GitHub Advanced Security

Real-Time API Protection Architecture

Proactive Detection · Continuous Monitoring · AI-Driven Remediation

Prepared for Cushman & Wakefield | February 2026

Executive Summary

The customer's question is the right one: **can GitHub protect every API endpoint across every repository in the organisation, in real time, before an attacker finds a weakness?** The answer is yes — but only when three GitHub capabilities work together: **GitHub Advanced Security (GHAS)** at the code layer, **GitHub Actions scheduled probes** at the runtime layer, and **the GHAS REST API** as the control plane that ties both together and enforces policy across the entire org.

This document describes the architecture, the specific APIs involved, and how to configure them so that Cushman & Wakefield achieves continuous, automated API security — from code commit to production runtime — with AI-generated fixes and human-in-the-loop approval.

The Core Principle: Shift Security Left AND Right

Traditional security tools (WAFs, API gateways like Kong, Apigee) operate at the **request layer** — they inspect traffic after the application is deployed. GitHub operates at the **code layer** and the **behaviour layer** simultaneously:

Layer	Tool	What It Catches	When
Code (static)	CodeQL + GHAS	SQL injection, BOLA patterns, hardcoded secrets in source	Every PR — blocks merge
Dependency	Dependabot + GHAS API	Known CVEs in npm/pip/maven packages	Daily — auto PR to upgrade
Secret	Secret Scanning + GHAS API	API keys, JWT secrets, connection strings committed to git	Every push — blocks in real time
Runtime (active)	GitHub Actions probes	Missing rate limits, CORS misconfig, expired tokens in prod	Every 15 minutes
Org-wide policy	Required Workflows + API	Ensures no repo can disable security checks	On every branch merge

GitHub Advanced Security API — The Control Plane

The GHAS REST API lets you **programmatically configure, query, and enforce** security policies across every repository in the organisation — without touching the GitHub UI. This is the answer to the customer's question about setting 'all settings using APIs'.

Key GHAS APIs for Real-Time Protection

API Endpoint	Purpose	Use in Automation
PATCH /orgs/{org}/advanced-security	Enable GHAS on all repos in org	Run once to activate across org
GET /orgs/{org}/code-scanning/alerts	List all open code-scanning alerts	Aggregate dashboard, SLA tracking
POST /repos/{repo}/code-scanning/sarifs	Upload custom scan results into GHAS	Our check-*.yml workflows upload findings
GET /orgs/{org}/secret-scanning/alerts	All exposed secrets across org	Real-time secret exposure monitoring
PUT /repos/{repo}/vulnerability-alerts	Enable/disable Dependabot per repo	Enforce Dependabot org-wide
GET /orgs/{org}/dependabot/alerts	All CVEs across all org repos	SLA reporting, auto-escalation
POST /repos/{repo}/branches/{branch}/protection	Set branch protection rules via API	Require security checks before merge
GET /orgs/{org}/security-advisories	Published security advisories for org	Audit trail and compliance

The critical insight: all of these APIs are callable from a GitHub Actions workflow. This means you can write a **single orchestrator workflow** that reads your current security posture, compares it to your policy baseline, identifies gaps, and opens issues or PRs to close them — completely automatically.

The 4-Layer Real-Time Protection Architecture

Layer 1 — Prevent: Code Scanning on Every PR (CodeQL)

CodeQL performs **semantic analysis** of the TypeScript/Node.js source code. Unlike grep-based tools, CodeQL understands data flow — it traces a user-supplied value from the HTTP request all the way into a SQL query or an eval() call, catching vulnerabilities that simple pattern matching misses.

What it catches automatically: SQL injection (VULN-8), path traversal, prototype pollution, ReDoS, hardcoded credentials, SSRF patterns (VULN-10), insecure deserialization.

Enforcement: Branch protection rule requires CodeQL to pass before any PR can be merged. Set via the branch protection API — no human can bypass it.

Required workflow snippet (codeql.yml):

```
on: [push, pull_request]
jobs:
  analyze:
    uses: github/codeql-action/analyze@v3
    with: { languages: javascript-typescript }
```

Layer 2 — Detect: Runtime API Probes Every 15 Minutes

The ten check-*.yml workflows we built for this demo represent this layer. They run on a schedule: cron: '*/15 * * * *' and make **real HTTP requests** to the live API, probing for the exact vulnerabilities that code analysis cannot

see — missing rate limits, CORS wildcards, expired JWT acceptance, business logic bypasses.

Key upgrade — SARIF upload: Every probe result is formatted as SARIF (Static Analysis Results Interchange Format) and uploaded to GHAS via POST /repos/{repo}/code-scanning/sarifs. This makes runtime findings appear in the Security tab alongside CodeQL findings — a single unified view.

SARIF upload step (added to every check-*.yml):

```
- name: Upload results to GHAS Code Scanning
  uses: github/codeql-action/upload-sarif@v3
  with:
    sarif_file: results.sarif
    category: runtime-api-probe
```

Layer 3 — Respond: AI Recommendations with Human-in-the-Loop Approval

When a probe confirms a vulnerability, the workflow creates a GitHub Issue with: (1) the exact HTTP evidence, (2) the AI-generated code fix from Copilot, (3) two label options — apply-fix or risk-accepted. A separate **apply-fix.yml** workflow watches for the apply-fix label and automatically opens a PR with the patched code, which Copilot then reviews.

This creates the full feedback loop: **Detect** → **Recommend** → **Human approves** → **Copilot applies** → **Actions verifies** → **Merge**. No vulnerability lives longer than one approval cycle.

Layer 4 — Enforce: Required Workflows + Rulesets Across the Org

Required Workflows (set via the GHAS API) force every repository in the organisation to run a specified security workflow before merging — even if the repo owner tries to skip it. This is how you scale from one repo to 500 repos with a single API call:

```
POST /orgs/{org}/rulesets
{
  "name": "API Security Required",
  "enforcement": "active",
  "rules": [{ "type": "required_workflows",
    "parameters": { "required_workflows": [
      { "repository_id": <security-repo-id>,
        "ref": "refs/heads/main",
        "path": ".github/workflows/check-bola.yml" }
    ]}}
  ]
}
```

Full Architecture — Real-Time Protection Flow

Stage	Trigger	GitHub Capability	Output
1. Code written	Developer pushes	CodeQL (GHAS)	PR blocked if vuln found
2. Secret check	Every git push	Secret Scanning (GHAS)	Commit blocked, owner alerted

3. Dependency	Daily / on push	Dependabot (GHAS API)	Auto-PR to upgrade package
4. Runtime probe	Every 15 min (cron)	GitHub Actions + custom scripts	SARIF → Security tab + Issue
5. AI analysis	After probe detects	Copilot + GitHub Models API	Fix recommendation in Issue
6. Human review	Issue label added	apply-fix label watcher	Copilot opens fix PR
7. Verification	PR opened	All check-*.yml re-run	Confirms vulnerability resolved
8. Org enforcement	Any branch merge	Required Workflows + Rulesets	Org-wide policy enforced

AI-Powered Anomaly Detection: Beyond Static Rules

Static rules (block requests > X/min) are blunt instruments. GitHub's AI layer adds **behavioural understanding**:

AI Capability	How GitHub Actions Implements It	Business Value
Traffic baseline modelling	24h historical data, Z-score anomaly detection per endpoint	Set limits based on real usage — not guesswork
Attacker attribution	Correlate anomalous requests → single user/IP	Identify the actor, not just the pattern
Endpoint risk scoring	Weighted heatmap: POST /bids vs GET /health	Apply strict limits only where needed
Predictive threshold setting	Mean + 2σ from baseline → recommended limit	Eliminate false positives on legitimate peaks
Cross-repo pattern matching	GitHub Security Overview API aggregates org-wide	See if same attack is hitting multiple services
Copilot code fix generation	Copilot reads the flagged line + OWASP context	Exact patch generated — developer just approves

Real-Time Protection: GitHub vs. API Gateway (Kong/Azure APIM)

A common question is whether GitHub can replace a dedicated API gateway for real-time protection. The answer: they are complementary. The right architecture uses both.

Capability	Kong / Azure APIM	GitHub Actions + GHAS
Per-request enforcement	■ Sub-millisecond, inline	■ Not in request path
Rate limiting in prod	■ Native, stateful	■■ Recommends code-level fix
Static code vulnerabilities	■ Cannot see source code	■ CodeQL catches before deploy
Secret exposure detection	■ Not applicable	■ Real-time on every push
Business logic flaws	■ Cannot understand intent	■ Workflow probes test behaviour
Dependency CVEs	■ Not applicable	■ Dependabot + GHAS
Cross-repo org visibility	■ Per-gateway only	■ Security Overview API
AI-generated fix	■ Flags only	■ Copilot writes the patch
Cost to scale to 100 APIs	$$$ Per-gateway licensing	■ One GitHub org subscription

> ***Recommended architecture:*** *Azure APIM enforces rate limits and auth at the gateway for runtime protection. GitHub Actions + GHAS finds and fixes the root-cause vulnerabilities in the code before they reach production. The two layers are complementary, not competing.*

Scaling to Every API Across the Organisation

The single most common follow-up question from engineering directors: *'This works for one repo — how does it scale to 200 microservices?'*

Step 1 — Enable GHAS org-wide (one API call):

```
PATCH /orgs/CushmanWakefield/advanced-security
{ "advanced_security_enabled_for_new_repositories": true,
  "secret_scanning_enabled_for_new_repositories": true,
  "dependabot_alerts_enabled_for_new_repositories": true }
```

Step 2 — Define a Reusable Security Workflow in the central security repo:

```
# .github/workflows/api-security-check.yml (in org-security repo)
on: workflow_call:
  inputs:
    api_url: { type: string, required: true }
    vuln_checks: { type: string, default: 'bola,auth,rate-limit,sqli' }
jobs:
  security-scan:
    # runs all selected checks, uploads SARIF, creates issues
```

Step 3 — Each API repo calls it with one line:

```
jobs:
  security:
    uses: org-security/.github/workflows/api-security-check.yml@main
    with:
      api_url: https://my-service.azure.com
      vuln_checks: 'bola,auth,rate-limit,sqli,ssrf'
```

Step 4 — Enforce via Required Workflows Ruleset (API call above). Every repo is covered.

> *Result: one engineering team manages one set of security workflows. Every API team in the org is automatically protected. New repos are enrolled automatically the moment they are created.*

Summary: What Cushman & Wakefield Gets

Requirement	GitHub Capability	Status
Set all security settings via API	GHAS REST API (org-level)	■ Available today
Detect exposure before attack	CodeQL on every PR	■ Available today
Real-time continuous monitoring	Actions cron probes + SARIF upload	■ Built in this demo
Anomaly detection with AI	GitHub Models API + Z-score analysis	■ Built in this demo
Human-in-the-loop fix approval	Issue labels + apply-fix workflow	■ Built in this demo
Scale to every API in org	Required Workflows + Rulesets API	■ Available today
Single pane of glass	Security Overview API	■ Available today

AI-written code fixes	GitHub Copilot Autofix	■ Available today

GitHub Advanced Security + GitHub Actions + Copilot is not just a security scanner — it is a continuous, AI-powered security engineering platform that finds vulnerabilities before attackers do, explains exactly what to fix, writes the fix, and verifies the fix — at org scale, with full audit trail, and with no additional infrastructure.