbooks and Workflows **46**
11.1 Triage and Prioritization Matrix 46
11.2 Assignment Resolution Strategy 47
11.3 SLA Configuration Matrix 47
11.4 Exception Management Workflow 47

12 Reliability Engineering **47**
12.1 Idempotency Implementation 47
12.2 Retry and Dead-Letter Strategy 48
12.3 Observability Requirements 48

13 Security Hardening Checklist **49**

14 Rollout Plan **50**
14.1 Phase 0: Prerequisites (Week 1–2) 50
14.2 Phase 1: Baseline Import (Week 3–4) 50
14.3 Phase 2: Webhook Event Path (Week 5–8) 50
14.4 Phase 3: Reconciliation and Advanced Features (Week 9–12) 50

14.5 Phase 4: Production Hardening (Week 13–16) . . . 51

15 Appendix A: Webhook Payload Examples 51

15.1 Code Scanning Alert Created (with Assignees) . . . 51

15.2 Secret Scanning Alert with Push Protection Bypass . . . 52

16 Appendix B: Normalized Event Envelope 53

17 Appendix C: Primary References 54

17.1 GitHub Documentation . . . 54

17.2 GitHub Changelog (2025) . . . 54

17.3 ServiceNow Documentation . . . 54

1 Executive Summary

An event-driven system integrating ServiceNow and GitHub's security products can be implemented to automate the complete vulnerability management lifecycle for:

- **Dependabot alerts** — Software Composition Analysis (SCA) covering dependency vulnerabilities, with automated PR generation and auto-merge capabilities for low-risk updates
- **Code Scanning alerts** — Static Application Security Testing (SAST) powered by CodeQL and third-party SARIF-producing tools, now with GA Copilot Autofix suggestions and alert assignees
- **Secret Scanning alerts** — Credential exposure detection including push protection bypass events, AI-powered detection, validity checking, and security campaigns for coordinated remediation

GitHub provides first-class webhook events for these alert families (`dependabot_alert`, `code_scanning_alert`, `secret_scanning_alert`) and mandates validating deliveries using the `X-Hub-Signature-256` HMAC header.[1]

Note

Product Evolution (April 2025): GitHub Advanced Security (GHAS) has been unbundled into two standalone products:

- **GitHub Secret Protection** ($19/month per active committer) — push protection, secret scanning, AI-powered detection, validity checking
- **GitHub Code Security** ($30/month per active committer) — code scanning, Copilot Autofix, security campaigns, Dependency Review Action

Both products are now available to GitHub Team plan customers without requiring GitHub Enterprise.[a]

[a] https://github.blog/changelog/2025-03-04-introducing-github-secret-protection-and-github-code-security/

On the ServiceNow side, the **GitHub Application Vulnerability Integration** (via the ServiceNow Store) provides the most direct path to import findings into **Vulnerability Response / Application Vulnerability Response** and manage remediation workflows at scale.[2]

1.1 Recommended Strategy: Hybrid Architecture

A production-grade design employs a **hybrid** approach combining three complementary integration patterns:

1. **Baseline Ingestion** — Official ServiceNow integration application for authoritative normalization, deduplication, GHSA-to-CVE mapping, and SecOps domain model alignment
2. **Event-Driven Updates** — GitHub webhooks into a hardened receiver (API Gateway, cloud function, or ServiceNow Scripted REST API) for immediate routing, assignment, security campaign synchronization, and workflow triggers

[1] https://docs.github.com/en/webhooks/using-webhooks/validating-webhook-deliveries
[2] https://store.servicenow.com/store/app/006dafe21b646a50a85b16db234bcba2

3. **Reconciliation Polling** — Scheduled jobs using GitHub REST APIs to detect missed events, synchronize assignee changes, campaign status updates, and ensure state convergence

This approach preserves OOTB integration benefits while achieving near-real-time automation with provable correctness.

2 Target Outcomes and Non-Functional Requirements

2.1 Operational Outcomes

- **Time-to-triage reduction**: Automated classification, routing, and prioritization at ingestion with sub-minute latency for critical alerts
- **SLA enforcement**: Deterministic due dates based on severity, exploitability (EPSS), asset criticality, and alert type (secrets typically warrant shortest SLAs)
- **Automated lifecycle tracking**: State synchronization (open → in-progress → fixed/dismissed) bidirectionally between GitHub and ServiceNow
- **Security campaign orchestration**: Synchronized campaign status, deadlines, and assignee notifications across both platforms
- **Audit-grade traceability**: Immutable event log of alert state transitions, assignment changes, and workflow actions with correlation IDs
- **Copilot Autofix integration**: Track autofix suggestions, acceptance rates, and time-to-remediation metrics

2.2 Non-Functional Requirements (NFRs)

Category	**Requirements**
Security	• HMAC-SHA256 signature verification for all webhook deliveries • Least-privilege GitHub App tokens with fine-grained permissions • Secrets vaulting (never log secret values from secret scanning) • OAuth 2.0 / mutual TLS for ServiceNow API authentication • IP allow-listing and WAF protection for webhook endpoints
Reliability	• Idempotent processing using `X-GitHub-Delivery` and canonical alert keys • Exponential backoff retries with jitter • Dead-letter queues (DLQ) with alerting and manual replay capability • At-least-once delivery guarantee with deduplication

Scalability	• Burst handling for webhook storms (large codebase scans) • Queue-based backpressure with configurable concurrency • GitHub API rate limit awareness (5,000 requests/hour for Apps) • Horizontal scaling of webhook processors
Maintainability	• Clear ownership boundaries: GitHub config / integration code / ServiceNow workflows • Version-controlled mapping rules and transformation logic • Staged rollout capability with canary repositories • Comprehensive logging and distributed tracing

3 Reference Architecture

3.1 High-Level Component Diagram

The integration architecture consists of five primary layers:

1. **Source Layer (GitHub)**
   - GitHub Secret Protection (secret scanning, push protection)
   - GitHub Code Security (CodeQL, code scanning, Copilot Autofix)
   - Dependabot (dependency vulnerability alerts, automated PRs)
   - Security Campaigns (coordinated remediation initiatives)
2. **Transport Layer (Webhooks + APIs)**
   - Outbound webhooks with HMAC signatures
   - REST API v3 for reconciliation and bidirectional sync
   - GraphQL API for complex queries and bulk operations
3. **Ingestion Layer (Webhook Receiver)**
   - Signature validation and request authentication
   - Event normalization and enrichment
   - Durable queue publication (SQS, Pub/Sub, Azure Service Bus)
   - DLQ handling and replay infrastructure
4. **Processing Layer (Integration Services)**
   - Event transformation and mapping
   - ServiceNow API client with retry logic
   - Reconciliation scheduler

- Metrics and observability collection

5. **Destination Layer (ServiceNow)**

   - Scripted REST API or IntegrationHub endpoint
   - Vulnerability Response / Application Vulnerability Response
   - Flow Designer workflows for automation
   - CMDB integration for asset correlation

3.2 Data Flow: Event Path (Near Real-Time)

1. GitHub emits a webhook delivery on alert lifecycle changes:

   - **Code Scanning**: `created`, `reopened`, `closed_by_user`, `fixed`, `appeared_in_branch`, `reopened_by_user`
   - **Secret Scanning**: `created`, `resolved`, `revoked`, `reopened`, `validated` (validity status change)
   - **Dependabot**: `created`, `dismissed`, `fixed`, `reintroduced`, `auto_dismissed`, `auto_reopened`
   - **New (2025)**: Alert assignee changes trigger webhook events for both code scanning and secret scanning alerts[3]

2. The **Webhook Receiver** validates:

   - HMAC signature via `X-Hub-Signature-256` header
   - Event type via `X-GitHub-Event` header
   - Unique delivery ID via `X-GitHub-Delivery` header
   - Optional: IP allow-list against GitHub's webhook IP ranges

3. Receiver publishes normalized event to durable queue with metadata:

   - Original payload (optionally redacted for secrets)
   - Computed canonical alert key
   - Received timestamp and processing priority

4. ServiceNow consumer processes the event:

   - Idempotent upsert into staging/event table
   - Map to Vulnerability Response domain objects
   - Trigger Flow Designer subflows for triage, assignment, and SLA
   - Update GitHub alert state/assignee if bidirectional sync enabled

[3] https://github.blog/changelog/2025-12-16-code-scanning-alert-assignees-are-now-generally-available/

3.3 Data Flow: Baseline Path (Authoritative Ingestion)

The ServiceNow **GitHub Application Vulnerability Integration** provides:

- **Code Scanning Integration**: SAST data import with rule metadata, CWE mapping, and CVSS enrichment
- **Dependabot Integration**: SCA data with advisory severity, GHSA-to-CVE correlation, and remediation guidance
- **Secret Scanning Integration**: Credential exposure findings with secret type, validity status, and location metadata
- **Recent Improvements (2025)**: Enhanced GHSA ID fallback when CVE is absent, improved delta sync for Dependabot alerts using sort field parameters[4]

Best Practice

Hybrid Approach Rationale: GitHub's integration guidance recommends webhooks for long-term metric collection and real-time alerting, while REST APIs provide point-in-time snapshots requiring polling infrastructure. The ServiceNow integration app provides optimal alignment to SecOps data models. Combining both yields speed *and* correctness.

4 Integration Patterns and Selection Criteria

4.1 Pattern A: OOTB Import-Driven (Scheduled)

Use when: Fastest time-to-value is priority and minutes-to-hours latency is acceptable for non-critical alerts.

Mechanism: Configure the GitHub Application Vulnerability Integration to ingest findings on a scheduled cadence (hourly, daily).

Strengths:

- Normalized ingestion with GHSA/CVE deduplication
- Native SecOps workflows (assignment, exception handling, false positives)
- Minimal custom code; vendor-supported upgrade path
- Automatic CWE and CVSS enrichment

Limitations:

- Not truly event-driven; insufficient for rapid-response use cases
- Secret exposure may remain unaddressed for hours
- No support for real-time assignee or campaign synchronization

[4] https://store.servicenow.com/store/app/006dafe21b646a50a85b16db234bcba2

4.2 Pattern B: Webhook-First Event Processing

Use when: Near-real-time automation is required (seconds to minutes latency).
Mechanism: Configure organization or repository webhooks for security alert events:

- `code_scanning_alert`
- `dependabot_alert`
- `secret_scanning_alert`
- `security_and_analysis` (for enablement/disablement tracking)

Strengths:

- Immediate alerting, incident creation, and paging/escalation
- Avoids API polling overhead and rate limit consumption
- Supports real-time assignee and campaign synchronization
- Can trigger lightweight workflows without waiting for import cadence

Limitations:

- Requires idempotency and replay design
- Possible missed deliveries without reconciliation
- More complex infrastructure requirements

4.3 Pattern C: Webhooks + Queue + Reconciliation (Recommended)

Use when: Near-real-time processing *and* provable correctness are both required.
Mechanism:

- Webhooks for instant triggers with acknowledgment within 10 seconds
- Durable message queue (SQS, Pub/Sub, Service Bus) with DLQ
- Periodic REST reconciliation for convergence (every 15–60 minutes)
- Security campaign status sync via REST API

Warning

For **secret scanning alerts**, always err on the side of immediate action. Leaked credentials can be exploited within minutes of exposure. Pattern C is strongly recommended for any deployment handling secret scanning.

5 Data Model Mapping

5.1 ServiceNow Domain Objects

ServiceNow Vulnerability Response uses the following core tables:

- **Vulnerable Items** (`sn_vul_vulnerable_item`): Infrastructure-style vulnerabilities tied to CIs
- **Application Vulnerable Items** (`sn_vul_app_vulnerable_item`): Application-scoped findings from third-party scanners
- **Third-Party Vulnerabilities** (`sn_vul_third_party_entry`): External vulnerability definitions (CVE, GHSA, CWE)
- **Vulnerability Groups**: Logical groupings for bulk operations and reporting

The GitHub Application Vulnerability Integration maps findings to Application Vulnerable Items with proper third-party entry linkage.

5.2 Canonical Alert Identifier

Define a deterministic, unique identifier for idempotent upserts:

```json
{
  "format": "<provider>:<org>/<repo>:<alert_type>:<alert_number>",
  "examples": [
    "github:acme/payments:dependabot:1234",
    "github:acme/api-gateway:code_scanning:5678",
    "github:acme/infrastructure:secret_scanning:9012"
  ]
}
```

Listing 1: Canonical Alert Key Format

Store this key in:

- Custom event staging table (`x_ghas_event`)
- External reference field on the target SecOps record
- Correlation ID for bidirectional synchronization

5.3 State Mapping Matrix

GitHub Status	**ServiceNow State**	**Automation Actions**
`open` / `active`	Open / New	Create or reopen task; start SLA clock; assign to owner
`dismissed`	Risk Accepted / False Positive / Exception	Require documented reason; risk acceptance owner approval; set expiration date; record compensating controls

`fixed` / `resolved`	Resolved	Validate via PR merge, dependency update, or secret rotation; stop SLA clock; update metrics
`reopened`	Reopened	Re-trigger assignment; recalculate SLA; notify original assignee
`auto_dismissed`	Auto-Closed (Dependency)	Record auto-dismissal reason; no manual action required
`reintroduced`	Reopened (Regression)	High-priority flag; escalate if repeat occurrence

5.4 Alert Assignee Mapping (New 2025)

With alert assignees now GA for both code scanning and secret scanning:

```json
{
  "alert_key": "github:acme/api:code_scanning:1234",
  "assignees": [
    {
      "github_login": "jsmith",
      "servicenow_user": "john.smith@acme.com",
      "assigned_at": "2026-01-14T10:30:00Z",
      "assigned_by": "security-bot"
    }
  ],
  "sync_direction": "github_to_servicenow",
  "last_synced": "2026-01-14T10:30:05Z"
}
```

Listing 2: Assignee Synchronization Payload Structure

6 GitHub Configuration

6.1 Webhook Events to Enable

At minimum, enable these webhook events at the organization level:

```yaml
webhook_events:
  # Core security alert events
  - code_scanning_alert       # SAST findings lifecycle
  - dependabot_alert          # SCA/dependency vulnerabilities
  - secret_scanning_alert     # Credential exposure detection

  # Supplementary events (recommended)
  - security_and_analysis     # Feature enablement changes
  - repository_advisory       # Private vulnerability reports

  # CI/CD correlation (optional)
  - check_run                 # CodeQL analysis completion
  - workflow_run              # Security workflow status
```

Listing 3: Required Webhook Events

6.2 Webhook Security Controls

1. **Configure webhook secret**: Generate a cryptographically strong secret (minimum 32 characters):

```
openssl rand -base64 32

```

Listing 4: Generate Webhook Secret

2. **Validate X-Hub-Signature-256**: GitHub computes HMAC-SHA256 of the payload body using your secret
3. **Log X-GitHub-Delivery**: Unique UUID for each delivery; use for deduplication and replay tracking
4. **Verify X-GitHub-Event**: Ensure event type matches expected values to prevent injection
5. **Optional IP filtering**: GitHub publishes webhook IP ranges via the Meta API

6.3 GitHub App Authentication (Recommended)

For production integrations, use a GitHub App instead of personal access tokens:

```
github_app_permissions:
  repository_permissions:
    # Read access for alert data
    security_events: read
    contents: read
    metadata: read

    # Write access for bidirectional sync
    security_events: write  # To update alert state/assignees

  organization_permissions:
    # For org-wide alert access
    organization_administration: read

  webhook_events:
    - code_scanning_alert
    - dependabot_alert
    - secret_scanning_alert
    - security_and_analysis
```

Listing 5: GitHub App Required Permissions

Benefits of GitHub App:

- Higher rate limits (5,000 requests/hour vs 5,000/hour for PATs)
- Fine-grained permissions scoped to specific repositories
- Installation tokens with short TTL (1 hour)
- Better audit trail and revocation capabilities
- No dependency on individual user accounts

6.4 REST APIs for Reconciliation

These endpoints support periodic snapshots and repair:

Alert Type	API Endpoint

Code Scanning	`GET /repos/{owner}/{repo}/code-scanning/alerts`
	`GET /orgs/{org}/code-scanning/alerts`

Secret Scanning	`GET /repos/{owner}/{repo}/secret-scanning/alerts`
	`GET /orgs/{org}/secret-scanning/alerts`

Dependabot	`GET /repos/{owner}/{repo}/dependabot/alerts`
	`GET /orgs/{org}/dependabot/alerts`

Security Campaigns	`GET /orgs/{org}/security-campaigns` (New 2025)

7 ServiceNow Configuration

7.1 Baseline: GitHub Application Vulnerability Integration

Install and configure from the ServiceNow Store:

1. Navigate to **System Applications** → **All Available Applications** → **All**
2. Search for "GitHub Application Vulnerability Integration"
3. Install the application (requires SecOps license)
4. Configure credentials:
   - Create a Connection record with GitHub App credentials
   - Configure organization scope and repository filters
   - Set import schedule (recommended: hourly for critical repos)

7.2 Event-Driven Inbound Options

7.2.1 Option 1: External Receiver (Recommended)

Deploy a hardened webhook receiver outside ServiceNow:

- **Infrastructure**: API Gateway (AWS, Azure, GCP) + serverless function
- **Responsibilities**: Signature validation, rate limiting, queue publication
- **Benefits**: Enhanced security, independent scaling, multi-region support
- **ServiceNow integration**: Authenticated REST call with OAuth bearer token

7.2.2 Option 2: ServiceNow Scripted REST API

Direct webhook endpoint within ServiceNow:

- **Use case**: Simpler deployments, moderate volumes
- **Requirements**: Strict ACLs, signature verification, async processing
- **Limitation**: ServiceNow must respond within webhook timeout

7.2.3 Option 3: IntegrationHub with External Trigger

Leverage ServiceNow's IntegrationHub:

- Create External Trigger Definition tied to Flow Designer flow
- Configure authentication (OAuth, API Key, mutual TLS)
- Use GitHub Spoke for bidirectional management where available

7.3 Recommended ServiceNow Workflow Components

```
flow_designer_subflows:
  # Inbound Processing
  - name: ProcessGHASWebhookEvent
    trigger: External Trigger / Scripted REST
    actions:
      - ValidatePayload
      - ComputeCanonicalKey
      - UpsertStagingRecord
      - QueueForProcessing

  # Triage and Classification
  - name: TriageAndPrioritizeAlert
    inputs: [alert_type, severity, repo_criticality]
    actions:
      - ComputeNormalizedPriority
      - ApplyBusinessRules
      - SetSLADueDates

  # Assignment
  - name: AssignToOwningTeam
    inputs: [repo_name, alert_type, priority]
    actions:
      - LookupCMDBApplication
      - ResolveOwnerFromCodeowners
      - FallbackToTriageQueue
      - NotifyAssignee

  # Lifecycle Management
  - name: HandleAlertStateChange
    inputs: [current_state, new_state, reason]
    actions:
      - ValidateStateTransition
      - UpdateVulnerableItem
      - SyncToGitHub (if bidirectional)
      - RecordAuditTrail

  # Exception Handling
  - name: ProcessExceptionRequest
    inputs: [alert_key, reason, approver, expiration]
    actions:
      - ValidateApproverAuthority
      - RecordCompensatingControls
      - SetExpirationReminder
      - UpdateGitHubDismissalReason
```

Listing 6: Flow Designer Subflow Architecture

8 Implementation: Webhook Receiver

8.1 Node.js/TypeScript Implementation

The following production-ready implementation includes signature validation, structured logging, queue integration, and comprehensive error handling:

```
import express, { Request, Response, NextFunction } from 'express';
import crypto from 'crypto';
import { SQSClient, SendMessageCommand } from '@aws-sdk/client-sqs';
import pino from 'pino';

// Configuration
const config = {
  port: parseInt(process.env.PORT || '8080'),
  webhookSecret: process.env.GITHUB_WEBHOOK_SECRET!,
  sqsQueueUrl: process.env.SQS_QUEUE_URL!,
  awsRegion: process.env.AWS_REGION || 'us-east-1',
  servicenowUrl: process.env.SERVICENOW_INGEST_URL,
  servicenowToken: process.env.SERVICENOW_BEARER_TOKEN,
};

// Initialize clients
const logger = pino({ level: 'info' });
const sqs = new SQSClient({ region: config.awsRegion });
const app = express();

// CRITICAL: Use raw body for HMAC verification
app.use('/webhook', express.raw({ type: 'application/json', limit: '5mb
    ' }));
app.use(express.json());

// Interfaces
interface WebhookHeaders {
  signature: string;
  event: string;
  delivery: string;
}
```

```typescript

interface ProcessedEvent {
  deliveryId: string;
  eventType: string;
  action: string;
  alertKey: string;
  repository: string;
  alertNumber: number;
  severity: string | null;
  state: string;
  assignees: string[];
  receivedAt: string;
  payload: object;
}

// Timing-safe comparison to prevent timing attacks
function timingSafeEqual(a: string, b: string): boolean {
  if (!a || !b) return false;
  const bufA = Buffer.from(a);
  const bufB = Buffer.from(b);
  if (bufA.length !== bufB.length) return false;
  return crypto.timingSafeEqual(bufA, bufB);
}

// Verify GitHub webhook signature
function verifySignature(payload: Buffer, signature: string): boolean {
  const hmac = crypto.createHmac('sha256', config.webhookSecret);
  hmac.update(payload);
  const expected = `sha256=${hmac.digest('hex')}`;
  return timingSafeEqual(signature, expected);
}

// Extract headers with validation
function extractHeaders(req: Request): WebhookHeaders {
  return {
    signature: req.header('X-Hub-Signature-256') || '',
    event: req.header('X-GitHub-Event') || '',
    delivery: req.header('X-GitHub-Delivery') || '',
  };
}

// Compute canonical alert key
function computeAlertKey(event: string, payload: any): string {
  const repo = payload.repository?.full_name || 'unknown';
  const alertNumber = payload.alert?.number || 0;

  const alertTypeMap: Record<string, string> = {
    'code_scanning_alert': 'code_scanning',
    'dependabot_alert': 'dependabot',
    'secret_scanning_alert': 'secret_scanning',
  };

  const alertType = alertTypeMap[event] || event;
  return `github:${repo}:${alertType}:${alertNumber}`;
```

```
}

// Extract severity from various alert types
function extractSeverity(event: string, payload: any): string | null {
  switch (event) {
    case 'code_scanning_alert':
      return payload.alert?.rule?.severity ||
             payload.alert?.rule?.security_severity_level || null;
    case 'dependabot_alert':
      return payload.alert?.security_advisory?.severity ||
             payload.alert?.security_vulnerability?.severity || null;
    case 'secret_scanning_alert':
      // Secret scanning doesn't have severity; return 'critical' by
    default
      return 'critical';
    default:
      return null;
  }
}

// Extract assignees (new 2025 feature)
function extractAssignees(payload: any): string[] {
  const assignees = payload.alert?.assignees || [];
  return assignees.map((a: any) => a.login);
}

// Process and normalize the webhook event
function processEvent(headers: WebhookHeaders, payload: any):
    ProcessedEvent {
  return {
    deliveryId: headers.delivery,
    eventType: headers.event,
    action: payload.action || 'unknown',
    alertKey: computeAlertKey(headers.event, payload),
    repository: payload.repository?.full_name || 'unknown',
    alertNumber: payload.alert?.number || 0,
    severity: extractSeverity(headers.event, payload),
    state: payload.alert?.state || 'unknown',
    assignees: extractAssignees(payload),
    receivedAt: new Date().toISOString(),
    payload: sanitizePayload(headers.event, payload),
  };
}

// Sanitize payload to remove sensitive data
function sanitizePayload(event: string, payload: any): object {
  if (event === 'secret_scanning_alert') {
    // CRITICAL: Never store actual secret values
    const sanitized = { ...payload };
    if (sanitized.alert) {
      sanitized.alert = {
        ...sanitized.alert,
        secret: '[REDACTED]',
      };
    }
    return sanitized;
  }
  return payload;
}

// Publish to SQS queue
async function publishToQueue(event: ProcessedEvent): Promise<void> {
  const command = new SendMessageCommand({
    QueueUrl: config.sqsQueueUrl,
    MessageBody: JSON.stringify(event),
    MessageAttributes: {
      EventType: { DataType: 'String', StringValue: event.eventType },
      AlertKey: { DataType: 'String', StringValue: event.alertKey },
      Priority: { DataType: 'String', StringValue: event.severity || 'medium' },
    },
    MessageDeduplicationId: event.deliveryId,
    MessageGroupId: event.repository.replace('/', '-'),
  });

  await sqs.send(command);
}

// Allowed event types
const ALLOWED_EVENTS = new Set([
  'code_scanning_alert',
  'dependabot_alert',
  'secret_scanning_alert',
  'security_and_analysis',
]);

// Main webhook handler
app.post('/webhook/github', async (req: Request, res: Response) => {
  const startTime = Date.now();
  const headers = extractHeaders(req);

  // Log incoming request
  logger.info({
    delivery: headers.delivery,
    event: headers.event,
    userAgent: req.header('User-Agent'),
  }, 'Webhook received');

  // Validate signature
  if (!verifySignature(req.body as Buffer, headers.signature)) {
    logger.warn({ delivery: headers.delivery }, 'Invalid signature');
    return res.status(401).json({
      error: 'invalid_signature',
      message: 'Webhook signature validation failed',
    });
  }

  // Validate event type
  if (!ALLOWED_EVENTS.has(headers.event)) {
    logger.info({
      delivery: headers.delivery,
      event: headers.event,
    }, 'Ignored event type');
    return res.status(200).json({
      status: 'ignored',
      reason: 'event_type_not_processed',
    });
  }

  try {
    const payload = JSON.parse(req.body.toString());
    const event = processEvent(headers, payload);

    // Publish to queue for async processing
    await publishToQueue(event);

    const duration = Date.now() - startTime;
    logger.info({
      delivery: headers.delivery,
      alertKey: event.alertKey,
      action: event.action,
      duration,
    }, 'Event queued successfully');

    return res.status(202).json({
      status: 'accepted',
      delivery_id: headers.delivery,
      alert_key: event.alertKey,
    });

  } catch (error) {
    logger.error({
      delivery: headers.delivery,
      error: error instanceof Error ? error.message : 'Unknown error',
    }, 'Processing failed');

    // Return 500 so GitHub will retry
    return res.status(500).json({
      error: 'processing_failed',
      delivery_id: headers.delivery,
    });
  }
});

// Health check endpoint
app.get('/health', (req: Request, res: Response) => {
  res.status(200).json({
    status: 'healthy',
    timestamp: new Date().toISOString(),
  });
});
```

```
// Start server
app.listen(config.port, () => {
  logger.info({ port: config.port }, 'Webhook receiver started');
});
```

Listing 7: TypeScript Webhook Receiver with Full Validation

8.2 Go Implementation

High-performance Go implementation for high-volume deployments:

```
package main

import (
  "context"
  "crypto/hmac"
  "crypto/sha256"
  "crypto/subtle"
  "encoding/hex"
  "encoding/json"
  "fmt"
  "io"
  "log/slog"
  "net/http"
  "os"
  "strings"
  "time"

  "github.com/aws/aws-sdk-go-v2/config"
  "github.com/aws/aws-sdk-go-v2/service/sqs"
)

type Config struct {
  Port          string
  WebhookSecret string
  SQSQueueURL   string
}

type WebhookEvent struct {
  DeliveryID  string    `json:"delivery_id"`
  EventType   string    `json:"event_type"`
  Action      string    `json:"action"`
  AlertKey    string    `json:"alert_key"`
  Repository  string    `json:"repository"`
  AlertNumber int       `json:"alert_number"`
  Severity    string    `json:"severity,omitempty"`
  State       string    `json:"state"`
  Assignees   []string  `json:"assignees"`
  ReceivedAt  time.Time `json:"received_at"`
  Payload     any       `json:"payload"`
}

type WebhookHandler struct {
  config    *Config
```

```go
  sqsClient *sqs.Client
  logger    *slog.Logger
}

func NewWebhookHandler(cfg *Config) (*WebhookHandler, error) {
  awsCfg, err := config.LoadDefaultConfig(context.Background())
  if err != nil {
    return nil, fmt.Errorf("failed to load AWS config: %w", err)
  }

  return &WebhookHandler{
    config:    cfg,
    sqsClient: sqs.NewFromConfig(awsCfg),
    logger:    slog.New(slog.NewJSONHandler(os.Stdout, nil)),
  }, nil
}

func (h *WebhookHandler) verifySignature(payload []byte, signature
    string) bool {
  if !strings.HasPrefix(signature, "sha256=") {
    return false
  }

  mac := hmac.New(sha256.New, []byte(h.config.WebhookSecret))
  mac.Write(payload)
  expected := "sha256=" + hex.EncodeToString(mac.Sum(nil))

  return subtle.ConstantTimeCompare([]byte(signature), []byte(expected)
    ) == 1
}

func (h *WebhookHandler) computeAlertKey(eventType string, payload map[
    string]any) string {
  repo := "unknown"
  if r, ok := payload["repository"].(map[string]any); ok {
    if fn, ok := r["full_name"].(string); ok {
      repo = fn
    }
  }

  var alertNumber float64
  if alert, ok := payload["alert"].(map[string]any); ok {
    if num, ok := alert["number"].(float64); ok {
      alertNumber = num
    }
  }

  alertTypeMap := map[string]string{
    "code_scanning_alert":   "code_scanning",
    "dependabot_alert":      "dependabot",
    "secret_scanning_alert": "secret_scanning",
  }

  alertType := alertTypeMap[eventType]
  if alertType == "" {
    alertType = eventType
  }

  return fmt.Sprintf("github:%s:%s:%.0f", repo, alertType, alertNumber)
}

func (h *WebhookHandler) extractSeverity(eventType string, payload map[
    string]any) string {
  alert, ok := payload["alert"].(map[string]any)
  if !ok {
    return ""
  }

  switch eventType {
  case "code_scanning_alert":
    if rule, ok := alert["rule"].(map[string]any); ok {
      if sev, ok := rule["severity"].(string); ok {
        return sev
      }
      if sev, ok := rule["security_severity_level"].(string); ok {
        return sev
      }
    }
  case "dependabot_alert":
    if adv, ok := alert["security_advisory"].(map[string]any); ok {
      if sev, ok := adv["severity"].(string); ok {
        return sev
      }
    }
  case "secret_scanning_alert":
    return "critical"
  }
  return ""
}

func (h *WebhookHandler) extractAssignees(payload map[string]any) []
    string {
  var assignees []string
  alert, ok := payload["alert"].(map[string]any)
  if !ok {
    return assignees
  }

  if ass, ok := alert["assignees"].([]any); ok {
    for _, a := range ass {
      if user, ok := a.(map[string]any); ok {
        if login, ok := user["login"].(string); ok {
          assignees = append(assignees, login)
        }
      }
    }
  }
  return assignees
```

```go
}

func (h *WebhookHandler) processEvent(
  deliveryID, eventType string,
  payload map[string]any,
) *WebhookEvent {
  action := ""
  if a, ok := payload["action"].(string); ok {
    action = a
  }

  state := "unknown"
  if alert, ok := payload["alert"].(map[string]any); ok {
    if s, ok := alert["state"].(string); ok {
      state = s
    }
  }

  var alertNumber int
  if alert, ok := payload["alert"].(map[string]any); ok {
    if num, ok := alert["number"].(float64); ok {
      alertNumber = int(num)
    }
  }

  repo := "unknown"
  if r, ok := payload["repository"].(map[string]any); ok {
    if fn, ok := r["full_name"].(string); ok {
      repo = fn
    }
  }

  // Sanitize secret scanning payloads
  sanitizedPayload := payload
  if eventType == "secret_scanning_alert" {
    if alert, ok := sanitizedPayload["alert"].(map[string]any); ok {
      alert["secret"] = "[REDACTED]"
    }
  }

  return &WebhookEvent{
    DeliveryID:  deliveryID,
    EventType:   eventType,
    Action:      action,
    AlertKey:    h.computeAlertKey(eventType, payload),
    Repository:  repo,
    AlertNumber: alertNumber,
    Severity:    h.extractSeverity(eventType, payload),
    State:       state,
    Assignees:   h.extractAssignees(payload),
    ReceivedAt:  time.Now().UTC(),
    Payload:     sanitizedPayload,
  }
}
```

```go

func (h *WebhookHandler) publishToQueue(ctx context.Context, event *
    WebhookEvent) error {
  body, err := json.Marshal(event)
  if err != nil {
    return fmt.Errorf("failed to marshal event: %w", err)
  }

  messageGroupID := strings.ReplaceAll(event.Repository, "/", "-")

  _, err = h.sqsClient.SendMessage(ctx, &sqs.SendMessageInput{
    QueueUrl:               &h.config.SQSQueueURL,
    MessageBody:            ptrString(string(body)),
    MessageDeduplicationId: &event.DeliveryID,
    MessageGroupId:         &messageGroupID,
  })

  return err
}

func ptrString(s string) *string { return &s }

var allowedEvents = map[string]bool{
  "code_scanning_alert":   true,
  "dependabot_alert":      true,
  "secret_scanning_alert": true,
  "security_and_analysis": true,
}

func (h *WebhookHandler) ServeHTTP(w http.ResponseWriter, r *http.
    Request) {
  if r.Method != http.MethodPost {
    http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
    return
  }

  // Extract headers
  signature := r.Header.Get("X-Hub-Signature-256")
  eventType := r.Header.Get("X-GitHub-Event")
  deliveryID := r.Header.Get("X-GitHub-Delivery")

  // Read body
  body, err := io.ReadAll(io.LimitReader(r.Body, 5<<20)) // 5MB limit
  if err != nil {
    h.logger.Error("Failed to read body", "delivery", deliveryID, "
    error", err)
    http.Error(w, "Failed to read body", http.StatusBadRequest)
    return
  }

  // Verify signature
  if !h.verifySignature(body, signature) {
    h.logger.Warn("Invalid signature", "delivery", deliveryID)
    w.WriteHeader(http.StatusUnauthorized)
```

```
      json.NewEncoder(w).Encode(map[string]string{
        "error":   "invalid_signature",
        "message": "Webhook signature validation failed",
      })
      return
    }

    // Check event type
    if !allowedEvents[eventType] {
      h.logger.Info("Ignored event type", "delivery", deliveryID, "event"
     , eventType)
      w.WriteHeader(http.StatusOK)
      json.NewEncoder(w).Encode(map[string]string{
        "status": "ignored",
        "reason": "event_type_not_processed",
      })
      return
    }

    // Parse payload
    var payload map[string]any
    if err := json.Unmarshal(body, &payload); err != nil {
      h.logger.Error("Failed to parse payload", "delivery", deliveryID, "
     error", err)
      http.Error(w, "Invalid JSON payload", http.StatusBadRequest)
      return
    }

    // Process and queue
    event := h.processEvent(deliveryID, eventType, payload)

    ctx, cancel := context.WithTimeout(r.Context(), 5*time.Second)
    defer cancel()

    if err := h.publishToQueue(ctx, event); err != nil {
      h.logger.Error("Failed to queue event",
        "delivery", deliveryID,
        "alert_key", event.AlertKey,
        "error", err,
      )
      http.Error(w, "Failed to queue event", http.
     StatusInternalServerError)
      return
    }

    h.logger.Info("Event queued",
      "delivery", deliveryID,
      "alert_key", event.AlertKey,
      "action", event.Action,
    )

    w.WriteHeader(http.StatusAccepted)
    json.NewEncoder(w).Encode(map[string]string{
      "status":      "accepted",
```

```go
			"delivery_id": deliveryID,
			"alert_key":   event.AlertKey,
		})
}

func main() {
	cfg := &Config{
		Port:          getEnv("PORT", "8080"),
		WebhookSecret: os.Getenv("GITHUB_WEBHOOK_SECRET"),
		SQSQueueURL:   os.Getenv("SQS_QUEUE_URL"),
	}

	handler, err := NewWebhookHandler(cfg)
	if err != nil {
		slog.Error("Failed to create handler", "error", err)
		os.Exit(1)
	}

	mux := http.NewServeMux()
	mux.Handle("/webhook/github", handler)
	mux.HandleFunc("/health", func(w http.ResponseWriter, r *http.Request
	 ) {
		json.NewEncoder(w).Encode(map[string]string{
			"status":    "healthy",
			"timestamp": time.Now().UTC().Format(time.RFC3339),
		})
	})

	slog.Info("Starting server", "port", cfg.Port)
	if err := http.ListenAndServe(":"+cfg.Port, mux); err != nil {
		slog.Error("Server failed", "error", err)
		os.Exit(1)
	}
}

func getEnv(key, fallback string) string {
	if v := os.Getenv(key); v != "" {
		return v
	}
	return fallback
}
```

Listing 8: Go Webhook Receiver with Concurrent Processing

8.3 ServiceNow Scripted REST API Implementation

For deployments preferring a ServiceNow-native approach:

```javascript
// Scripted REST API Resource: POST /api/x_ghas/webhook
// Table: x_ghas_event (custom staging table)

(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/
    response) {
		'use strict';
```

```

    var startTime = new GlideDateTime();

    // Configuration
    var SECRET = gs.getProperty('x_ghas.webhook.secret');
    var MAX_PAYLOAD_SIZE = 5242880; // 5MB

    // Extract headers
    var signature = request.getHeader('X-Hub-Signature-256') || '';
    var eventType = request.getHeader('X-GitHub-Event') || '';
    var deliveryId = request.getHeader('X-GitHub-Delivery') || '';

    // Get raw body for signature verification
    var bodyStr = request.body.dataString || '';

    // Check payload size
    if (bodyStr.length > MAX_PAYLOAD_SIZE) {
        response.setStatus(413);
        response.setBody({
            error: 'payload_too_large',
            max_size: MAX_PAYLOAD_SIZE
        });
        return;
    }

    // Verify signature using Java crypto
    var expectedSignature = 'sha256=' + computeHmacSha256(SECRET, bodyStr);
    if (!constantTimeEquals(signature, expectedSignature)) {
        gs.warn('GHAS Webhook: Invalid signature for delivery ' + deliveryId);
        response.setStatus(401);
        response.setBody({
            error: 'invalid_signature',
            delivery_id: deliveryId
        });
        return;
    }

    // Validate event type
    var allowedEvents = [
        'code_scanning_alert',
        'dependabot_alert',
        'secret_scanning_alert',
        'security_and_analysis'
    ];

    if (allowedEvents.indexOf(eventType) === -1) {
        response.setStatus(200);
        response.setBody({
            status: 'ignored',
            reason: 'event_type_not_processed',
            event_type: eventType
        });
```

```
            return;
        }

        try {
            var payload = JSON.parse(bodyStr);

            // Compute canonical alert key
            var alertKey = computeAlertKey(eventType, payload);

            // Check for duplicate delivery (idempotency)
            var existingEvent = new GlideRecord('x_ghas_event');
            existingEvent.addQuery('u_delivery_id', deliveryId);
            existingEvent.query();

            if (existingEvent.next()) {
                response.setStatus(200);
                response.setBody({
                    status: 'duplicate',
                    delivery_id: deliveryId,
                    existing_sys_id: existingEvent.sys_id.toString()
                });
                return;
            }

            // Create staging record
            var eventRecord = new GlideRecord('x_ghas_event');
            eventRecord.initialize();
            eventRecord.u_delivery_id = deliveryId;
            eventRecord.u_event_type = eventType;
            eventRecord.u_action = payload.action || 'unknown';
            eventRecord.u_alert_key = alertKey;
            eventRecord.u_repository = getNestedValue(payload, 'repository.full_name', 'unknown');
            eventRecord.u_alert_number = getNestedValue(payload, 'alert.number', 0);
            eventRecord.u_severity = extractSeverity(eventType, payload);
            eventRecord.u_state = getNestedValue(payload, 'alert.state', 'unknown');
            eventRecord.u_assignees = JSON.stringify(extractAssignees(payload));
            eventRecord.u_processing_state = 'received';
            eventRecord.u_received_at = startTime;

            // Sanitize and store payload
            var sanitizedPayload = sanitizePayload(eventType, payload);
            eventRecord.u_payload = JSON.stringify(sanitizedPayload);

            var sysId = eventRecord.insert();

            if (!sysId) {
                throw new Error('Failed to insert event record');
            }

            // Queue for async processing via event
            gs.eventQueue('x_ghas.event.received', eventRecord, deliveryId, eventType);

            // Calculate processing time
            var endTime = new GlideDateTime();
            var duration = GlideDateTime.subtract(startTime, endTime).getNumericValue();

            gs.info('GHAS Webhook: Queued event ' + deliveryId +
                    ' (' + alertKey + ') in ' + duration + 'ms');

            response.setStatus(202);
            response.setBody({
                status: 'accepted',
                delivery_id: deliveryId,
                alert_key: alertKey,
                sys_id: sysId,
                processing_time_ms: duration
            });

        } catch (e) {
            gs.error('GHAS Webhook: Processing error for ' + deliveryId + ': ' + e.message);
            response.setStatus(500);
            response.setBody({
                error: 'processing_failed',
                delivery_id: deliveryId,
                message: e.message
            });
        }

        // ========== Helper Functions ==========

        function computeHmacSha256(key, data) {
            var Mac = Packages.javax.crypto.Mac;
            var SecretKeySpec = Packages.javax.crypto.spec.SecretKeySpec;
            var StandardCharsets = Packages.java.nio.charset.StandardCharsets;

            var mac = Mac.getInstance('HmacSHA256');
            var keyBytes = new java.lang.String(key).getBytes(StandardCharsets.UTF_8);
            var keySpec = new SecretKeySpec(keyBytes, 'HmacSHA256');
            mac.init(keySpec);

            var dataBytes = new java.lang.String(data).getBytes(StandardCharsets.UTF_8);
            var hashBytes = mac.doFinal(dataBytes);

            var sb = new java.lang.StringBuilder();
            for (var i = 0; i < hashBytes.length; i++) {
                sb.append(java.lang.String.format('%02x', hashBytes[i] & 0xff));
            }
            return sb.toString();
        }

        function constantTimeEquals(a, b) {
            if (!a || !b || a.length !== b.length) return false;
            var result = 0;
            for (var i = 0; i < a.length; i++) {
                result |= a.charCodeAt(i) ^ b.charCodeAt(i);
            }
            return result === 0;
        }

        function computeAlertKey(eventType, payload) {
            var repo = getNestedValue(payload, 'repository.full_name', 'unknown');
            var alertNumber = getNestedValue(payload, 'alert.number', 0);

            var typeMap = {
                'code_scanning_alert': 'code_scanning',
                'dependabot_alert': 'dependabot',
                'secret_scanning_alert': 'secret_scanning'
            };

            var alertType = typeMap[eventType] || eventType;
            return 'github:' + repo + ':' + alertType + ':' + alertNumber;
        }

        function extractSeverity(eventType, payload) {
            var alert = payload.alert || {};

            switch (eventType) {
                case 'code_scanning_alert':
                    var rule = alert.rule || {};
                    return rule.severity || rule.security_severity_level || '';
                case 'dependabot_alert':
                    var advisory = alert.security_advisory || {};
                    return advisory.severity || '';
                case 'secret_scanning_alert':
                    return 'critical';
                default:
                    return '';
            }
        }

        function extractAssignees(payload) {
            var assignees = [];
            var alertAssignees = getNestedValue(payload, 'alert.assignees', []);

            for (var i = 0; i < alertAssignees.length; i++) {
                if (alertAssignees[i] && alertAssignees[i].login) {
                    assignees.push(alertAssignees[i].login);
                }
            }
            return assignees;
        }

        function sanitizePayload(eventType, payload) {
            if (eventType === 'secret_scanning_alert') {
                var sanitized = JSON.parse(JSON.stringify(payload));
                if (sanitized.alert) {
                    sanitized.alert.secret = '[REDACTED]';
                }
                return sanitized;
            }
            return payload;
        }

        function getNestedValue(obj, path, defaultValue) {
            var parts = path.split('.');
            var current = obj;

            for (var i = 0; i < parts.length; i++) {
                if (current === null || current === undefined) {
                    return defaultValue;
                }
                current = current[parts[i]];
            }

            return (current !== null && current !== undefined) ? current :
        defaultValue;
        }

    })(request, response);
```

Listing 9: ServiceNow Scripted REST API Resource

9 Implementation: Reconciliation Worker

9.1 Python Reconciliation Service

Production-grade Python implementation with comprehensive API support:

```
#!/usr/bin/env python3
"""
GitHub Security Alerts Reconciliation Worker

Synchronizes GitHub security alerts with ServiceNow, supporting:
- Code scanning, secret scanning, and Dependabot alerts
- Alert assignees (new 2025 feature)
- Security campaigns
- Bidirectional state synchronization
"""

import os
import sys
```

```python
import json
import logging
import hashlib
from datetime import datetime, timedelta, timezone
from typing import Any, Iterator, Optional
from dataclasses import dataclass, asdict
from enum import Enum

import requests
from requests.adapters import HTTPAdapter
from urllib3.util.retry import Retry

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)


class AlertType(Enum):
    CODE_SCANNING = "code_scanning"
    SECRET_SCANNING = "secret_scanning"
    DEPENDABOT = "dependabot"


@dataclass
class GitHubConfig:
    """GitHub API configuration."""
    token: str
    api_base: str = "https://api.github.com"
    api_version: str = "2022-11-28"

    @property
    def headers(self) -> dict[str, str]:
        return {
            "Accept": "application/vnd.github+json",
            "Authorization": f"Bearer {self.token}",
            "X-GitHub-Api-Version": self.api_version,
        }


@dataclass
class ServiceNowConfig:
    """ServiceNow API configuration."""
    instance_url: str
    username: str
    password: str
    ingest_endpoint: str = "/api/x_ghas/reconciliation"

    @property
    def full_url(self) -> str:
        return f"{self.instance_url.rstrip('/')}{self.ingest_endpoint}"

```

```python

@dataclass
class NormalizedAlert:
    """Normalized alert structure for ServiceNow ingestion."""
    provider: str
    alert_type: str
    organization: str
    repository: str
    alert_number: int
    canonical_key: str
    state: str
    severity: Optional[str]
    created_at: str
    updated_at: str
    html_url: str
    rule_id: Optional[str]
    rule_description: Optional[str]
    cwe_ids: list[str]
    assignees: list[str]
    dismissed_by: Optional[str]
    dismissed_reason: Optional[str]
    fixed_at: Optional[str]
    raw_alert: dict[str, Any]


class GitHubClient:
    """GitHub API client with retry logic and pagination support."""

    def __init__(self, config: GitHubConfig):
        self.config = config
        self.session = self._create_session()

    def _create_session(self) -> requests.Session:
        session = requests.Session()

        # Configure retries with exponential backoff
        retry_strategy = Retry(
            total=5,
            backoff_factor=1,
            status_forcelist=[429, 500, 502, 503, 504],
            allowed_methods=["GET", "PATCH"],
            respect_retry_after_header=True,
        )

        adapter = HTTPAdapter(max_retries=retry_strategy)
        session.mount("https://", adapter)
        session.headers.update(self.config.headers)

        return session

    def _paginate(
        self,
        url: str,
        params: Optional[dict[str, Any]] = None
    ) -> Iterator[dict[str, Any]]:
        """Paginate through GitHub API results."""
        params = params or {}
        params.setdefault("per_page", 100)

        while url:
            response = self.session.get(url, params=params, timeout=30)
            response.raise_for_status()

            # Check rate limits
            remaining = int(response.headers.get("X-RateLimit-Remaining", 0))
            if remaining < 100:
                reset_time = int(response.headers.get("X-RateLimit-Reset", 0))
                logger.warning(
                    f"Rate limit low: {remaining} remaining, "
                    f"resets at {datetime.fromtimestamp(reset_time)}"
                )

            data = response.json()
            if isinstance(data, list):
                yield from data
            else:
                yield data

            # Get next page URL from Link header
            url = None
            params = None  # Only apply params on first request

            link_header = response.headers.get("Link", "")
            for link in link_header.split(","):
                if 'rel="next"' in link:
                    url = link[link.find("<") + 1:link.find(">")]
                    break

    def list_code_scanning_alerts(
        self,
        owner: str,
        repo: str,
        state: str = "open",
        since: Optional[datetime] = None,
    ) -> Iterator[dict[str, Any]]:
        """List code scanning alerts for a repository."""
        url = f"{self.config.api_base}/repos/{owner}/{repo}/code-scanning/alerts"
        params = {"state": state}

        if since:
            # Filter by tool_name and created_at if needed
            pass

        yield from self._paginate(url, params)

    def list_secret_scanning_alerts(
        self,
        owner: str,
        repo: str,
        state: str = "open",
    ) -> Iterator[dict[str, Any]]:
        """List secret scanning alerts for a repository."""
        url = f"{self.config.api_base}/repos/{owner}/{repo}/secret-scanning/alerts"
        params = {"state": state}
        yield from self._paginate(url, params)

    def list_dependabot_alerts(
        self,
        owner: str,
        repo: str,
        state: str = "open",
    ) -> Iterator[dict[str, Any]]:
        """List Dependabot alerts for a repository."""
        url = f"{self.config.api_base}/repos/{owner}/{repo}/dependabot/alerts"
        params = {"state": state}
        yield from self._paginate(url, params)

    def list_org_repos(
        self,
        org: str,
        repo_type: str = "all",
    ) -> Iterator[dict[str, Any]]:
        """List repositories in an organization."""
        url = f"{self.config.api_base}/orgs/{org}/repos"
        params = {"type": repo_type, "sort": "updated", "direction": "desc"}
        yield from self._paginate(url, params)

    def update_alert_assignees(
        self,
        owner: str,
        repo: str,
        alert_type: AlertType,
        alert_number: int,
        assignees: list[str],
    ) -> dict[str, Any]:
        """Update alert assignees (bidirectional sync)."""
        if alert_type == AlertType.CODE_SCANNING:
            url = f"{self.config.api_base}/repos/{owner}/{repo}/code-scanning/alerts/{alert_number}"
        elif alert_type == AlertType.SECRET_SCANNING:
            url = f"{self.config.api_base}/repos/{owner}/{repo}/secret-scanning/alerts/{alert_number}"
        else:
            raise ValueError(f"Assignees not supported for {alert_type}")

        response = self.session.patch(
            url,
            json={"assignees": assignees},
            timeout=30,
        )
        response.raise_for_status()
        return response.json()


class AlertNormalizer:
    """Normalize GitHub alerts to common format."""

    @staticmethod
    def compute_canonical_key(
        org: str,
        repo: str,
        alert_type: AlertType,
        alert_number: int,
    ) -> str:
        """Compute deterministic canonical key."""
        return f"github:{org}/{repo}:{alert_type.value}:{alert_number}"

    @classmethod
    def normalize_code_scanning(
        cls,
        org: str,
        repo: str,
        alert: dict[str, Any],
    ) -> NormalizedAlert:
        """Normalize code scanning alert."""
        rule = alert.get("rule", {})

        # Extract CWE IDs from tags
        cwe_ids = [
            tag.replace("external/cwe/", "")
            for tag in rule.get("tags", [])
            if tag.startswith("external/cwe/")
        ]

        # Extract assignees
        assignees = [a.get("login", "") for a in alert.get("assignees",
    [])]

        return NormalizedAlert(
            provider="github",
            alert_type=AlertType.CODE_SCANNING.value,
            organization=org,
            repository=repo,
            alert_number=alert.get("number", 0),
            canonical_key=cls.compute_canonical_key(
                org, repo, AlertType.CODE_SCANNING, alert.get("number",
    0)
            ),
            state=alert.get("state", "unknown"),
            severity=rule.get("severity") or rule.get("security_severity_level"),
            created_at=alert.get("created_at", ""),
            updated_at=alert.get("updated_at", ""),
            html_url=alert.get("html_url", ""),
            rule_id=rule.get("id"),
            rule_description=rule.get("description"),
            cwe_ids=cwe_ids,
            assignees=assignees,
            dismissed_by=alert.get("dismissed_by", {}).get("login") if alert.get("dismissed_by") else None,
            dismissed_reason=alert.get("dismissed_reason"),
            fixed_at=alert.get("fixed_at"),
            raw_alert=alert,
        )

    @classmethod
    def normalize_secret_scanning(
        cls,
        org: str,
        repo: str,
        alert: dict[str, Any],
    ) -> NormalizedAlert:
        """Normalize secret scanning alert.

        CRITICAL: Never store actual secret values.
        """
        # Redact secret value
        sanitized_alert = {**alert}
        if "secret" in sanitized_alert:
            sanitized_alert["secret"] = "[REDACTED]"

        assignees = [a.get("login", "") for a in alert.get("assignees", [])]

        return NormalizedAlert(
            provider="github",
            alert_type=AlertType.SECRET_SCANNING.value,
            organization=org,
            repository=repo,
            alert_number=alert.get("number", 0),
            canonical_key=cls.compute_canonical_key(
                org, repo, AlertType.SECRET_SCANNING, alert.get("number", 0)
            ),
            state=alert.get("state", "unknown"),
            severity="critical",  # Secret exposure is always critical
            created_at=alert.get("created_at", ""),
            updated_at=alert.get("updated_at", ""),
            html_url=alert.get("html_url", ""),
            rule_id=alert.get("secret_type"),
            rule_description=f"Exposed {alert.get('secret_type_display_name', 'secret')}",
            cwe_ids=["CWE-798"],  # Hard-coded credentials
```

```
                assignees=assignees,
                dismissed_by=alert.get("resolved_by", {}).get("login") if
    alert.get("resolved_by") else None,
                dismissed_reason=alert.get("resolution"),
                fixed_at=alert.get("resolved_at"),
                raw_alert=sanitized_alert,
            )

    @classmethod
    def normalize_dependabot(
            cls,
            org: str,
            repo: str,
            alert: dict[str, Any],
    ) -> NormalizedAlert:
            """Normalize Dependabot alert."""
            advisory = alert.get("security_advisory", {})
            vulnerability = alert.get("security_vulnerability", {})

            # Extract CWE IDs
            cwe_ids = [
                cwe.get("cwe_id", "")
                for cwe in advisory.get("cwes", [])
            ]

            return NormalizedAlert(
                provider="github",
                alert_type=AlertType.DEPENDABOT.value,
                organization=org,
                repository=repo,
                alert_number=alert.get("number", 0),
                canonical_key=cls.compute_canonical_key(
                    org, repo, AlertType.DEPENDABOT, alert.get("number", 0)
                ),
                state=alert.get("state", "unknown"),
                severity=advisory.get("severity") or vulnerability.get("
    severity"),
                created_at=alert.get("created_at", ""),
                updated_at=alert.get("updated_at", ""),
                html_url=alert.get("html_url", ""),
                rule_id=advisory.get("ghsa_id") or advisory.get("cve_id"),
                rule_description=advisory.get("summary"),
                cwe_ids=cwe_ids,
                assignees=[],  # Dependabot doesn't support assignees
                dismissed_by=alert.get("dismissed_by", {}).get("login") if
    alert.get("dismissed_by") else None,
                dismissed_reason=alert.get("dismissed_reason"),
                fixed_at=alert.get("fixed_at"),
                raw_alert=alert,
            )


class ServiceNowClient:
    """ServiceNow API client."""

    def __init__(self, config: ServiceNowConfig):
            self.config = config
            self.session = self._create_session()

    def _create_session(self) -> requests.Session:
            session = requests.Session()
            session.auth = (self.config.username, self.config.password)
            session.headers.update({
                "Content-Type": "application/json",
                "Accept": "application/json",
            })
            return session

    def upsert_alert(self, alert: NormalizedAlert) -> dict[str, Any]:
            """Upsert alert to ServiceNow."""
            payload = asdict(alert)

            # Don't send full raw alert to reduce payload size
            payload["raw_alert_hash"] = hashlib.sha256(
                json.dumps(alert.raw_alert, sort_keys=True).encode()
            ).hexdigest()
            del payload["raw_alert"]

            response = self.session.post(
                self.config.full_url,
                json=payload,
                timeout=30,
            )
            response.raise_for_status()
            return response.json()

    def upsert_alerts_batch(
            self,
            alerts: list[NormalizedAlert],
            batch_size: int = 100,
    ) -> dict[str, Any]:
            """Batch upsert alerts."""
            results = {"success": 0, "failed": 0, "errors": []}

            for i in range(0, len(alerts), batch_size):
                batch = alerts[i:i + batch_size]

                payloads = []
                for alert in batch:
                    payload = asdict(alert)
                    payload["raw_alert_hash"] = hashlib.sha256(
                        json.dumps(alert.raw_alert, sort_keys=True).encode
    ()
                    ).hexdigest()
                    del payload["raw_alert"]
                    payloads.append(payload)

                try:
                    response = self.session.post(
                        f"{self.config.full_url}/batch",
                        json={"alerts": payloads},
                        timeout=60,
                    )
                    response.raise_for_status()
                    result = response.json()
                    results["success"] += result.get("success", 0)
                    results["failed"] += result.get("failed", 0)
                except Exception as e:
                    logger.error(f"Batch upsert failed: {e}")
                    results["failed"] += len(batch)
                    results["errors"].append(str(e))

            return results


class ReconciliationWorker:
    """Main reconciliation orchestrator."""

    def __init__(
            self,
            github_client: GitHubClient,
            servicenow_client: ServiceNowClient,
    ):
            self.github = github_client
            self.servicenow = servicenow_client
            self.normalizer = AlertNormalizer()

    def reconcile_repository(
            self,
            org: str,
            repo: str,
            alert_types: Optional[list[AlertType]] = None,
            states: Optional[list[str]] = None,
    ) -> dict[str, Any]:
            """Reconcile alerts for a single repository."""
            alert_types = alert_types or list(AlertType)
            states = states or ["open", "dismissed", "fixed"]

            results = {
                "repository": f"{org}/{repo}",
                "timestamp": datetime.now(timezone.utc).isoformat(),
                "alerts_processed": 0,
                "by_type": {},
            }

            all_alerts: list[NormalizedAlert] = []

            for alert_type in alert_types:
                type_count = 0

                for state in states:
                    try:
```

```python
                    if alert_type == AlertType.CODE_SCANNING:
                        alerts = self.github.list_code_scanning_alerts(org, repo, state)
                        for alert in alerts:
                            normalized = self.normalizer.normalize_code_scanning(org, repo, alert)
                            all_alerts.append(normalized)
                            type_count += 1

                    elif alert_type == AlertType.SECRET_SCANNING:
                        alerts = self.github.list_secret_scanning_alerts(org, repo, state)
                        for alert in alerts:
                            normalized = self.normalizer.normalize_secret_scanning(org, repo, alert)
                            all_alerts.append(normalized)
                            type_count += 1

                    elif alert_type == AlertType.DEPENDABOT:
                        alerts = self.github.list_dependabot_alerts(org, repo, state)
                        for alert in alerts:
                            normalized = self.normalizer.normalize_dependabot(org, repo, alert)
                            all_alerts.append(normalized)
                            type_count += 1

                except requests.HTTPError as e:
                    if e.response.status_code == 404:
                        logger.info(f"{alert_type.value} not enabled for {org}/{repo}")
                    else:
                        logger.error(f"Error fetching {alert_type.value} for {org}/{repo}: {e}")

            results["by_type"][alert_type.value] = type_count

        # Batch upsert to ServiceNow
        if all_alerts:
            sync_result = self.servicenow.upsert_alerts_batch(all_alerts)
            results["sync_result"] = sync_result

        results["alerts_processed"] = len(all_alerts)
        return results

    def reconcile_organization(
        self,
        org: str,
        alert_types: Optional[list[AlertType]] = None,
        max_repos: Optional[int] = None,
    ) -> dict[str, Any]:
        """Reconcile alerts for all repositories in an organization."""
        results = {
            "organization": org,
            "timestamp": datetime.now(timezone.utc).isoformat(),
            "repositories_processed": 0,
            "total_alerts": 0,
            "repositories": [],
        }

        repo_count = 0
        for repo in self.github.list_org_repos(org):
            if max_repos and repo_count >= max_repos:
                break

            repo_name = repo.get("name", "")
            logger.info(f"Reconciling {org}/{repo_name}")

            try:
                repo_result = self.reconcile_repository(
                    org, repo_name, alert_types
                )
                results["repositories"].append(repo_result)
                results["total_alerts"] += repo_result["alerts_processed"]
            except Exception as e:
                logger.error(f"Failed to reconcile {org}/{repo_name}: {e}")
                results["repositories"].append({
                    "repository": f"{org}/{repo_name}",
                    "error": str(e),
                })

            repo_count += 1

        results["repositories_processed"] = repo_count
        return results


def main():
    """Main entry point."""
    # Load configuration from environment
    github_config = GitHubConfig(
        token=os.environ["GITHUB_TOKEN"],
    )

    servicenow_config = ServiceNowConfig(
        instance_url=os.environ["SERVICENOW_INSTANCE_URL"],
        username=os.environ["SERVICENOW_USERNAME"],
        password=os.environ["SERVICENOW_PASSWORD"],
    )

    # Initialize clients
    github_client = GitHubClient(github_config)
    servicenow_client = ServiceNowClient(servicenow_config)

    # Create worker
```

```python
    worker = ReconciliationWorker(github_client, servicenow_client)

    # Run reconciliation
    org = os.environ.get("GITHUB_ORG", "")
    repo = os.environ.get("GITHUB_REPO", "")

    if repo:
        # Single repository mode
        result = worker.reconcile_repository(org, repo)
    else:
        # Organization mode
        result = worker.reconcile_organization(org)

    # Output results
    print(json.dumps(result, indent=2, default=str))

    # Exit with error if failures occurred
    if result.get("sync_result", {}).get("failed", 0) > 0:
        sys.exit(1)


if __name__ == "__main__":
    main()
```

Listing 10: Python Reconciliation Worker with Full Feature Support

10 Implementation: GitHub Actions Integration

10.1 Security Results to ServiceNow DevOps

Leverage the official ServiceNow DevOps Security Results action:

```yaml
name: Security Scanning with ServiceNow Integration

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

permissions:
  contents: read
  security-events: write
  actions: read

jobs:
  # ============ CodeQL Analysis ============
  codeql-analysis:
    name: CodeQL SAST Scan
    runs-on: ubuntu-latest

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

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

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

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

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

  # ============ Dependency Review ============
  dependency-review:
    name: Dependency Review
    runs-on: ubuntu-latest
    if: github.event_name == 'pull_request'

    steps:
      - name: Checkout repository
        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

  # ============ Container Scanning ============
  container-scan:
    name: Container Security Scan
    runs-on: ubuntu-latest
    if: github.event_name == 'push'

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

      - name: Build container image
        run: docker build -t app:${{ github.sha }} .

      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: 'app:${{ github.sha }}'
          format: 'sarif'
          output: 'trivy-results.sarif'
          severity: 'CRITICAL,HIGH,MEDIUM'

      - name: Upload Trivy scan results to GitHub Security
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: 'trivy-results.sarif'
          category: 'container-scanning'

  # ============ ServiceNow Integration ============
  servicenow-sync:
    name: Sync to ServiceNow DevOps
    runs-on: ubuntu-latest
    needs: [codeql-analysis]
    if: always() && github.event_name == 'push'

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

      - name: Register Security Results with ServiceNow
        uses: ServiceNow/servicenow-devops-security-result@v3.1.0
        with:
          devops-integration-token: ${{ secrets.SN_DEVOPS_INTEGRATION_TOKEN }}
          instance-url: ${{ secrets.SN_INSTANCE_URL }}
          tool-id: ${{ secrets.SN_ORCHESTRATION_TOOL_ID }}
          context-github: ${{ toJSON(github) }}
          job-name: 'ServiceNow Security Results'

      - name: Notify on critical findings
        if: failure()
        uses: slackapi/slack-github-action@v1
        with:
          payload: |
            {
              "text": ":rotating_light: Critical security findings detected in ${{ github.repository }}",
              "blocks": [
                {
                  "type": "section",
                  "text": {
                    "type": "mrkdwn",
                    "text": "*Security Alert*\nCritical findings detected in `${{ github.repository }}`\n<${{ github.server_url }}/${{ github.repository }}/security|View Security Dashboard>"
                  }
                }
              ]
            }
        env:
          SLACK_WEBHOOK_URL: ${{ secrets.SLACK_SECURITY_WEBHOOK }}

  # ============ SBOM Generation ============
  sbom-generation:
    name: Generate and Upload SBOM
    runs-on: ubuntu-latest
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'

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

      - name: Generate SBOM
        uses: anchore/sbom-action@v0
        with:
          format: spdx-json
          output-file: sbom.spdx.json

      - name: Upload SBOM to ServiceNow
        uses: ServiceNow/vulnerability-response@v2.0.1
        with:
          snSbomUser: ${{ secrets.SN_SBOM_USERNAME }}
          snSbomPassword: ${{ secrets.SN_SBOM_PASSWORD }}
          snInstanceUrl: ${{ secrets.SN_INSTANCE_URL }}
          ghToken: ${{ secrets.GITHUB_TOKEN }}
          ghAccountOwner: ${{ github.repository_owner }}
          repository: ${{ github.event.repository.name }}
          provider: 'repository'
          path: 'sbom.spdx.json'
```

Listing 11: GitHub Actions: Complete Security Workflow with ServiceNow Integration

11 Automation Design: Playbooks and Workflows

11.1 Triage and Prioritization Matrix

Compute normalized priority using multiple signals:

Signal	**Weighting and Logic**
GitHub Severity	Direct mapping: critical=P1, high=P2, medium=P3, low=P4
CVSS Score	$\geq$ 9.0: P1, $\geq$ 7.0: P2, $\geq$ 4.0: P3, $<$ 4.0: P4
EPSS Score	Exploit Prediction Scoring: $>$ 0.5: escalate one level
Asset Criticality	Business-critical repo: escalate one level
Alert Type	Secret scanning: always P1 or P2 (credential exposure)
Push Protection Bypass	If `push_protection_bypassed`: true, immediate escalation
Known Exploited	If in CISA KEV catalog: immediate P1

Autofix Available	Copilot Autofix available: may accelerate SLA

11.2 Assignment Resolution Strategy

Ownership resolution follows this precedence:

1. **CMDB/Application Registry**: Map repository to owning application/service via ServiceNow CMDB
2. **GitHub CODEOWNERS**: Parse `.github/CODEOWNERS` file for path-based ownership
3. **Repository Metadata**: Use GitHub repository topics or custom properties
4. **Recent Committers**: Assign to most active contributor if other methods fail
5. **Fallback Queue**: Route to AppSec triage queue for manual assignment

11.3 SLA Configuration Matrix

Priority	**Secret**	**Code**	**Dependabot**	**Escalation**
P1 / Critical	4 hours	24 hours	48 hours	Immediate page
P2 / High	24 hours	7 days	14 days	Daily summary
P3 / Medium	7 days	30 days	60 days	Weekly report
P4 / Low	30 days	90 days	180 days	Quarterly review

11.4 Exception Management Workflow

For dismissed alerts requiring risk acceptance:

1. **Documented Reason**: Mandatory dismissal reason (false positive, won't fix, used in tests, etc.)
2. **Risk Owner Approval**: Designated risk acceptance owner must approve
3. **Expiration Date**: All exceptions expire (default: 90 days for code, 30 days for secrets)
4. **Compensating Controls**: Document mitigations for accepted risk
5. **Review Cycle**: Automated reminder for review before expiration

12 Reliability Engineering

12.1 Idempotency Implementation

- **Delivery ID**: Store `X-GitHub-Delivery` UUID; reject duplicates
- **Canonical Key**: Use as record identity for upsert semantics
- **State Machine**: Only allow valid state transitions
- **Optimistic Locking**: Use `updated_at` timestamp for conflict detection

12.2 Retry and Dead-Letter Strategy

```yaml
retry_policy:
  initial_delay_ms: 1000
  max_delay_ms: 300000  # 5 minutes
  backoff_multiplier: 2.0
  jitter_factor: 0.1
  max_attempts: 5

  # Retriable error codes
  retriable_status_codes:
    - 429  # Rate limited
    - 500  # Internal server error
    - 502  # Bad gateway
    - 503  # Service unavailable
    - 504  # Gateway timeout

dlq_policy:
  # Move to DLQ after max_attempts exhausted
  max_receive_count: 5

  # DLQ retention for investigation
  retention_days: 14

  # Alert thresholds
  alert_on_dlq_depth: 10
  alert_on_dlq_age_hours: 24

  # Replay configuration
  replay_batch_size: 100
  replay_delay_ms: 500
  require_manual_approval: true
```

Listing 12: Retry and DLQ Configuration

12.3 Observability Requirements

Key metrics to capture:

- **Ingest Latency**: GitHub webhook timestamp → ServiceNow record creation
- **Processing Success Rate**: By event type and action
- **Deduplication Hits**: Percentage of duplicate deliveries
- **Queue Depth**: Backlog size and age
- **API Rate Limit Consumption**: GitHub and ServiceNow
- **SLA Compliance**: By severity, alert type, and owning team
- **Mean Time to Remediation**: By severity and alert type
- **Autofix Acceptance Rate**: Copilot Autofix suggestions adopted

13 Security Hardening Checklist

1. **Webhook Signature Verification**
   - Enforce `X-Hub-Signature-256` validation on every request
   - Use constant-time comparison to prevent timing attacks
   - Reject requests with missing or invalid signatures
2. **Authentication and Authorization**
   - Use GitHub App tokens (not PATs) for API access
   - Implement least-privilege permissions
   - Rotate tokens regularly; automate rotation where possible
   - Use OAuth 2.0 or mutual TLS for ServiceNow authentication
3. **Secret Scanning Data Handling**
   - **CRITICAL**: Never log or store actual secret values
   - Redact secrets in all stored payloads
   - Retain only remediation-relevant metadata
   - Implement data retention policies for alert data
4. **Network Security**
   - Deploy webhook receivers behind WAF
   - Implement rate limiting (per-IP and per-organization)
   - Consider IP allow-listing using GitHub's Meta API
   - Use TLS 1.3 for all connections
5. **ServiceNow Security**
   - Restrict Scripted REST API with ACLs
   - Use dedicated integration user with minimal permissions
   - Enable audit logging for all security operations
   - Implement input validation and sanitization
6. **Change Management**
   - Version-control all mapping logic and workflow rules
   - Use staged rollout with canary repositories
   - Maintain rollback capability for configuration changes
   - Document all integration touchpoints

14 Rollout Plan

14.1 Phase 0: Prerequisites (Week 1–2)

- Confirm GitHub Secret Protection and/or Code Security is enabled for target org/repos
- Install ServiceNow GitHub Application Vulnerability Integration
- Confirm SecOps licenses and roles are provisioned
- Define ownership mapping source of truth (CMDB vs CODEOWNERS)
- Establish SLA policies and escalation procedures
- Create GitHub App with required permissions

14.2 Phase 1: Baseline Import (Week 3–4)

- Enable official integration ingestion; verify records land correctly
- Validate severity mapping and deduplication logic
- Configure CWE/CVE enrichment
- Stand up initial dashboards and reports
- Establish baseline metrics for comparison

14.3 Phase 2: Webhook Event Path (Week 5–8)

- Deploy webhook receiver (external recommended)
- Implement signature verification and staging table
- Configure durable queue with DLQ
- Build Flow Designer workflows: assignment + SLA + notifications
- Test with canary repositories (non-critical)
- Monitor latency and error rates

14.4 Phase 3: Reconciliation and Advanced Features (Week 9–12)

- Deploy reconciliation poller with API rate limit awareness
- Implement drift detection: GitHub state vs ServiceNow state
- Add assignee synchronization (bidirectional if needed)
- Configure security campaign integration
- Add governance gates: false positive approvals, risk acceptance expiry
- Implement Copilot Autofix tracking and metrics

14.5 Phase 4: Production Hardening (Week 13–16)

- Performance testing under load (webhook storms)
- Security review and penetration testing
- Documentation and runbook completion
- Training for SecOps and development teams
- Full production rollout across organization
- Establish operational review cadence

15 Appendix A: Webhook Payload Examples

15.1 Code Scanning Alert Created (with Assignees)

```
{
  "action": "created",
  "alert": {
    "number": 42,
    "created_at": "2026-01-14T10:30:00Z",
    "updated_at": "2026-01-14T10:30:00Z",
    "url": "https://api.github.com/repos/acme/api/code-scanning/alerts
   /42",
    "html_url": "https://github.com/acme/api/security/code-scanning/42"
   ,
    "state": "open",
    "fixed_at": null,
    "dismissed_by": null,
    "dismissed_at": null,
    "dismissed_reason": null,
    "dismissed_comment": null,
    "assignees": [
      {
        "login": "jsmith",
        "id": 12345,
        "type": "User"
      }
    ],
    "rule": {
      "id": "js/sql-injection",
      "severity": "error",
      "security_severity_level": "critical",
      "description": "SQL injection vulnerability",
      "name": "SQL Injection",
      "tags": ["security", "external/cwe/cwe-089"],
      "help": {
        "text": "Use parameterized queries..."
      }
    },
    "tool": {
      "name": "CodeQL",
      "guid": null,
      "version": "2.23.8"
    },
    "most_recent_instance": {
      "ref": "refs/heads/main",
      "state": "open",
      "commit_sha": "abc123def456",
      "location": {
        "path": "src/db/queries.js",
        "start_line": 42,
        "end_line": 42,
        "start_column": 10,
        "end_column": 55
      },
      "message": {
        "text": "User input flows to SQL query without sanitization"
      }
    }
  },
  "ref": "refs/heads/main",
  "commit_oid": "abc123def456",
  "repository": {
    "id": 987654,
    "name": "api",
    "full_name": "acme/api",
    "private": true,
    "default_branch": "main"
  },
  "organization": {
    "login": "acme",
    "id": 111222
  },
  "sender": {
    "login": "github-actions[bot]",
    "type": "Bot"
  }
}
```

Listing 13: Code Scanning Alert Webhook Payload

15.2 Secret Scanning Alert with Push Protection Bypass

```
{
  "action": "created",
  "alert": {
    "number": 15,
    "created_at": "2026-01-14T11:00:00Z",
    "updated_at": "2026-01-14T11:00:00Z",
    "url": "https://api.github.com/repos/acme/api/secret-scanning/alerts/15",
    "html_url": "https://github.com/acme/api/security/secret-scanning/15",
```

```json
    "locations_url": "https://api.github.com/repos/acme/api/secret-scanning/alerts/15/locations",
    "state": "open",
    "secret_type": "aws_access_key_id",
    "secret_type_display_name": "AWS Access Key ID",
    "secret": "[REDACTED - never stored]",
    "validity": "unknown",
    "push_protection_bypassed": true,
    "push_protection_bypassed_by": {
      "login": "developer",
      "id": 54321
    },
    "push_protection_bypassed_at": "2026-01-14T10:55:00Z",
    "resolution": null,
    "resolved_by": null,
    "resolved_at": null,
    "resolution_comment": null,
    "assignees": [
      {
        "login": "security-team",
        "id": 99999,
        "type": "Team"
      }
    ]
  },
  "repository": {
    "id": 987654,
    "name": "api",
    "full_name": "acme/api",
    "private": true
  },
  "organization": {
    "login": "acme",
    "id": 111222
  },
  "sender": {
    "login": "developer",
    "type": "User"
  }
}
```

Listing 14: Secret Scanning Alert Webhook Payload

16 Appendix B: Normalized Event Envelope

```json
{
  "delivery_id": "550e8400-e29b-41d4-a716-446655440000",
  "event_type": "code_scanning_alert",
  "action": "created",
  "provider": "github",
  "organization": "acme",
  "repository": "acme/api",
  "alert_type": "code_scanning",
  "alert_number": 42,
  "canonical_key": "github:acme/api:code_scanning:42",
  "state": "open",
  "severity": "critical",
  "priority_computed": "P1",
  "rule_id": "js/sql-injection",
  "cwe_ids": ["CWE-089"],
  "assignees": ["jsmith"],
  "html_url": "https://github.com/acme/api/security/code-scanning/42",
  "autofix_available": true,
  "received_at": "2026-01-14T10:30:05Z",
  "processing_state": "received",
  "raw_payload_ref": "attachment_sys_id_or_hash"
}
```

Listing 15: Normalized Event Envelope for ServiceNow Staging

17 Appendix C: Primary References

17.1 GitHub Documentation

- Webhook events and payloads — https://docs.github.com/en/webhooks/webhook-events-and-payloads
- Validating webhook deliveries — https://docs.github.com/en/webhooks/using-webhooks/validating-webhook-deliveries
- Auditing security alerts — https://docs.github.com/en/code-security/getting-started/auditing-security-alerts
- REST API: Code scanning — https://docs.github.com/en/rest/code-scanning/code-scanning
- REST API: Secret scanning — https://docs.github.com/en/rest/secret-scanning/secret-scanning
- REST API: Dependabot alerts — https://docs.github.com/en/rest/dependabot/alerts
- GitHub Apps — https://docs.github.com/en/apps

17.2 GitHub Changelog (2025)

- GitHub Secret Protection and Code Security — https://github.blog/changelog/2025-03-04-introducing-
- Secret scanning alert assignees GA — https://github.blog/changelog/2025-11-25-secret-scanning-aler
- Code scanning alert assignees GA — https://github.blog/changelog/2025-12-16-code-scanning-alert-a

17.3 ServiceNow Documentation

- GitHub Application Vulnerability Integration — https://store.servicenow.com/store/app/006dafe21b646a50a85b16db234bcba2
- Vulnerability Response documentation — https://www.servicenow.com/docs/bundle/security-management

- ServiceNow DevOps Security Results action — https://github.com/ServiceNow/servicenow-devops-securi
- ServiceNow Vulnerability Response action — https://github.com/ServiceNow/vulnerability-response

---

Document Version 2.0 — January 14, 2026
For questions or feedback, contact the AppSec Engineering team