PassQueen
a.k.a dumpsqueen.com

GitHub Advanced Security Exam

Microsoft GH-500

Version Demo

Total Demo Questions: 17

Total Premium Questions: 172

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Configure and use secret scanning	45
Topic 2, Configure and use dependency management	59
Topic 3, Configure and use code scanning	55
Topic 4, Mix Questions	13
Total	172

QUESTION NO: 1

Which GitHub secret scanning feature helps security teams prioritize alerts for exposed credentials that pose an immediate risk?

A. Non-provider patterns

B. Push protection

C. Custom pattern dry runs

D. Secret validation

ANSWER: D

Explanation:

Secret validation is the feature that helps teams identify and prioritize secret scanning alerts requiring immediate attention. For supported token types, GitHub checks with the relevant service provider to determine whether a detected credential is currently active. If the provider confirms that the credential is active, GitHub marks the secret scanning alert as verified. This status is a strong risk signal because an active credential exposed in a repository could potentially be used to access the associated service, data, or resources without authorization. Teams can use verified status to focus remediation work on the most urgent findings, including revoking or rotating the exposed credential and reviewing its potential use. Secret validation applies only to supported secret types and providers, and the verification result is shown in the alert details so that responders can make informed prioritization decisions. GitHub documents the validation process and verified-alert status in its Secret validation documentation. For broader context on how detected credentials and alerts are managed, see the GitHub secret scanning overview.

QUESTION NO: 2

Which GitHub secret scanning feature determines whether a detected secret is still active or valid?

A. Push protection

B. Validity checks

C. Branch protection

D. Custom patterns

ANSWER: B

Explanation:

Validity checks is correct because GitHub Secret Protection can work with supported secret providers to determine whether a detected credential remains active. After secret scanning identifies a supported token or credential, GitHub can perform a validity check and assign a validation status to the resulting alert. This status gives security teams useful context for triage: an active credential represents a potentially usable exposure and generally warrants prompt remediation, such as revoking or rotating the secret and reviewing any associated access. A credential that is no longer active may still require investigation, but its immediate risk is different. Validity checks are therefore specifically designed to add real-world credential status to secret-scanning findings rather than merely identifying a pattern that resembles a secret. GitHub documents that validity checks are available for selected partner patterns and that alert validation status can help organizations prioritize remediation based on whether the exposed secret is active. See GitHub Docs: About secret validity checks and GitHub Docs: About secret scanning.

QUESTION NO: 3

Configure and use dependency management

Which GitHub security feature identifies a vulnerable dependency as part of a pull request?

A. Dependency graph

B. Dependency review

C. Dependabot alert

D. The repository's Security tab

ANSWER: B

Explanation:

Dependency review is the correct feature because it is designed to evaluate dependency changes in the context of a pull request before those changes are merged. When a pull request modifies a supported dependency manifest or lock file, dependency review compares the base revision with the proposed revision and reports dependencies that are added, removed, or updated. Crucially, it flags newly introduced dependencies that have known vulnerabilities, helping reviewers identify supply-chain risk during code review rather than after deployment or merge.

Dependency review can also show related information that supports an informed merge decision, such as the affected package, its version change, and whether the dependency has available security updates. Repository rules can require dependency review checks to pass, allowing teams to prevent pull requests that introduce vulnerable packages from being merged unless an authorized reviewer accepts the risk or the dependency is remediated. This makes it a proactive pull-request security control within GitHub Advanced Security. For implementation requirements and supported package ecosystems, see GitHub Docs: About dependency review. For guidance on enforcing this check through rulesets, see GitHub Docs: Configuring dependency review.

QUESTION NO: 4

How can developers prevent a pull request from introducing new dependencies with known vulnerabilities?

A. Enable Dependabot alerts.

B. Add Dependabot rules.

C. Add a workflow with the dependency review action.

D. Enable Dependabot security updates.

ANSWER: C

Explanation:

Add a workflow with the dependency review action. This GitHub Action is designed to run in pull-request workflows and evaluates dependency changes between the pull request and its base branch. It identifies dependencies that are newly added or updated by the proposed change, then checks the relevant package versions for known vulnerabilities. This gives developers actionable security feedback before the change is merged into the target branch.

The action can be configured with a vulnerability severity threshold, enabling the workflow to fail when a newly introduced dependency has a vulnerability at or above that threshold. When combined with required status checks on a protected branch, this provides an effective control that prevents the pull request from being merged until the dependency issue is addressed. Teams can then select a safe package version, replace the dependency, or remove it if it is unnecessary. This approach evaluates risk precisely at the point where the dependency is introduced, making it appropriate for preventing vulnerable additions in pull requests. See the Dependency Review Action repository and GitHub's dependency review documentation.

QUESTION NO: 5

You need to integrate CodeQL code scanning into a third-party continuous integration system rather than GitHub Actions. Which three steps are required to run the analysis and make the results available in GitHub? Choose three.

A. Process alerts

B. Analyze code

C. Upload scan results

D. Install the CLI

E. Write queries

ANSWER: B C D

Explanation:

To use CodeQL in a third-party CI environment, first install the CodeQL CLI on the runner or build agent. The CLI provides the commands needed to create a CodeQL database from the repository's source code and to execute the relevant CodeQL analysis. Next, analyze code by using the CLI as part of the CI workflow. For compiled languages, the workflow must generally build the code while CodeQL database creation is in progress so that CodeQL can observe the build and construct an accurate database. Finally, upload scan results to GitHub after producing results in SARIF format. GitHub ingests SARIF uploads and displays the resulting findings in the repository's code-scanning views, where alerts can then be triaged and managed. These steps let an external CI platform perform scanning while GitHub Advanced Security centralizes the security results. GitHub documents the required CLI workflow, including database creation, analysis, and SARIF upload, in its guidance for CodeQL CLI integration. See Using CodeQL code scanning with your existing CI system and SARIF support for code scanning.

QUESTION NO: 6

Before defining a custom secret-scanning pattern for a repository, what must you do?

A. Provide a regular expression for the format of your secret pattern.

B. Add a secret scanning custom pattern.

C. Enable secret scanning on the repository.

D. Provide match requirements for the secret format.

ANSWER: C

Explanation:

Enable secret scanning on the repository. Custom patterns are a secret-scanning capability: they let an organization define detectors for sensitive values that are specific to its own services, tokens, or credential formats. Therefore, the repository must have secret scanning enabled before GitHub can use a custom pattern to scan repository content and produce secret-scanning alerts for matches. Once the feature is available, an authorized user can define the pattern's regular expression and optionally configure additional match requirements to make detections more precise. The resulting pattern extends GitHub's built-in secret detectors and is evaluated as part of secret scanning. For private repositories, use of secret scanning and custom patterns can depend on the organization's GitHub Advanced Security entitlement and configuration. GitHub's guidance for defining custom patterns for secret scanning describes custom patterns as part of the secret-scanning workflow. The prerequisite feature can be enabled and managed according to the documentation for configuring secret scanning for repositories.

QUESTION NO: 7

An organization has created reusable CodeQL queries and wants multiple repositories to use them through the `packs` setting in an advanced CodeQL setup. Which actions are required to make the shared query pack available to these workflows? (Choose two.)

A. Publish the CodeQL pack to a package registry that CodeQL can access

B. Reference the pack by its pack name and version or version range

C. Copy every query file from the pack into each repository

D. Upload a CodeQL database containing the pack

E. Create a Dependabot configuration for the pack

ANSWER: A B

Explanation:

Publish the CodeQL pack to a package registry that CodeQL can access is required because CodeQL packs are versioned, distributable packages. A pack must be published to a supported package registry before GitHub Actions runners can download and use its contents during CodeQL analysis. For GitHub-hosted pack distribution, organizations can publish packs to the GitHub Container Registry, using the package identity and a version tag to make a particular release available. This provides a centrally maintained query collection that can be consumed without duplicating query source files across repositories.

Reference the pack by its pack name and version or version range is also required. The advanced setup configuration uses the `packs` setting to identify the pack that CodeQL should download. A reference such as `OWNER/PACK@VERSION` identifies the package and selects an exact version or a compatible version range. During analysis, the CodeQL action resolves that reference, retrieves the published pack, and runs its included queries in addition to the configured query suite. GitHub documents the syntax and use of the `packs` setting in its CodeQL configuration-file guidance, and publishing workflow details are available in the CodeQL pack publishing documentation.

QUESTION NO: 8

When Dependabot alerts are enabled for a repository and GitHub detects a vulnerable dependency, what actions does GitHub take when alert and notification recipients use their default settings? Select two answers.

A. It generates a Dependabot alert and displays it on the Security tab for the repository.

B. It notifies the repository administrators about the new alert.

C. It generates Dependabot alerts by default for all private repositories.

D. It consults with a security service and conducts a thorough vulnerability review.

ANSWER: A B

Explanation:

GitHub creates a Dependabot alert when GitHub Advisory Database information matches a dependency in the repository with a known vulnerable version range. The alert is surfaced in the repository's Security area, where authorized users can inspect the affected manifest and dependency, review the associated security advisory and severity information, and determine whether an updated version or other remediation is available. This repository-level alert provides a trackable security finding that can be reviewed, dismissed when appropriate, or remediated.

Under the default notification configuration assumed in the question, GitHub also notifies repository administrators about newly created Dependabot alerts. Repository administrators are included in the default audience for Dependabot alert notifications because they can manage repository security settings and coordinate remediation. Actual delivery can still depend on a recipient's GitHub notification preferences and enabled delivery channels, but GitHub targets repository administrators by default when no custom alert-notification recipient configuration has been applied.

For details on how alerts are created and displayed, see About Dependabot alerts. Default notification recipients and notification configuration are described in Configuring notifications for Dependabot alerts.

QUESTION NO: 9

Which repository **Watch** notification settings can be used to receive notifications about Dependabot alerts? Each answer represents part of the solution. Choose two.

A. The Custom setting

B. The Participating and @mentions setting

C. The All Activity setting

D. The Ignore setting

ANSWER: A C

Explanation:

The Custom setting and the All Activity setting can both deliver Dependabot alert notifications. Dependabot alerts are security alerts that GitHub creates when the dependency graph identifies a dependency with a known vulnerability. A user who selects the Custom setting can explicitly subscribe to the repository's *Security alerts* notification category. This provides focused notification coverage for security events, including Dependabot alerts, without requiring notifications for every category of repository activity.

The All Activity setting also provides Dependabot alert notifications because it subscribes the watcher to all repository activity. This broad subscription includes security-related events alongside notifications for issues, pull requests, releases, and other repository activity. The appropriate choice depends on whether the user needs only targeted security notifications or comprehensive repository notifications. Notification delivery can also be affected by the repository's security features and the recipient's access and personal notification configuration. GitHub documents the available repository watch settings and their notification categories in Configuring notifications. For Dependabot-specific notification behavior, see Configuring notifications for Dependabot alerts.

QUESTION NO: 10

Where can CodeQL analysis be run to perform code scanning? Each correct answer represents part of the solution. Choose two.

A. In a third-party Git repository

B. In a workflow

C. In an external continuous integration (CI) system

D. In the Files changed tab of the pull request

ANSWER: B C

Explanation:

CodeQL analysis can run **in a workflow**, most commonly through a GitHub Actions workflow configured for code scanning. A workflow can initialize CodeQL for the repository's languages, build compiled code where required, run the analysis, and upload the resulting SARIF findings to GitHub. GitHub then uses those results to create and manage code-scanning alerts for the repository.

CodeQL analysis can also run **in an external continuous integration (CI) system**. This supports organizations that use CI platforms other than GitHub Actions or that need to integrate scanning into an existing build pipeline. The external system runs the CodeQL CLI to create a database and analyze it, then uploads the SARIF results to GitHub so that findings appear in the repository's code-scanning experience. This approach enables the same GitHub alerting and remediation workflow while allowing analysis to be executed by the organization's chosen CI infrastructure. For implementation guidance, see Using CodeQL code scanning with your existing CI system and About code scanning with CodeQL.

QUESTION NO: 11

A security team must choose between the default and extended CodeQL query suites for a repository. They want to detect additional potential security issues and can investigate findings that may have lower precision. Which query suite should the team use?

A. The default query suite

B. The extended query suite

C. A dependency review workflow

D. A custom secret scanning pattern

ANSWER: B

Explanation:

The extended query suite is the appropriate choice because it is designed for teams that want broader CodeQL coverage than the default suite provides. It includes the queries in the default suite and adds additional security, quality, and maintainability queries. These extra queries can surface a wider range of potential vulnerabilities and coding issues, including findings that are useful for proactive investigation but may require more manual triage to establish whether they represent an exploitable security risk in the repository's specific context.

This aligns directly with the team's goal: maximizing discovery of possible security issues while accepting that some alerts will not have the same high-confidence precision expected from the default-focused set. Selecting the extended suite therefore provides a practical balance when the team has sufficient capacity to review and validate a broader alert set. GitHub documents that the extended suite includes all default queries plus additional queries, and that its results can include lower-precision findings. See GitHub Docs: Specifying CodeQL queries to run and GitHub Docs: About code scanning with CodeQL.

QUESTION NO: 12

After receiving an alert that a pull request introduces a vulnerable dependency, what action should you take before merging the branch?

A. Disable Dependabot alerts for all repositories owned by your organization

B. Fork the branch and deploy the new fork

C. Update the vulnerable dependencies before the branch is merged

D. Deploy the code to your default branch

ANSWER: C

Explanation:

Update the vulnerable dependencies before the branch is merged. GitHub dependency review examines dependency changes proposed in pull requests and can identify when the change introduces a dependency with known vulnerabilities. Remediating the dependency during pull-request review prevents a known security issue from entering the repository's default branch and potentially being included in a build or release. The remediation should update the affected package to an available version that resolves the vulnerability, while preserving the project's dependency requirements. After making the update, review any associated release notes or compatibility considerations and run the project's relevant tests and validation workflows before approving and merging the pull request. This approach aligns security remediation with the normal development workflow, allowing the dependency change and its verification to be reviewed together. GitHub documents that dependency review provides vulnerability information for dependency changes in pull requests, helping maintainers prevent vulnerable dependencies from being added. GitHub's Dependabot alert guidance likewise recommends updating to a patched version when one is available. See GitHub Docs: About dependency review and GitHub Docs: Viewing and updating Dependabot alerts.

QUESTION NO: 13

Which operations are available as GitHub code-scanning REST API endpoints? Select two answers.

A. List all open code scanning alerts for the default branch

B. Modify the severity of an open code scanning alert

C. Get a single code scanning alert

D. Delete all open code scanning alerts

ANSWER: A C

Explanation:

List all open code scanning alerts for the default branch is supported by GitHub's repository code-scanning alerts listing operation. This REST operation can return alerts for a repository and supports filters including the alert state and a Git reference. An integration can request open alerts and scope the request to the repository's default branch reference, which is useful for reporting and triage workflows focused on findings currently affecting the primary development line.

Get a single code scanning alert is also a supported repository code-scanning operation. It retrieves the details of one alert by its alert number. The returned alert information can include the alert's current state, the security rule that produced the finding, the analysis tool, locations, and instances. This allows automation or a security-review workflow to inspect a particular finding after identifying it in an alerts listing. Access to these operations requires appropriate repository permissions and an authentication token with the required code-scanning alert permission. GitHub documents the available repository code-scanning REST operations, their endpoint paths, parameters, and token requirements in the REST API code-scanning documentation and the repository alert-listing reference.

QUESTION NO: 14

As a repository owner, configure a GitHub Actions workflow so that it runs for pull requests targeting the `main` branch, except when the pull request changes only Markdown or `.txt` files. Given the existing trigger configuration below, which three lines should be added? Each answer represents part of the solution.

```
on:
  pull_request:
    branches: [main]
```

A. - '**.md'

B. - '**.txt'

C. paths:

D. paths-ignore:

E. - 'docs/*.md'

ANSWER: A B D

Explanation:

The required configuration uses `paths-ignore:` within the `pull_request` event, together with the `'**.md'` and `'**.txt'` path patterns. The `branches: [main]` filter already limits this event to pull requests whose target (base) branch is `main`. Adding `paths-ignore` applies a file-path filter to those pull-request events. A workflow with these patterns will not run when every changed file matches one of the ignored patterns; it can still run if the pull request includes a changed file outside those patterns. The `**` wildcard makes the extension patterns apply throughout the repository, including files in nested directories, rather than limiting the filter to a particular folder. This is appropriate when the requirement covers any Markdown or text file, regardless of where it is stored. GitHub Actions supports `paths-ignore` for `pull_request` events and evaluates the changed files to determine whether a workflow should run. See GitHub Docs: path filters for pull requests and GitHub Docs: filter pattern cheat sheet.

QUESTION NO: 15

In which GitHub location can you locate a line of code containing a secret after that line has been deleted from the current version of a file?

A. Insights

B. Issues

C. Commits

D. Dependency graph

ANSWER: C

Explanation:

Commits is correct because Git repositories preserve historical snapshots of a project. Removing a line from the current version of a file changes the latest state, but it does not normally erase the line from commits that were created before the deletion. A person with appropriate repository access can review the commit history, open an earlier revision, or compare a commit's changes to find code that previously contained the secret value.

This persistence is an important security consideration: a secret that has been committed must be considered exposed even when it is deleted immediately afterward. The appropriate first response is to revoke or rotate the credential, token, or key so that it can no longer be used. If the sensitive content must also be removed from repository history, maintainers can use Git history-rewrite tools and follow GitHub's documented remediation process. GitHub explains that sensitive data can remain accessible through previous commits and provides guidance for removing it from history in Removing sensitive data from a repository. For details on inspecting repository history and commit changes, see Viewing and comparing commits.

QUESTION NO: 16

While configuring GitHub Advanced Security tools in GitHub Enterprise, which repository **Watch** notification settings can be used to receive Dependabot alert notifications? Each answer represents part of the solution. Choose two.

A. The Custom setting

B. The Participating and @mentions setting

C. The All Activity setting

D. The Ignore setting

ANSWER: A C

Explanation:

The **Custom setting** is appropriate because it lets a user select specific notification types for a watched repository. In the custom notification dialog, users can opt in to *Security alerts*, which includes notifications generated for Dependabot alerts. This is useful when a user needs awareness of dependency vulnerabilities but does not want every repository event, such as all issue or pull-request activity, delivered as a notification.

The **All Activity setting** also provides Dependabot alert notifications because it subscribes the user to all notification activity for the repository. Since security alerts are part of the repository notification categories covered by watching settings, a user watching all activity receives notifications for Dependabot alerts in addition to other repository events.

These settings control a user's personal notification subscription for a repository; they complement, rather than replace, the repository-level configuration that enables Dependabot alerts and notifications. GitHub documents that repository watching can be customized by notification type, including security alerts, and that Dependabot alerts notify relevant recipients when vulnerable dependencies are detected. See Configuring notifications and About Dependabot alerts.

QUESTION NO: 17

You have write access to a repository and are viewing a code scanning alert. Under what condition will GitHub automatically close the alert?

A. After you triage the pull request containing the alert

B. When you use data-flow analysis to find potential security issues in code

C. After you find the code and click the alert within the pull request

D. After the code fix is merged into the default branch and a code-scanning analysis no longer detects the alert

ANSWER: D

Explanation:

GitHub automatically closes a code scanning alert as fixed when a subsequent code-scanning analysis determines that the alert is no longer present in the repository's default branch. Therefore, the remediation must be merged into the default branch and a scan must successfully analyze the updated code without finding the relevant alert. A commit in an open pull request can show that a proposed fix exists, but the alert's lifecycle is ultimately tracked against the default branch and its latest analysis results. This behavior gives teams reliable remediation status: an alert is only marked fixed after the code that GitHub treats as the repository's current baseline no longer contains the detected vulnerability or coding pattern. GitHub may also close alerts when the relevant code is deleted or when analysis configuration changes mean the alert can no longer be found, but the standard developer remediation path is to merge a code fix and allow code scanning to run. Details on alert states and automatic closure are available in GitHub Docs: About code scanning alerts and GitHub Docs: Managing code scanning alerts for your repository.