Developer Training

GitHub Advanced Security

Agenda

Introduction

Supply Chain Security

Secret Scanning

Code Scanning

Security Overview

The state of security today

DevSecOps Today

Security Risks

Applications are the #1 attack vector.

80% of breaches occur through web application exploits.

Source: Verizon Data Breach Investigations Reports 2023

Stagnant Progress

65% of vulnerabilities still exist after 3 months.

Only 33% of breaches are discovered by an organization's teams or tools.

Source: Veracode State of Security Report 2023

Force Multiplier

Organizations believe AI offers greater ROI

84% of executives plan to prioritize generative AI cybersecurity solutions over conventional cybersecurity solutions.

Source: IBM CEO's Guide to Generative AI, 2023

Looking to the future…

Our attack surface is growing at an unprecedented rate

We live in a world now fully consumed by software. Every organization is a software organization and must learn how to thrive and innovate digitally.

700M

More applications will be written in the next 5 years that’s more than the last 40 years combined

GitHub

AI-powered developer platform

GitHub Advanced Security

Feature List

Supply Chain

Know your environment

- GitHub Advisory Database
- Dependency Graph
- Dependency Insights
- Dependency Submission API
- Software Bill of Materials (SBOM)

Manage your environment

- Dependabot Alerts
- Dependabot Security Updates
- Dependabot Version Updates
- Dependency Review

Secret Scanning

- Partner pattern and non provider pattern detection by default
- Push Protections for users and repositories
- Custom Patterns
- Regex generation with Copilot
- Generic Secrets AI detection

Code Scanning

- Static analysis using CodeQL
- Use GitHub Actions or any other third-party CI/CD - SARIF output
- Repo rulesets
- Autofix AI
- Third-party code scanning tool support
- Security Campaigns

Security Overview Dashboard

Licence Feature List

What’s included in each licence

GHE Licence

- GitHub Advisory Database
- Dependency Graph
- Dependency Insights
- Dependency Submission API
- Software Bill of Materials (SBOM) generation
- Dependabot Alerts
- Dependabot Security Updates
- Dependabot Version Updates

GHAS Licence

- Dependabot auto triage rules
- Dependency Review
- All of the Secret Scanning capabilities including Copilot features
- All of the Code Scanning capabilities including Copilot features (not including runners)

Supply Chain Security

GitHub Advanced Security

Feature List

Supply Chain

Know your environment

- GitHub Advisory Database
- Dependency Graph
- Dependency Insights
- Dependency Submission API
- Software Bill of Materials (SBOM)

Manage your environment

- Dependabot Alerts
- Dependabot Security Updates
- Dependabot Version Updates
- Dependency Review

Secret Scanning

- Partner pattern and non provider pattern detection by default
- Push Protections for users and repositories
- Custom Patterns
- Regex generation with Copilot
- Generic Secrets AI detection

Code Scanning

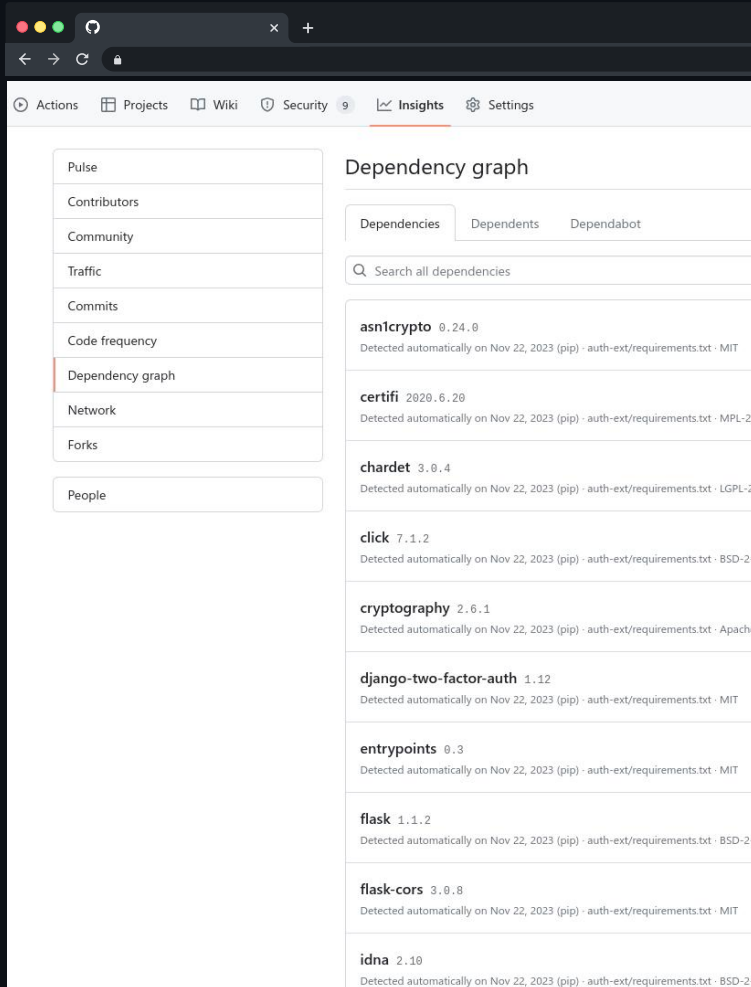
- Static analysis using CodeQL
- Use GitHub Actions or any other third-party CI/CD - SARIF output
- Repo rulesets
- Autofix AI
- Third-party code scanning tool support
- Security Campaigns

Security Overview Dashboard

Know Your Environment

Dependency Graph

- Summary of the manifest and lock files stored in a repository
- Includes information on your direct dependencies
- Each dependency includes the license information* and vulnerability severity
- Automatically updated when:
    - A commit is pushed that changes a supported manifest or lock file in a default branch
    - A change is pushed to a repository of one of your dependencies
- Repository and Organisation views



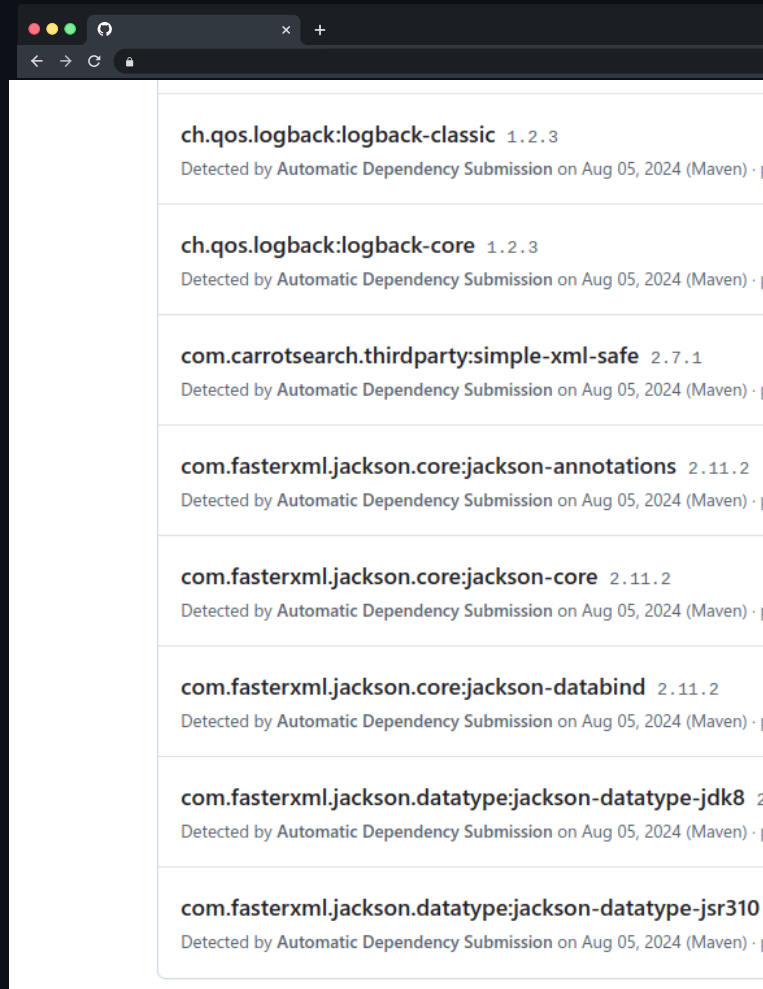
Dependency Submission API

- REST API to submit dependencies for repositories
- The dependency graph shows any dependencies you submit using the API in addition to any dependencies that are identified from manifest or lock files in the repository
- Submitted dependencies will receive Dependabot alerts and Dependabot security updates for any known vulnerabilities
- Submitted in the form of a snapshot which is a set of dependencies associated with a commit SHA and other metadata, that reflects the current state of your repository for a commit.

```yaml
name: Go Dependency Submission
on:
  push:
    branches:
      - main

# The API requires write permission on the repository to submit
permissions:
  contents: write

# Environment variables to configure Go and Go modules. Customi
env:
  GOPROXY: '' # A Go Proxy server to be used
  GOPRIVATE: '' # A list of modules are considered private and

jobs:
  go-action-detection:
    runs-on: ubuntu-latest
    steps:
      - name: 'Checkout Repository'
        uses: actions/checkout@v3

      - uses: actions/setup-go@v3
        with:
          go-version: ">=1.18.0"

      - name: Run snapshot action
        uses: actions/go-dependency-submission@v2
        with:
            # Required: Define the repo path to the go.mod file
            # build target
            go-mod-path: go-example/go.mod
            #
```

Automatic Dependency Submission

- For some ecosystems, resolution of transitive dependencies occurs at build-time
- Identifies and submits direct and transitive dependencies via the dependency submission API
- Only for Maven
- Requires GitHub Actions



Software Bill of Materials (SBOM)

- Formal, machine-readable inventory of project dependencies
  - Includes versions, package identifiers, licenses
- Benefits:
  - Reduces supply chain risks
  - Provides transparency of repository dependencies
  - Early identification of vulnerabilities
  - Insights into license compliance, security, and quality issues
  - Helps comply with data protection standards
- Generation Methods:
  - GitHub UI
  - GitHub Actions (attached as a workflow artifact)
  - REST API
- Uses SPDX industry standard format

GitHub Advisory Database

https://github.com/advisories

Each advisory in the GitHub Advisory Database is for a vulnerability in open source projects or for malicious open source software

We add advisories to the GitHub Advisory Database from the following sources:

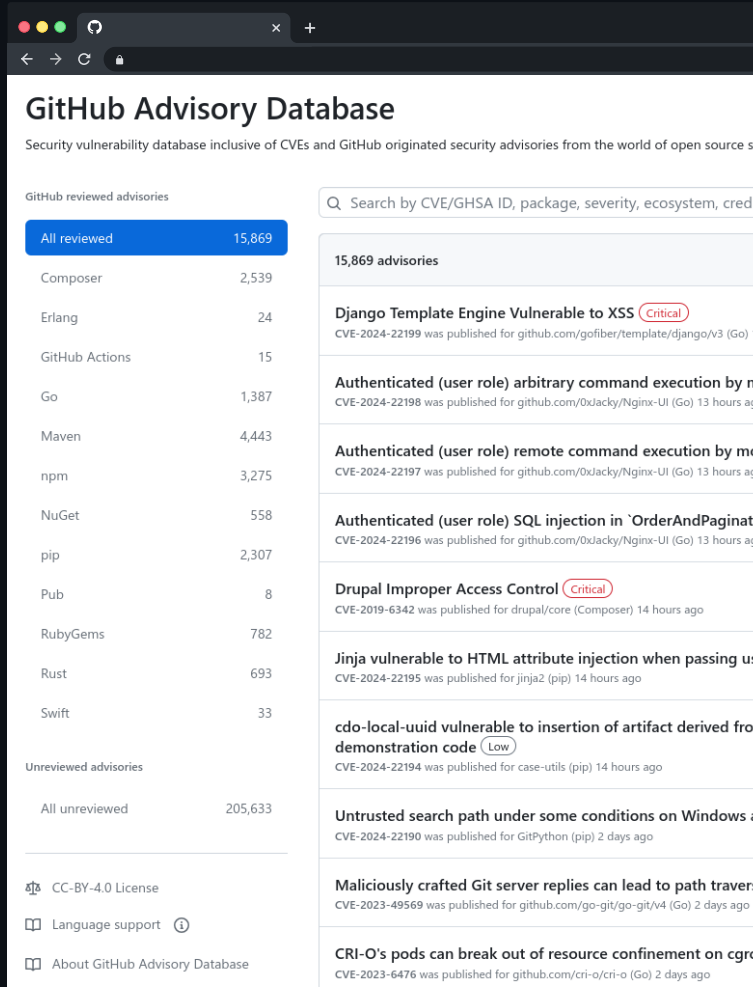
- Security advisories reported on GitHub
- The National Vulnerability database
- The npm Security advisories database
- The FriendsOfPHP database
- The Go Vulncheck database
- The Python Packaging Advisory database
- The Ruby Advisory database
- The RustSec Advisory database
- Community contributions. For more information, see https://github.com/github/advisory-database/pulls.

GitHub Advisory Database

https://github.com/advisories

The GitHub Advisory Database groups vulnerabilities into three categories:

- GitHub-reviewed advisories
- Unreviewed advisories
- Malware Advisories

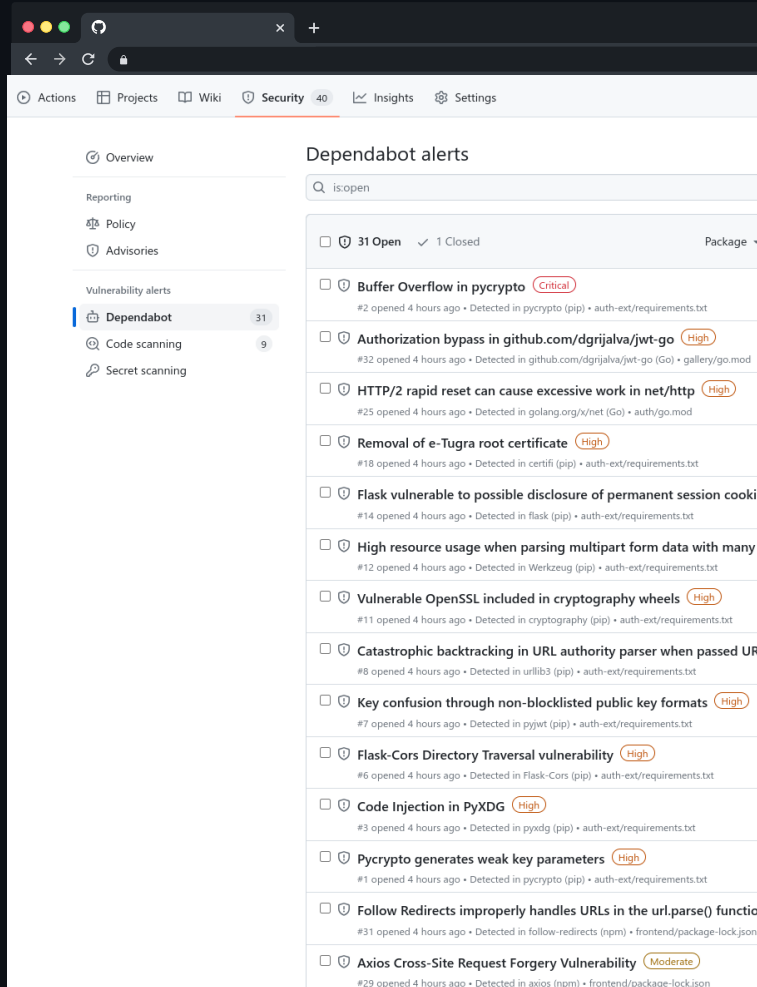


Q & A

Manage Your Environment

Dependabot Alerts

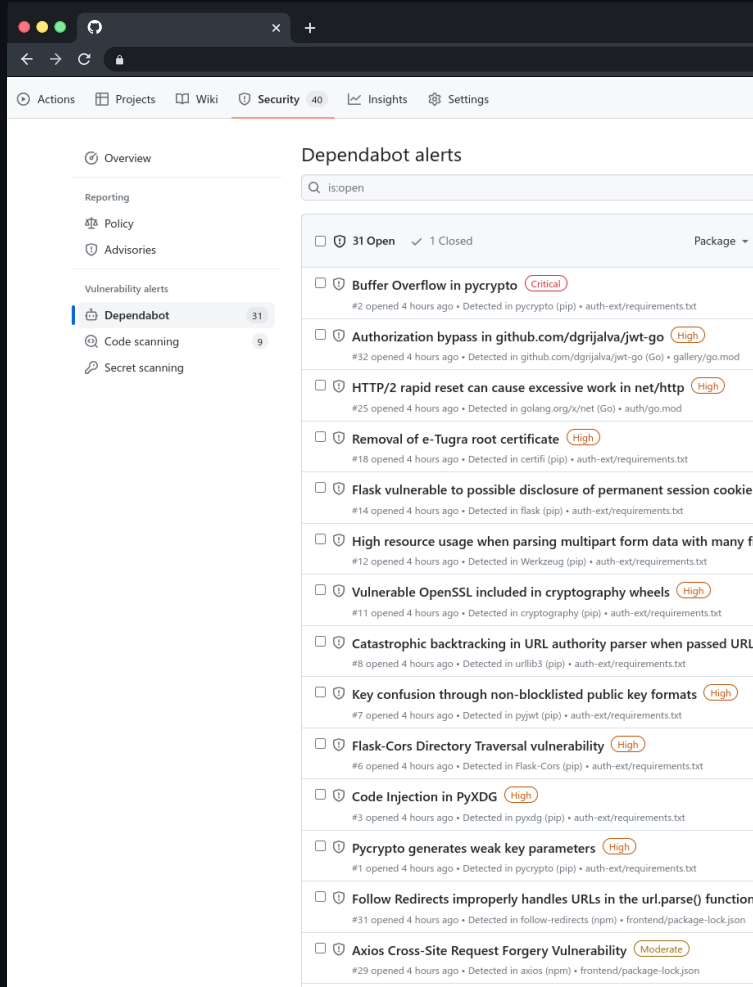
- A Dependabot alert is generated when a vulnerable dependency or malware is detected in a repository on the default branch
- Achieved by comparing a repository's dependency graph with the entries in the GitHub Advisory Database
- Admins can enable Dependabot alerts at the repository, organisation or enterprise



Dependabot Alerts

- After enablement a scan is triggered on the default branch when
  - A new advisory is added to the GitHub Advisory Database
  - The dependency graph for a repository changes.

- Users with write, maintain or admin privileges can view Dependabot alerts once enabled.

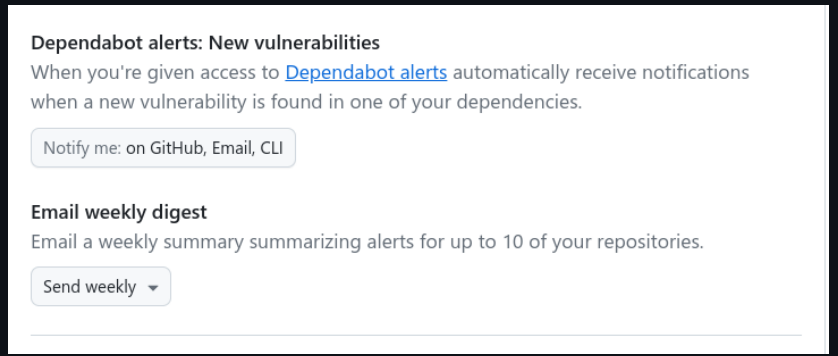


Dependabot Notifications

- When a new Dependabot alert is detected, GitHub notifies all users with access to Dependabot alerts for the repository according to their notification preferences

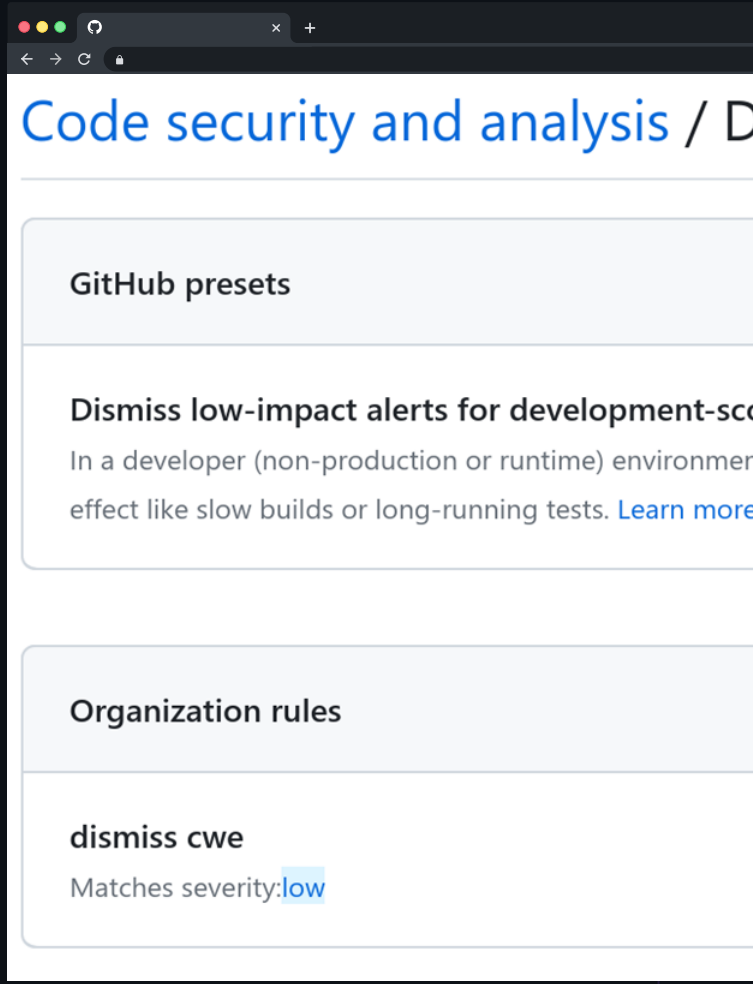
- You will receive alerts if you are:
  - watching the repository
  - have enabled notifications for security alerts or for all the activity on the repository,
  - and are not ignoring the repository

If you are concerned about receiving too many notifications for Dependabot alerts, we recommend you opt into the weekly email digest, or turn off notifications while keeping Dependabot alerts enabled.



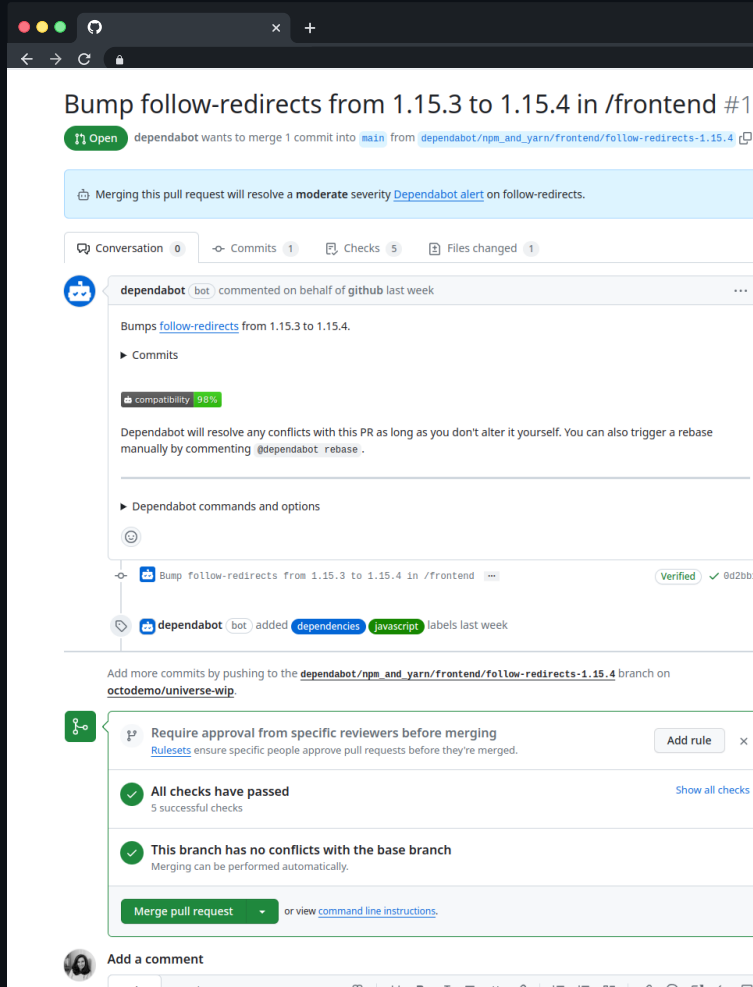
Dependabot auto-triage rules

- Instruct dependabot to automatically triage Dependabot alerts based on a specified criteria
- There are two types of Dependabot auto-triage rules:
  - GitHub-curated default rules
  - Custom auto-triage rules
- Auto-dismissed rules can be reopened if the alert metadata changes



Dependabot security updates

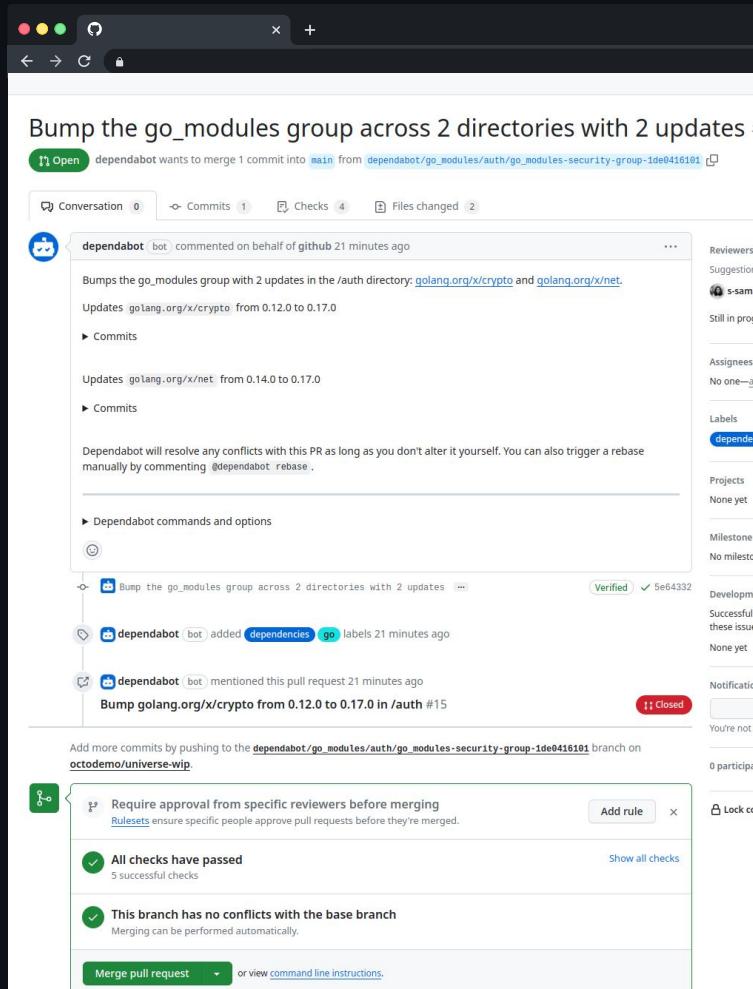
- Dependabot raises a pull request to update the dependency to the minimum version that includes the patch and links the pull request to the Dependabot alert, or reports an error on the alert
- Each pull request includes information about the vulnerability like release notes, changelog entries, and commit details. Details of which vulnerability a pull request resolves are hidden from anyone who does not have access to Dependabot alerts for the repository
- Pull request raised to default branch. This cannot be configured



Grouped security updates

- To reduce the number of pull requests, grouped security updates can be enabled which groups sets of dependencies together (per package ecosystem) and raises a PR.

- Dependabot will group dependencies from different directories. Dependabot will not group dependencies from different package ecosystems together, and it will not group security updates with version updates



Dependabot version updates

- Keeps your packages updated to the latest version by it raising a pull request to update the manifest to the latest version of the dependency.
- Enabled through a `dependabot.yml` configuration file
- Looks at the semantic versioning (semver) of the dependency to decide whether it should update to that version

Configuring Dependabot

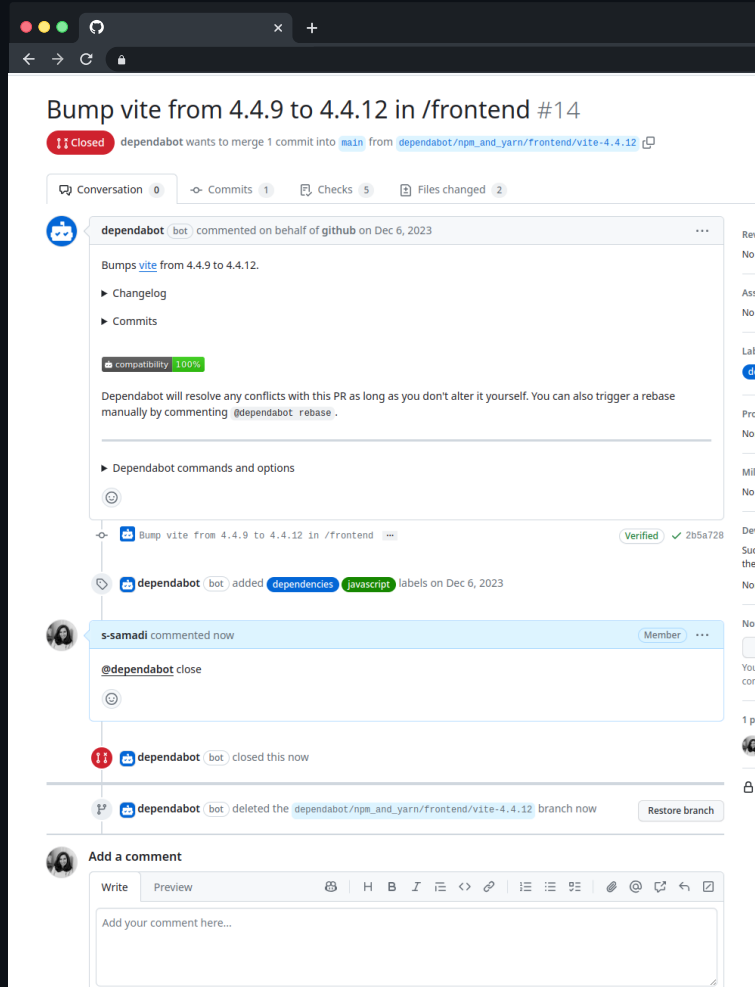
- Dependabot can be configured via the `Dependabot.yml` file
- Must be stored in the `.github` directory of the repository's default branch
- Configure it to access private registries

Configuring Dependabot

- These options fit broadly into the following categories:
  - Essential set up options
  - Options to customize the update schedule
  - Options to control which dependencies are updated
  - Options to add metadata to pull requests
  - Options to change the behavior of the pull requests

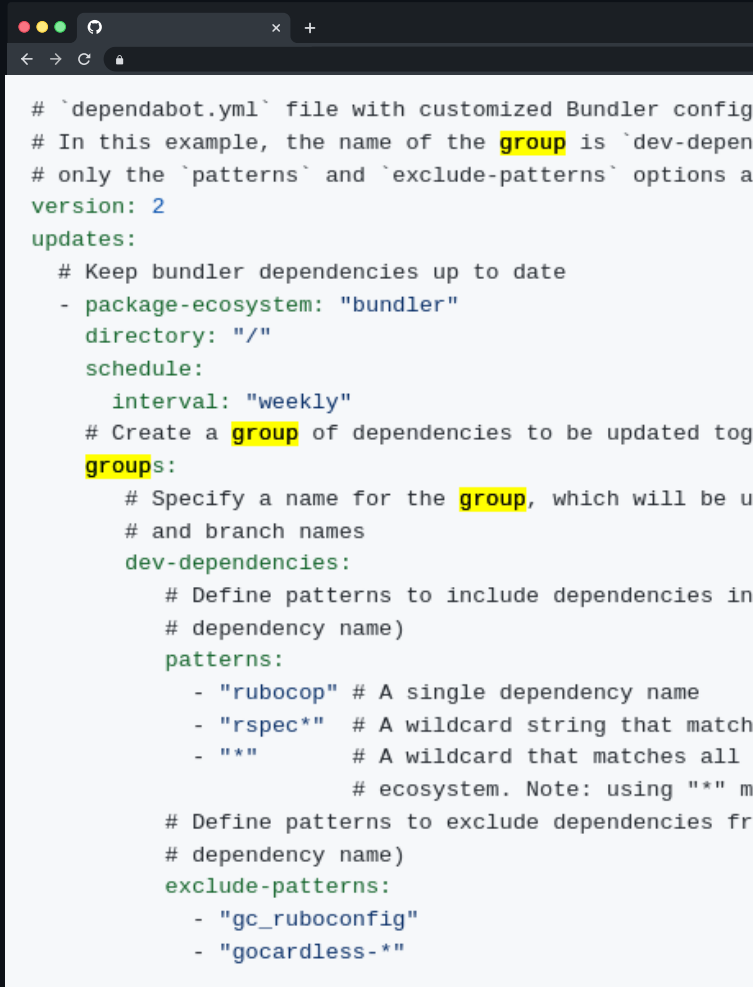
Managing Dependabot PRs

- Both security updates and version updates can be managed with comment commands in the PR
  - For example - @dependabot merge
- Comment commands are supported for grouped version updates but not grouped security updates
- Each dependabot run is captured in logs via the dependabot jobs list which is accessible via the dependency graph



Grouped version updates

- Dependabot version updates can be grouped via the `dependabot.yml` file
- Use the `groups` property. Options available:
  - dependency-type - either development or production
  - patterns - strings to match dependency name
  - exclude-patterns - if this is used, Dependabot will continue to use single PRs for version updates
  - update-types - specify semantic versioning; can be minor, patch, and major



```
# `dependabot.yml` file with customized Bundler config
# In this example, the name of the group is `dev-depen
# only the `patterns` and `exclude-patterns` options a
version: 2
updates:
  # Keep bundler dependencies up to date
  - package-ecosystem: "bundler"
    directory: "/"
    schedule:
      interval: "weekly"
    # Create a group of dependencies to be updated tog
    groups:
       # Specify a name for the group, which will be u
       # and branch names
       dev-dependencies:
          # Define patterns to include dependencies in
          # dependency name)
          patterns:
            - "rubocop" # A single dependency name
            - "rspec*"  # A wildcard string that match
            - "*"       # A wildcard that matches all
                        # ecosystem. Note: using "*" m
          # Define patterns to exclude dependencies fr
          # dependency name)
          exclude-patterns:
            - "gc_ruboconfig"
            - "gocardless-*"
```

Dependabot - Pitfalls and Wins

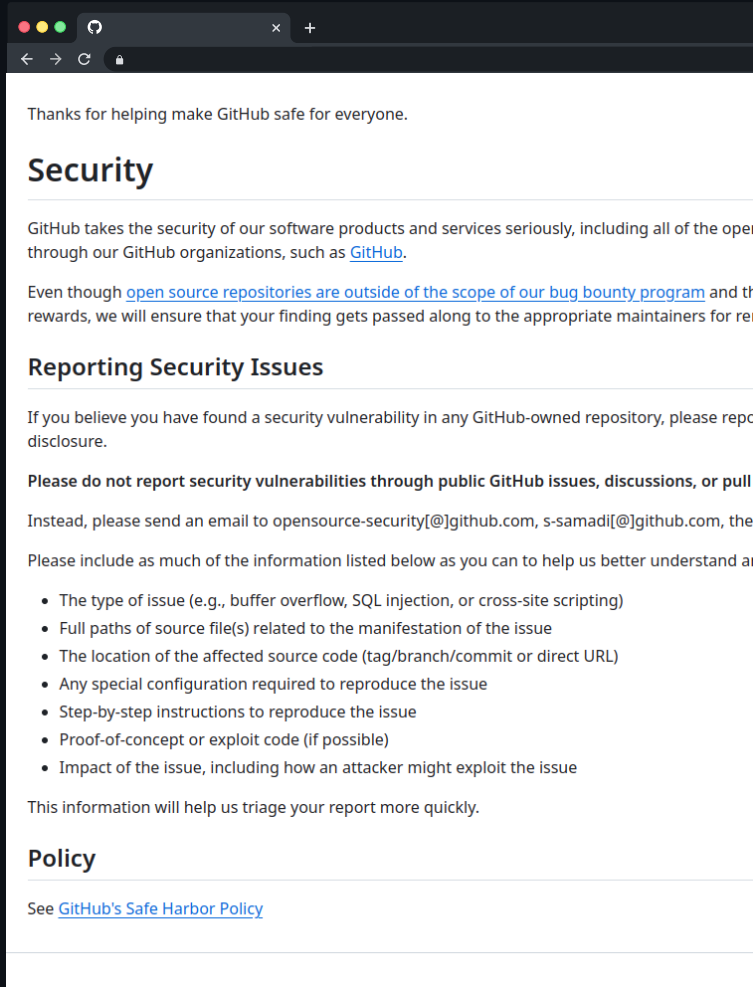
- You can configure Dependabot to only access private registries for a subset of ecosystems. You can find detailed information here
- If your private registry is configured with an IP allow list, you can find the IP addresses Dependabot uses to access the registry in the meta API endpoint
- By default, Dependabot can access public registries and additionally (if configured) can access private registries. If you want to restrict Dependabot from accessing public registries then you have to set the flag ***replaces-base*** as ***true*** (doc ref)
- When implementing branch protection rules, such as those for naming conventions, you have the option to exclude the Dependabot app using the bypass functionality. Failure to do so may result in Dependabot issues arising from branch protection rules
- Dependabot security updates raises a maximum of 10 PRs to the default branch. This cannot be configured.
- By default, Dependabot automatically rebases pull requests to resolve any conflicts. If a pull request has not been merged for 30 days, Dependabot will stop rebasing the pull request.
- Dependabot will prioritize newer alerts when generating security update PRs
- The Dependabot job logs list is accessible from the dependency graph of a repository. From the dependency graph, click the Dependabot tab, then to the right of the affected manifest file, click Recent update jobs.
- If you only require security updates and want to exclude version updates, you can set `open-pull-requests-limit` to 0 in order to prevent version updates for a given package-ecosystem.

Q & A

Publishing Vulnerability Information

SECURITY.md

- Instructions on how to report a security vulnerability in your repository
- May reside in the repository root, docs or .github folder
- When an issue is raised in the repository, a link to your repository's security policy will be present
- After someone reports a security vulnerability in your project, you can use GitHub Security Advisories to disclose, fix, and publish information about the vulnerability.



Thanks for helping make GitHub safe for everyone.

Security

GitHub takes the security of our software products and services seriously, including all of the oper
through our GitHub organizations, such as GitHub.

Even though open source repositories are outside of the scope of our bug bounty program and th
rewards, we will ensure that your finding gets passed along to the appropriate maintainers for re

Reporting Security Issues

If you believe you have found a security vulnerability in any GitHub-owned repository, please repo
disclosure.

Please do not report security vulnerabilities through public GitHub issues, discussions, or pull

Instead, please send an email to opensource-security[@]github.com, s-samadi[@]github.com, the

Please include as much of the information listed below as you can to help us better understand a

- The type of issue (e.g., buffer overflow, SQL injection, or cross-site scripting)
- Full paths of source file(s) related to the manifestation of the issue
- The location of the affected source code (tag/branch/commit or direct URL)
- Any special configuration required to reproduce the issue
- Step-by-step instructions to reproduce the issue
- Proof-of-concept or exploit code (if possible)
- Impact of the issue, including how an attacker might exploit the issue

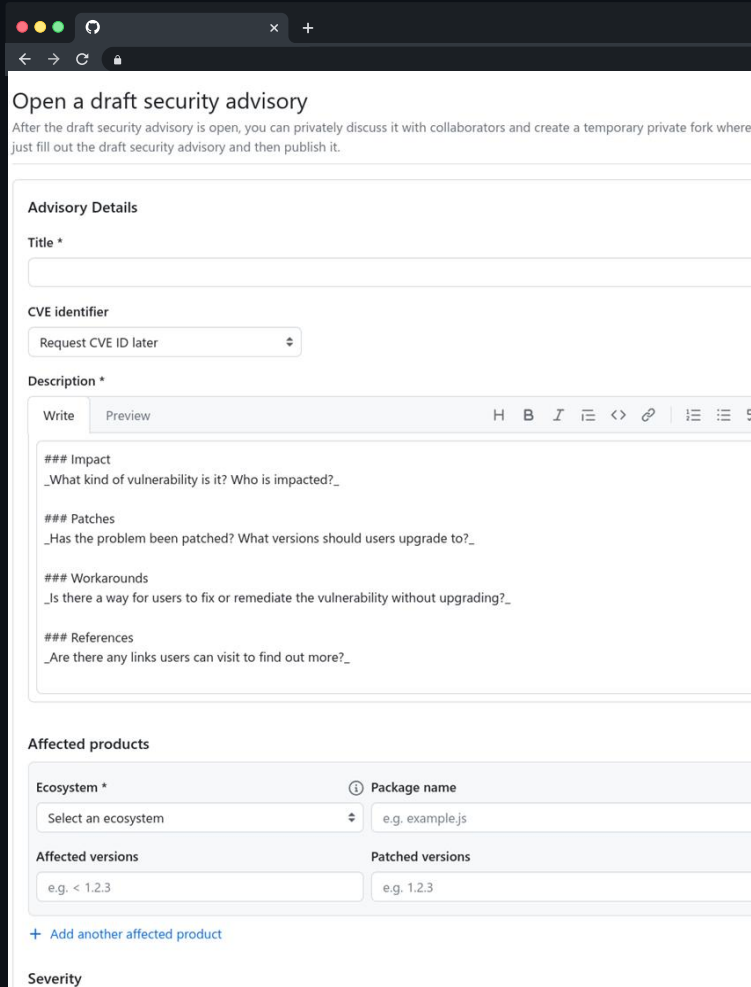
This information will help us triage your report more quickly.

Policy

See GitHub's Safe Harbor Policy

Security Advisories

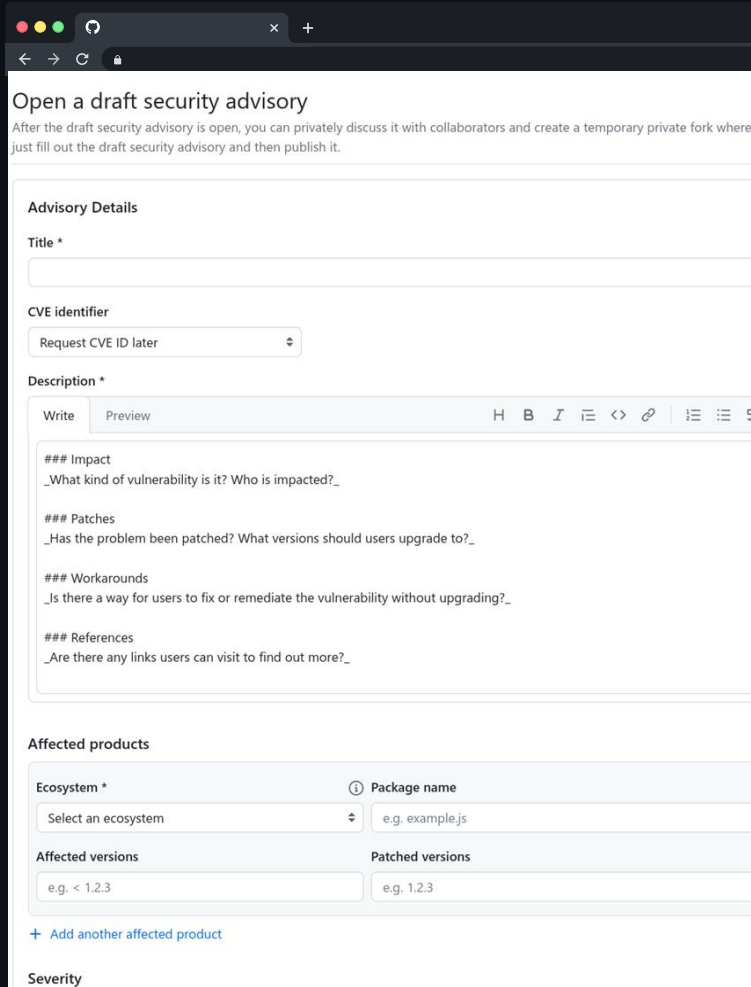
- Security advisories allow repository maintainers to privately discuss and fix a security vulnerability in a project.
- GitHub Security Advisories builds upon the foundation of the Common Vulnerabilities and Exposures (CVE) list. The security advisory form on GitHub is a standardized form that matches the CVE description format.



Security Advisories

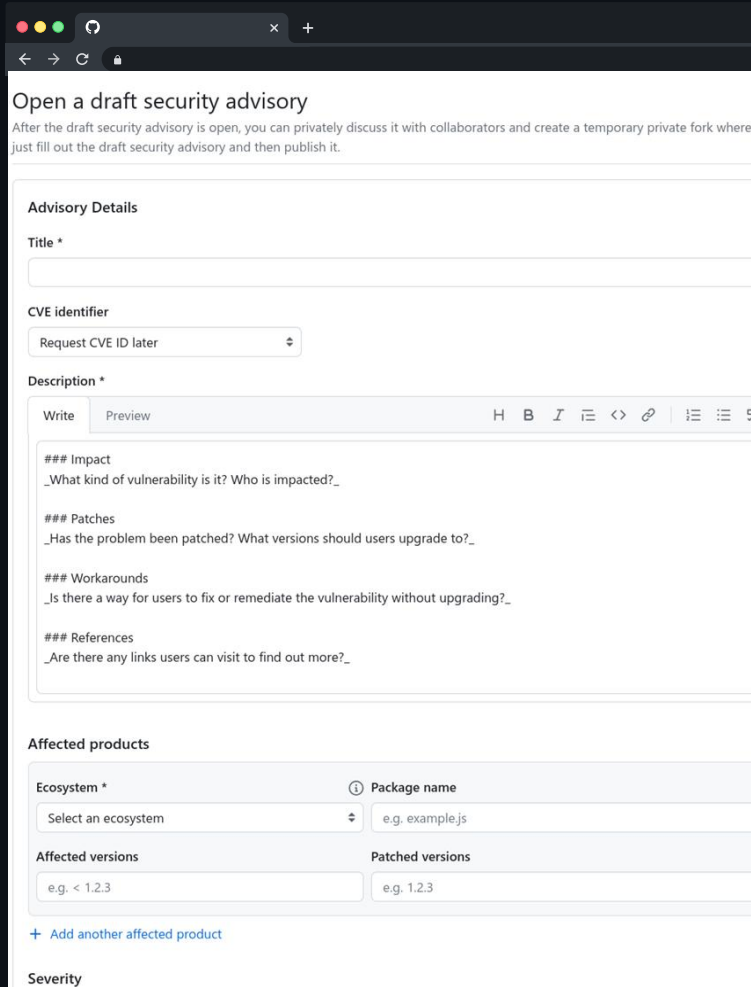
General process:

- Create a draft security advisory, and use the draft to privately discuss the impact of the vulnerability on your project
- Privately collaborate to fix the vulnerability in a temporary private fork.
- Publish the security advisory to alert your community of the vulnerability once a patch is released.



Security Advisories

- You can also use the REST API to create, list, and update repository security advisories
- You can give credit to individuals who contributed to a security advisory.
- For public repositories, once you've published the security advisory and GitHub has assigned a CVE identification number to the vulnerability, GitHub publishes the CVE to the MITRE database.



Q & A

Agenda

Introduction

Supply Chain Security

Secret Scanning

Code Scanning

Security Overview

Secret Scanning

GitHub Advanced Security

Feature List

Supply Chain

Know your environment

- GitHub Advisory Database
- Dependency Graph
- Dependency Insights
- Dependency Submission API
- Software Bill of Materials (SBOM)

Manage your environment

- Dependabot Alerts
- Dependabot Security Updates
- Dependabot Version Updates
- Dependency Review

Secret Scanning

- Partner pattern and non provider pattern detection by default
- Push Protections for users and repositories
- Custom Patterns
- Regex generation with Copilot
- Generic Secrets AI detection

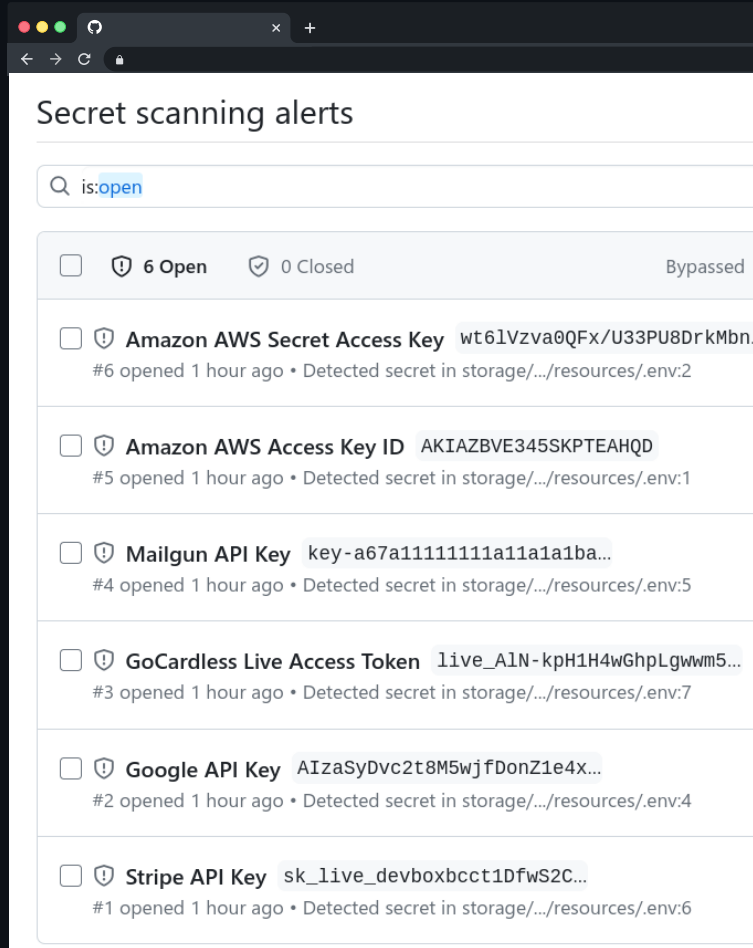
Code Scanning

- Static analysis using CodeQL
- Use GitHub Actions or any other third-party CI/CD - SARIF output
- Repo rulesets
- Autofix AI
- Third-party code scanning tool support
- Security Campaigns

Security Overview Dashboard

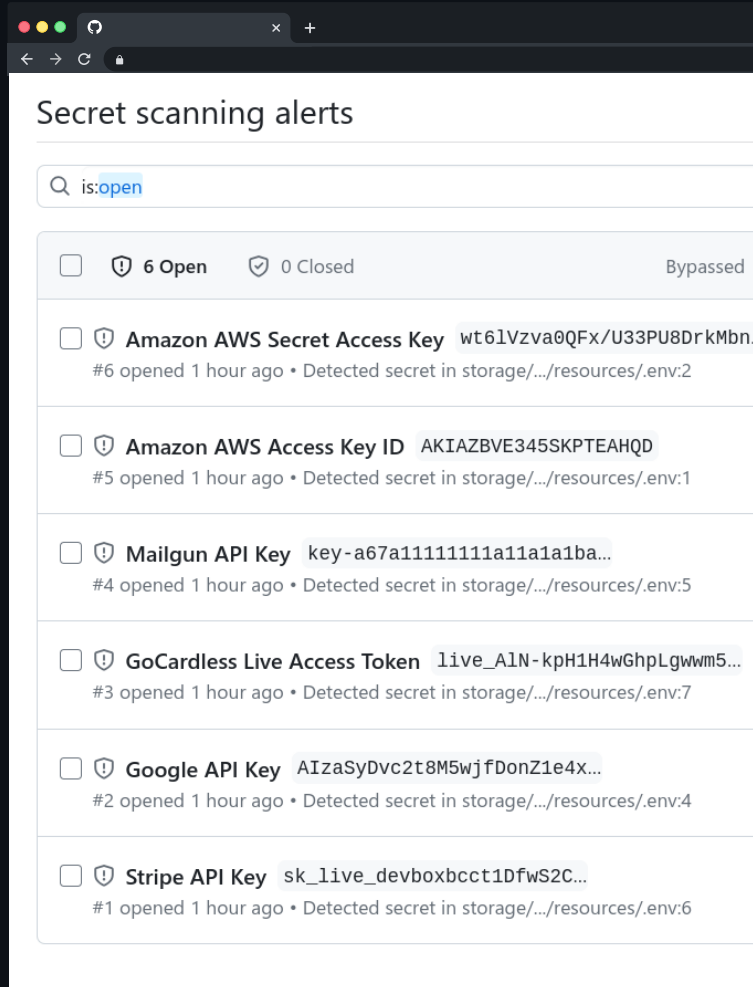
Secret Scanning

- Scans your entire Git history on all branches present in your GitHub repository for secrets, even if the repository is archived.
- Furthermore, it also scans descriptions, comments in issues, pull requests, and GitHub Discussions.
- Once enabled, a scan is triggered on every push. GitHub will periodically run full git history scans



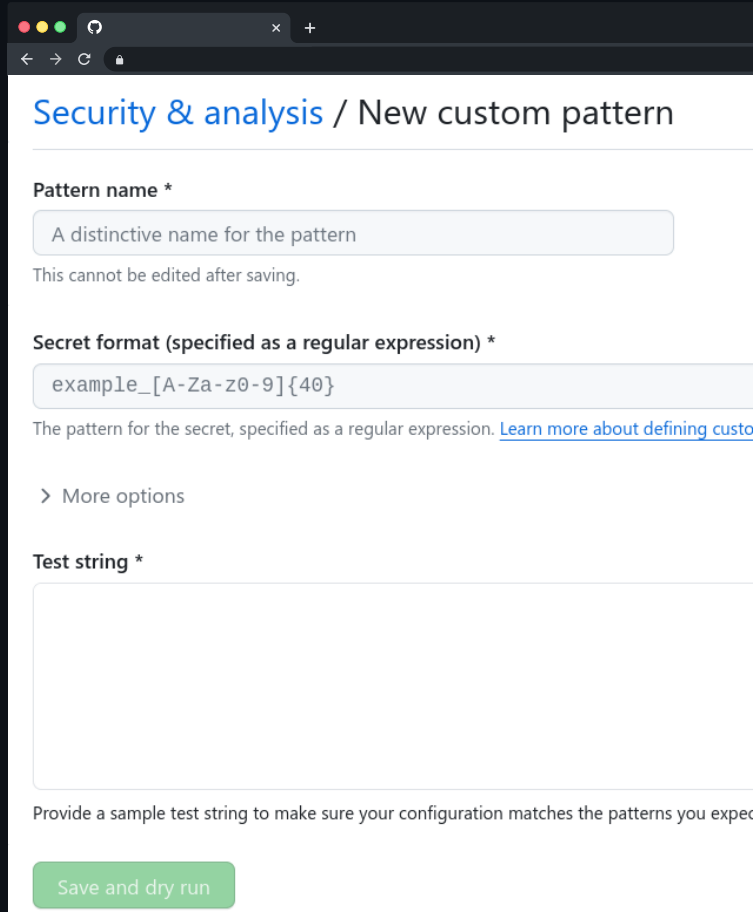
Secret Scanning

- Three types of secret scanning
  - Secret Scanning for token providers
  - Secret Scanning for non-provider patterns
  - User defined custom patterns
- Can perform validity checks
- Admin privileges are required to enable secret scanning on repository, organisation, or enterprise level



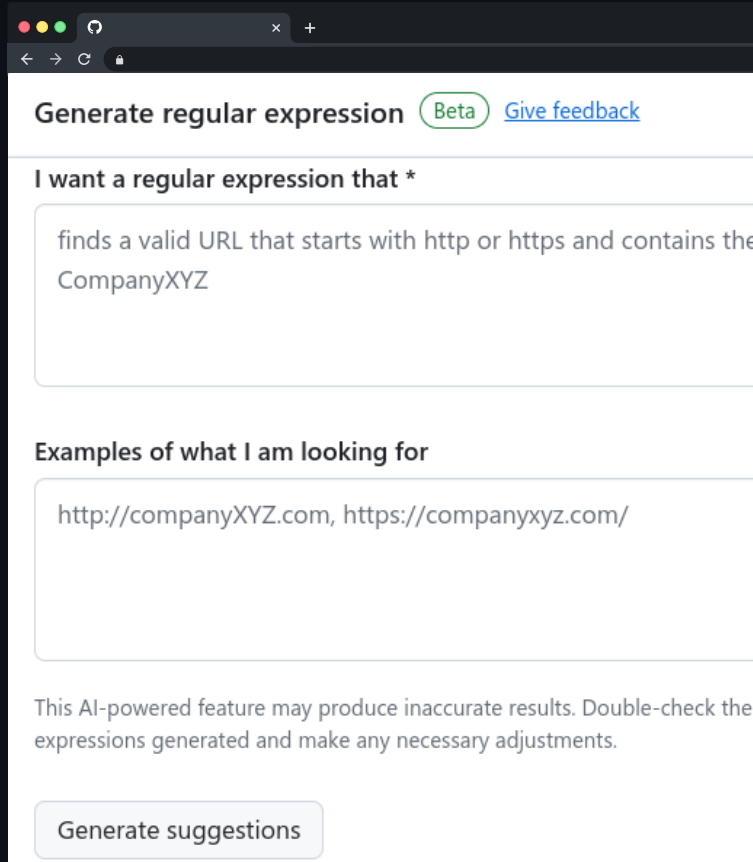
Custom Patterns

- Use regular expressions to define custom patterns detected as a part of secret scanning
- Repository, Organisation and Enterprise level
- Secret Scanning uses the Hyperscan library which only supports the Hyperscan regex construct. This is a subset of the PCRE syntax.
- Custom pattern specifiers:
  - Secret format
  - Before secret
  - After secret
  - Additional match requirements



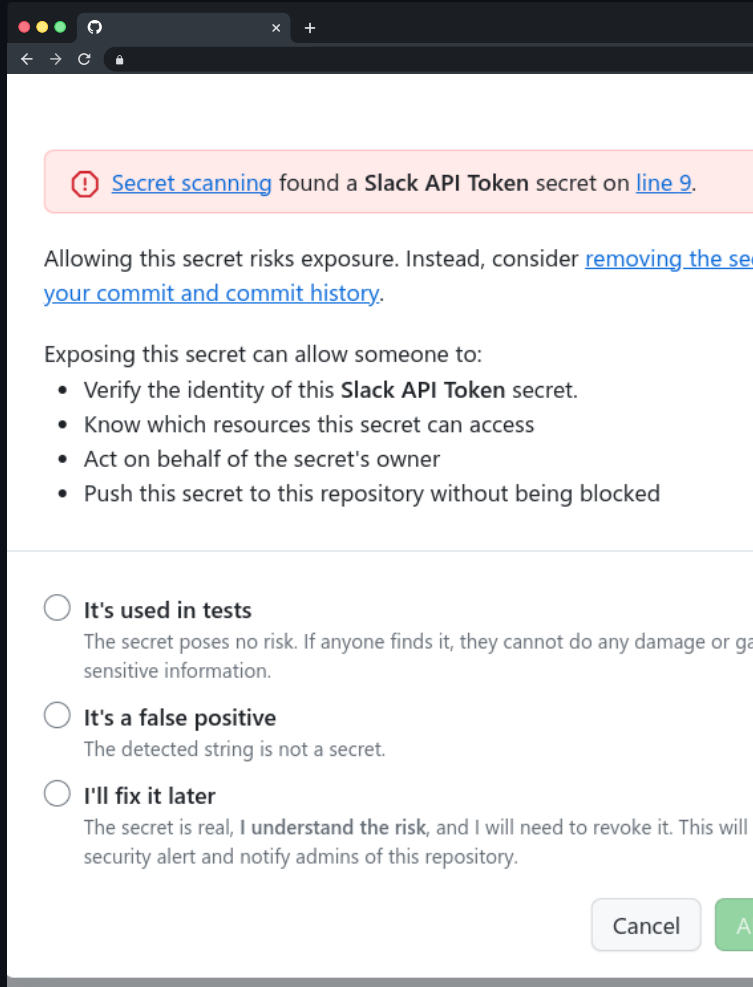
Custom Patterns + AI

- The regular expression generator allows for the generation of custom patterns without requiring knowledge of regular expressions
- Input - textual description of pattern to identify and option example string
- Output - up to 3 regular expressions



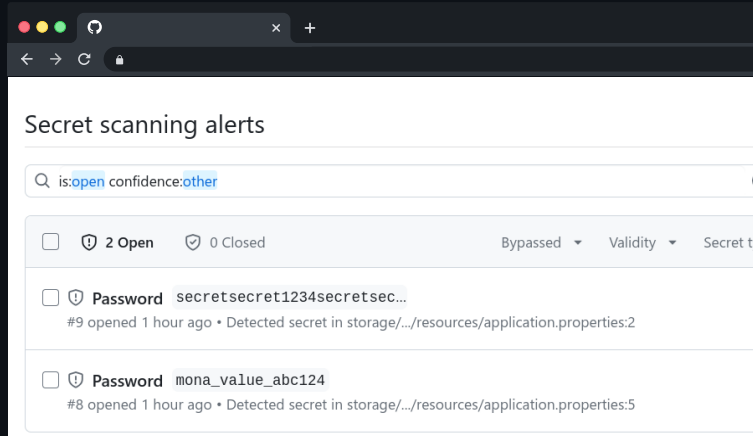
Push Protections

- Prevents secrets from being pushed into your repository for high confidence secrets
- Set at repository or organisation level
- Push protections for custom patterns can be enabled on the organisation and repository level
- Delegated bypass for push protection allows you to manage how contributors can bypass push protection rules, adding an approval process for those not on a designated bypass list.



Generic Secrets + AI

- Generic secret detection is an AI-powered expansion of secret scanning that identifies unstructured secrets (passwords) in your source code and then generates an alert
- Potential for false positives and false negatives
- Scope limited to Git repositories



Generic Secrets Limitations

Limited Scope

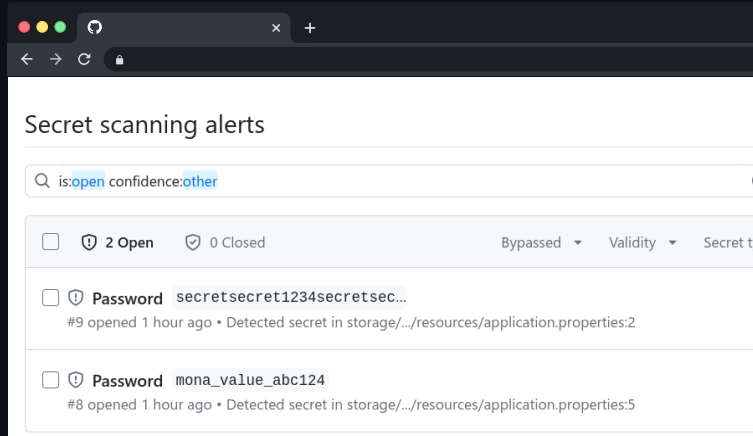
- AI-powered generic secret detection currently only scans for passwords in Git content.
- Does not detect other types of secrets.
- Non-git content (e.g., GitHub Issues) is excluded from the scan.

Potential for Incomplete Reporting

- May generate more false positives compared to existing secret scanning
- May miss credentials checked into repositories.

Evaluation Process

- Subject to Responsible AI Red Teaming.
- GitHub will continue monitoring the feature's efficacy and safety over time.



Secret Scanning Notifications

Historical Scans

Notifications are sent out to:

- Organization owners, enterprise owners, and security managers—whenever a historical scan is complete, even if no secrets are found.
- Repository administrators, security managers, and users with custom roles with read/write access—whenever a historical scan detects a secret, and according to their notification preferences.
- We do not notify commit authors.

Incremental Scans

Notifications are sent out to:

- Repository administrators
- Security managers
- Users with custom roles with read/write access
- Organization owners and enterprise owners, if they are administrators of repositories where secrets were leaked
- Commit authors who've accidentally committed secrets will be notified, regardless of their notification preferences

Q & A

Agenda

Introduction

Supply Chain Security

Secret Scanning

Code Scanning

Security Overview

Code Scanning

GitHub Advanced Security

Feature List

Supply Chain

Know your environment

- GitHub Advisory Database
- Dependency Graph
- Dependency Insights
- Dependency Submission API
- Software Bill of Materials (SBOM)

Manage your environment

- Dependabot Alerts
- Dependabot Security Updates
- Dependabot Version Updates
- Dependency Review

Secret Scanning

- Partner pattern and non provider pattern detection by default
- Push Protections for users and repositories
- Custom Patterns
- Regex generation with Copilot
- Generic Secrets AI detection

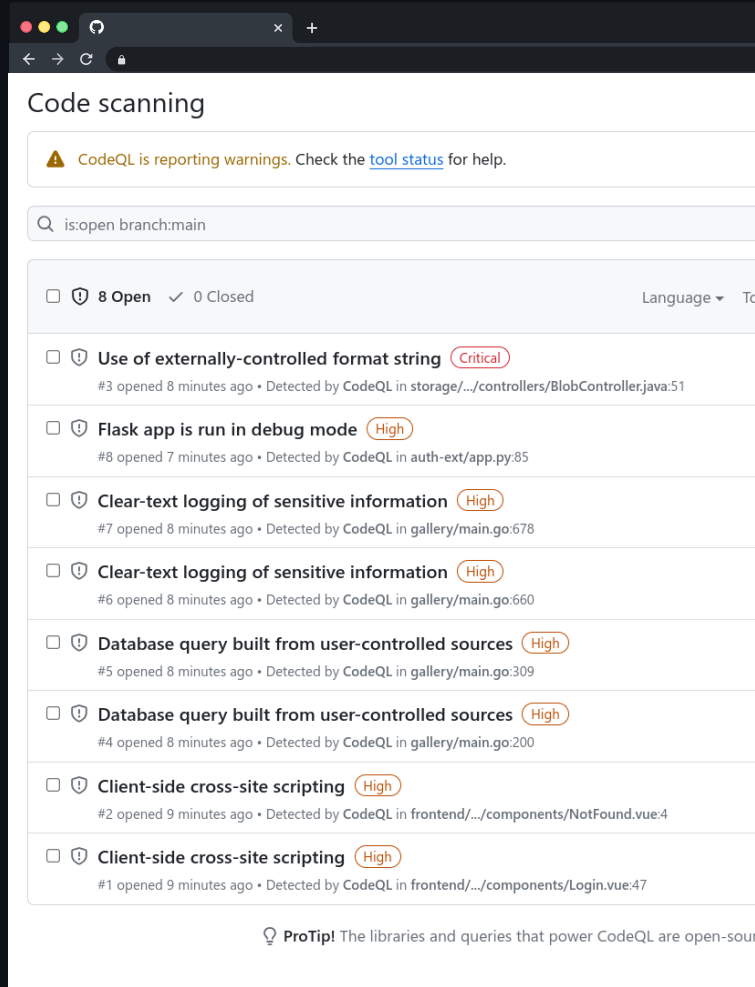
Code Scanning

- Static analysis using CodeQL
- Use GitHub Actions or any other third-party CI/CD - SARIF output
- Repo rulesets
- Autofix AI
- Third-party code scanning tool support
- Security Campaigns

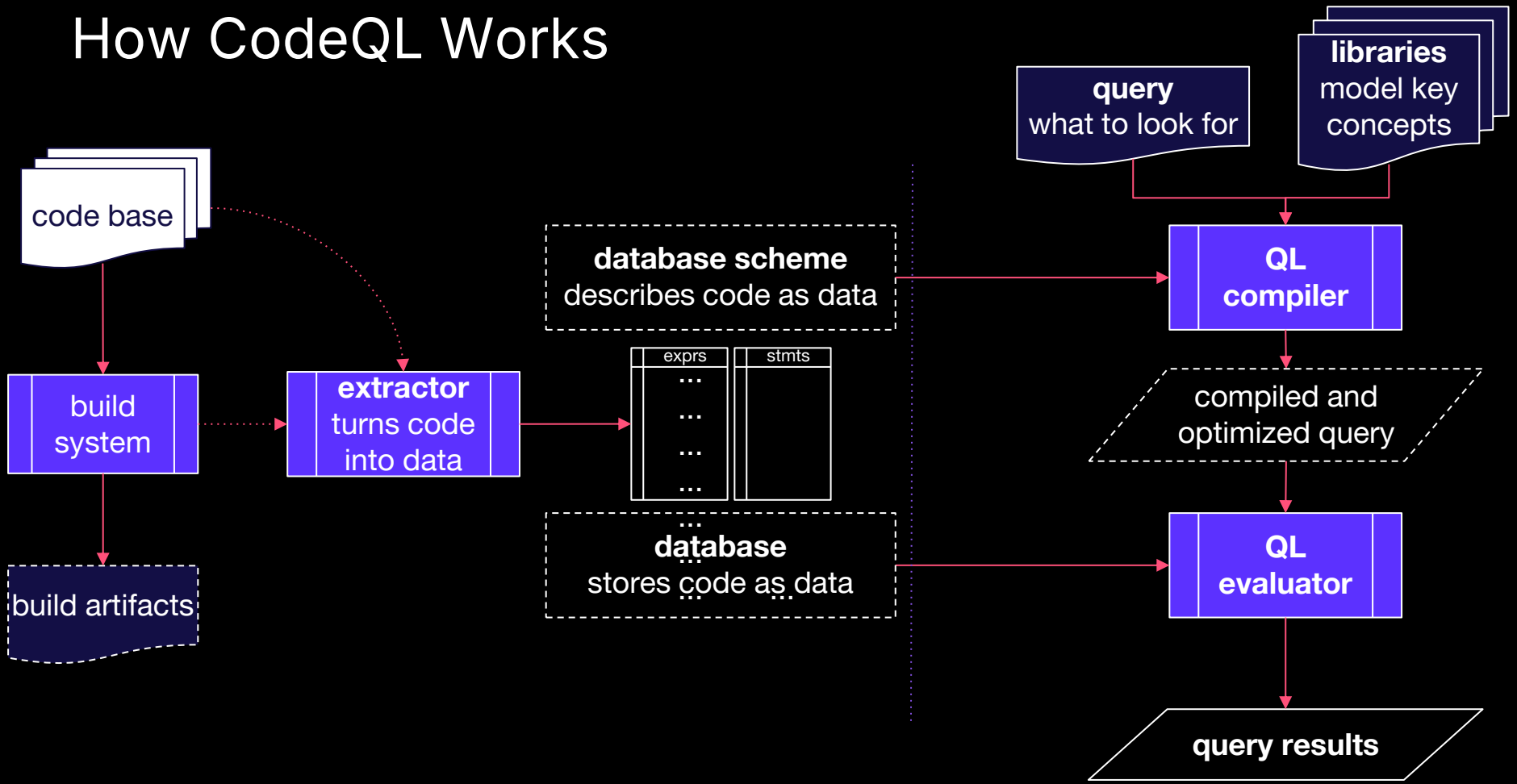
Security Overview Dashboard

Code Scanning

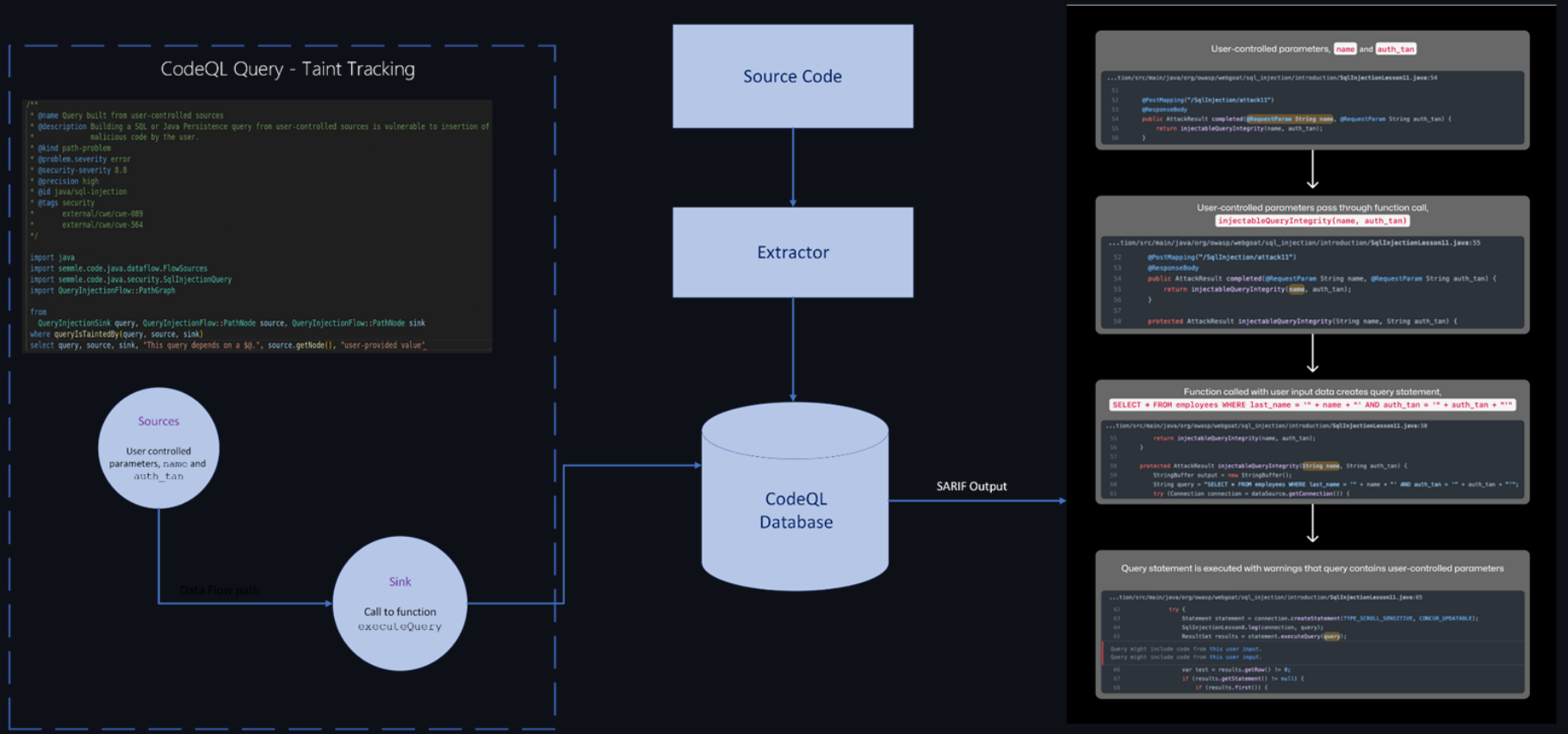
- Static analysis is a process that allows you to analyze an application's code for potential issues without executing the code itself
- GitHub code scanning allows you to automate scanning source code to prevent vulnerabilities and catch them earlier in the development pipeline
- Alerts can be viewed in the GitHub UI in a designated tab and also on the PR directly in the developer workflow
- At Github, we perform static analysis in code scanning via CodeQL, our semantic analysis engine



How CodeQL Works

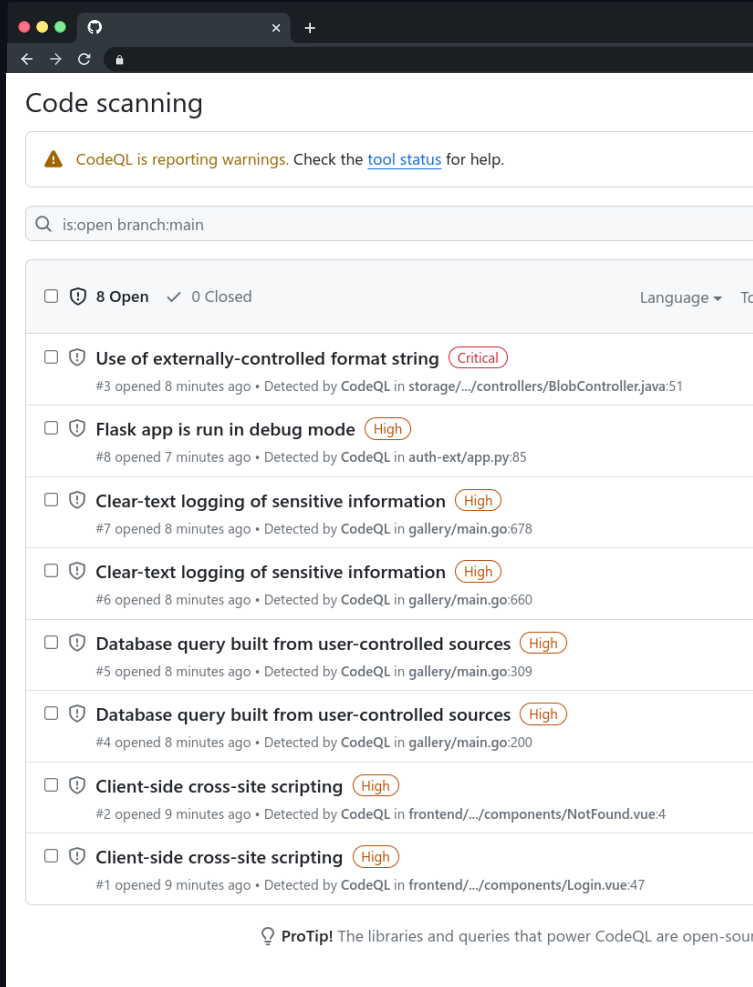


How CodeQL Works - SQL Injection Example



CodeQL

- CodeQL treats code as data. A database of the codebase is created and CodeQL queries are used to query this database
- These queries are divided into 3 suites - code scanning (default), security-extended, security-and-quality
- Three main ways to run CodeQL
  - Use default setup - automatically chooses language to analyze, query suite to run and events that trigger the scan
  - Advanced setup - GitHub Action workflow
  - Run CodeQL CLI in you external CI/CD and upload result to GitHub



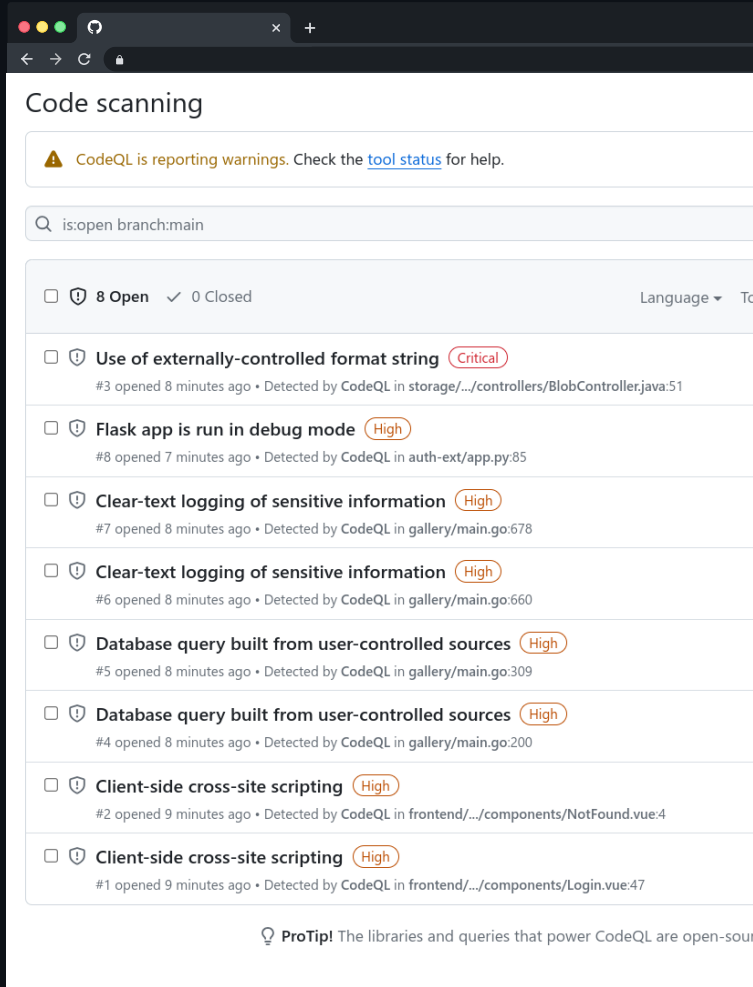
Default Setup

After enabling default setup, the code in your repository will be scanned:

- on each push to the repository's default branch, or any protected branch.
- when creating or committing to a pull request based against the repository's default branch, or any protected branch
- on a weekly schedule.

Your repository is eligible for default setup for code scanning if:

- it includes at least one CodeQL-supported language.
- GitHub Actions are enabled - supports both GitHub hosted runners and self hosted runner
- it is publicly visible, or GitHub Advanced Security is enabled.



Advanced Setup

- GitHub Actions workflow - YAML file
- Main components in files
  - Trigger points
    - Push
    - Pull request
    - Schedule (default branch)
    - Workflow Dispatch
    - Path-ignore (when not what).
  - Operating System: Linux, Windows, OS
  - CodeQL Action Init - creates CodeQL database, input query suite and language
  - Input custom build steps if manual
  - CodeQL Action Analyze - run CodeQL queries on database

```yaml
name: "CodeQL"

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]
  schedule:
    - cron: '33 15 * * 4'

jobs:
  analyze:
    name: Analyze
    runs-on: ${{ (matrix.language == 'swift' && 'macos-latest') || 'ubuntu-latest' }}
    timeout-minutes: ${{ (matrix.language == 'swift' && 120) || 360 }}
    permissions:
      # required for all workflows
      security-events: write
      # only required for workflows in private repositories
      actions: read
      contents: read

    strategy:
      fail-fast: false
      matrix:
        language: [ 'go', 'java-kotlin', 'javascript-typescript', 'python' ]

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

    # Initializes the CodeQL tools for scanning.
    - name: Initialize CodeQL
      uses: github/codeql-action/init@v3
      with:
        languages: ${{ matrix.language }}

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

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

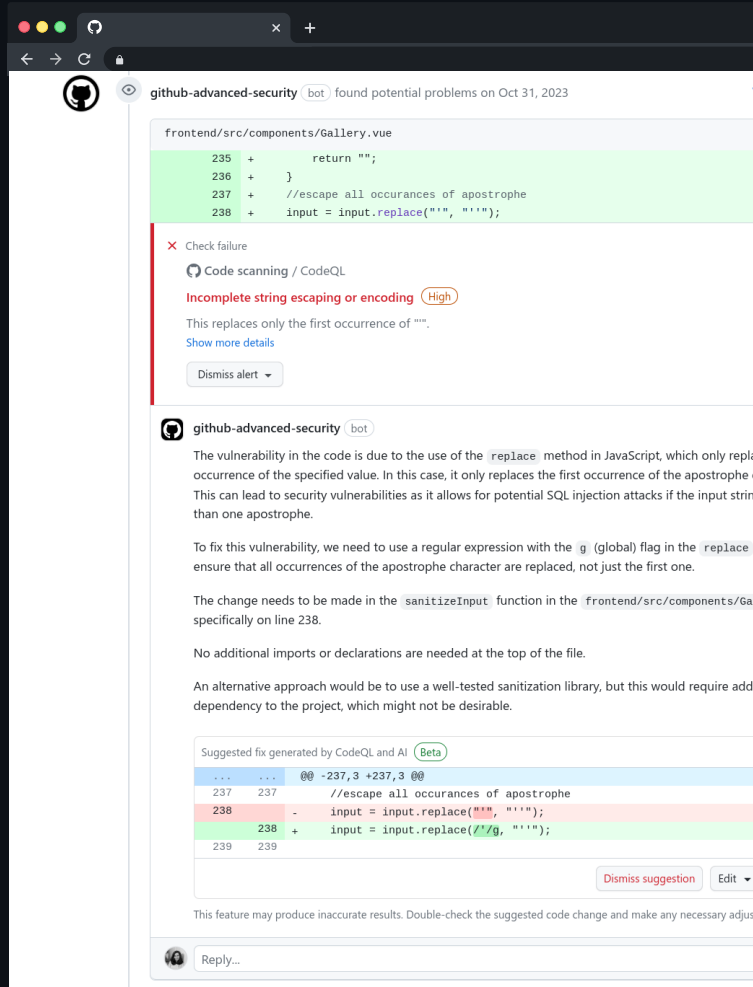
CodeQL CLI

- Standalone, command-line tool
- 3 commands to generate analysis results and upload on GitHub:
  - `database create`
  - `database analyze`
  - `github upload results`

```
# Create CodeQL databases for Java and Python in the 'codeql-dbs' directory
# Call the normal build script for the codebase: 'myBuildScript'

codeql database create codeql-dbs --source-root=src \
    --db-cluster --language=java,python --command=./myBuildScript

# Analyze the CodeQL database for Java, 'codeql-dbs/java'
# Tag the data as 'java' results and store in: 'java-results.sarif'

codeql database analyze codeql-dbs/java java-code-scanning.qls \
    --format=sarif-latest --sarif-category=java --output=java-results.sarif

# Analyze the CodeQL database for Python, 'codeql-dbs/python'
# Tag the data as 'python' results and store in: 'python-results.sarif'

codeql database analyze codeql-dbs/python python-code-scanning.qls \
    --format=sarif-latest --sarif-category=python --output=python-results.sarif

# Upload the SARIF file with the Java results: 'java-results.sarif'
# The GitHub App or personal access token created for authentication
# with GitHub's REST API is available in the `GITHUB_TOKEN` environment variable.

codeql github upload-results \
    --repository=my-org/example-repo \
    --ref=refs/heads/main --commit=deb275d2d5fe9a522a0b7bd8b6b6a1c939552718 \
    --sarif=java-results.sarif

# Upload the SARIF file with the Python results: 'python-results.sarif'

codeql github upload-results \
    --repository=my-org/example-repo \
    --ref=refs/heads/main --commit=deb275d2d5fe9a522a0b7bd8b6b6a1c939552718 \
    --sarif=python-results.sarif
```

Autofix AI

- Code scanning autofix, powered by GitHub Copilot, enhances CodeQL's capabilities by providing automatic, targeted suggestions to resolve code scanning alerts. It helps prevent new security vulnerabilities by leveraging large language models (LLMs) to generate potential fixes based on codebase data and CodeQL analysis
- Converts alert descriptions and locations into actionable code changes using GPT-4o from OpenAI
- Enabled by default for repositories using CodeQL; can be disabled at the enterprise, organization, or repository level
- Autofix supports C#, C/C++, Go, Java/Kotlin, Swift, JavaScript/TypeScript, Python, and Ruby.



Security Overview

GitHub Advanced Security

Feature List

Supply Chain

Know your environment

- GitHub Advisory Database
- Dependency Graph
- Dependency Insights
- Dependency Submission API
- Software Bill of Materials (SBOM)

Manage your environment

- Dependabot Alerts
- Dependabot Security Updates
- Dependabot Version Updates
- Dependency Review

Secret Scanning

- Partner pattern and non provider pattern detection by default
- Push Protections for users and repositories
- Custom Patterns
- Regex generation with Copilot
- Generic Secrets AI detection

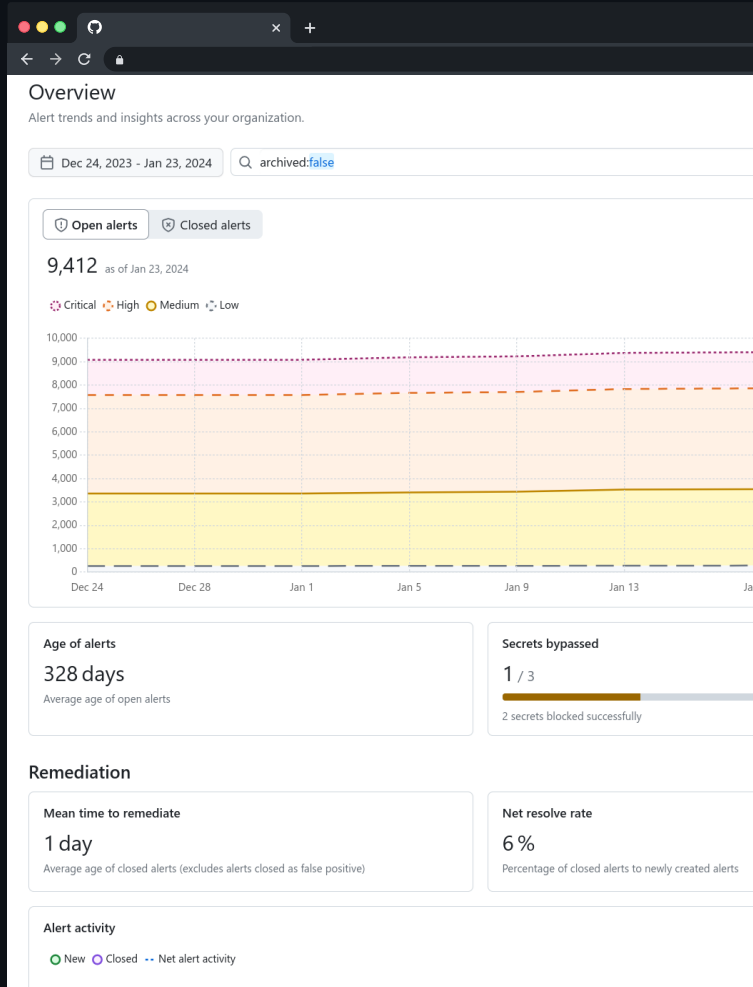
Code Scanning

- Static analysis using CodeQL
- Use GitHub Actions or any other third-party CI/CD - SARIF output
- Repo rulesets
- Autofix AI
- Third-party code scanning tool support
- Security Campaigns

Security Overview Dashboard

Security Overview

- Provides high-level summaries of the security landscape of an organization or enterprise and makes it easy to identify repositories that require intervention
- Shows metrics on default branches
- At organisation level, 3 summary views:
  - Overview
  - Coverage
  - Risk
- Show data only for high confidence alerts. Code scanning alerts from third-party tools, and secret scanning alerts for ignored directories and non-provider alerts are all omitted from these views



Thank you