PassQueen
a.k.a dumpsqueen.com

GitHub Administration

Microsoft GH-100

Version Demo

Total Demo Questions: 9

Total Premium Questions: 99

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, Manage user identities and access	14
Topic 2, Manage GitHub organizations	22
Topic 3, Manage GitHub Enterprise	24
Topic 4, Manage GitHub Actions	16
Topic 5, Manage GitHub security	23
Total	99

QUESTION NO: 1

When a user belongs to multiple GitHub organizations, which three administrative considerations are important? Choose three.

A. The user will automatically have the same role across all organizations.

B. The user's repository access and/or team membership needs to be managed separately for each organization.

C. The user will need to authorize credentials separately for each SAML-enabled organization.

D. The user will have different permission levels in each organization.

E. The user's profile information becomes private to non-organization members.

F. The user's personal repositories will become accessible to all organizations.

ANSWER: B C D

Explanation:

The user's repository access and/or team membership needs to be managed separately for each organization because organizations are distinct administrative and access-control boundaries. Team membership, repository permissions, and organization membership do not transfer from one organization to another. Administrators must therefore evaluate and maintain the member's appropriate access in each organization independently.

The user will need to authorize credentials separately for each SAML-enabled organization. When an organization enforces SAML single sign-on, authorization applies to that specific organization. A personal access token or SSH key used to access resources protected by SAML SSO must be authorized through the relevant organization's identity provider. This organization-specific authorization preserves separate identity controls for each organization. See Authorizing a personal access token for use with SAML SSO.

The user will have different permission levels in each organization because organization roles and repository roles are assigned independently. The same GitHub account can receive a different organization role, team assignment, or repository permission in every organization it joins. This enables each organization to apply its own governance and least-privilege requirements, as described in Roles in an organization.

QUESTION NO: 2

Which audit-log event types can be queried by using the GitHub GraphQL Audit Log API? Choose three answers.

A. changes in permissions

B. promoting users to administrators

C. pushes to repositories

D. changes to permissions of a GitHub App

E. cloning of repositories

ANSWER: A B D

Explanation:

changes in permissions, **promoting users to administrators**, and **changes to permissions of a GitHub App** are audit-log activities that describe security-sensitive changes to access and authorization within an organization or enterprise. The GraphQL Audit Log API exposes audit entries through the `auditLog` connection, enabling administrators to retrieve and filter recorded actions. Permission changes are significant because they can grant, remove, or alter a person's or team's ability to access repositories and organization resources. Administrative-role promotions are similarly recorded because an administrator receives elevated control over organizational settings, members, repositories, and security configuration. GitHub App permission changes are also tracked because an app's granted permissions determine what data and operations the integration can access on behalf of its installation.

These event categories support security review, compliance investigations, and monitoring of privilege changes. When querying audit data, administrators can use audit-log fields and qualifiers to identify the actor, action, timestamp, affected resource, and other relevant context. GitHub documents GraphQL-based audit-log queries for both organization and enterprise scopes in the GitHub GraphQL Audit Log API organization guide and the enterprise audit-log guide.

QUESTION NO: 3

As an enterprise administrator, you must configure the enterprise policy that controls whether private and internal repositories can be forked. Which three policy choices are available at the enterprise level?

A. Allow organization owners to administer the setting at the organization level.

B. Allow people who have access to private and internal repositories to fork these repositories.

C. Allow specific people or teams to fork private and internal repositories.

D. Disallow repository owners from administering the setting at the repository level.

E. Disallow forking of private and internal repositories.

ANSWER: A B E

Explanation:

At the enterprise level, GitHub provides three choices for governing forks of private and internal repositories. **Allow organization owners to administer the setting at the organization level.** delegates the decision to each organization, allowing organization owners to select the appropriate forking policy for repositories within their organization. This is useful when organizations in the same enterprise have different collaboration, security, or development requirements.

Allow people who have access to private and internal repositories to fork these repositories. establishes an enterprise-wide policy that permits authorized users to create forks of repositories they can access. Forks of private repositories remain private, and access to fork networks is governed by GitHub's permissions and visibility model.

Disallow forking of private and internal repositories. applies a restrictive enterprise-wide setting that prevents forks of those repository types across the enterprise. Enterprise owners configure this policy through enterprise repository management settings, either enforcing one consistent behavior or permitting organization-level administration. GitHub documents these available policy choices in Enforcing repository management policies in your enterprise. For fork visibility and permission behavior, see About permissions and visibility of forks.

QUESTION NO: 4 - (SIMULATION)

How is CodeQL different from other static analysis tools?

A Itremoves insecure code automatically

B.
It allows querying of code semantics using a database-like language.

C.
It only works for open-source projects.

D.
It runs analysis only after a security breach.

ANSWER: See the explanation for the answer

Explanation:

Correct answer: B. It allows querying of code semantics using a database-like language.

CodeQL treats source code as data by creating a database representation of the code. You can then use the CodeQL (QL) query language to query code structure, control flow, data flow, and semantics to identify vulnerabilities and other patterns.

A is incorrect: CodeQL identifies issues; it does not automatically remove insecure code.
C is incorrect: CodeQL is not limited to open-source projects.
D is incorrect: CodeQL is proactive static analysis and can run during development or CI/CD workflows.

QUESTION NO: 5

An organization has a parent team named Engineering and a child team named Engineering-API. If the Engineering team has Read access to a repository, what repository access does the Engineering-API child team receive through team nesting?

A. Read access inherited from the Engineering parent team

B. Write access because it is a child team

C. No repository access until it is granted directly

D. Admin access inherited from the organization owners

ANSWER: A

Explanation:

Read access inherited from the Engineering parent team is correct. GitHub nested teams are designed to make permission management scalable: administrators can grant a repository role to a broader parent team, then organize members into more specialized child teams without having to repeat the same repository assignment. When Engineering is granted the Read role for a repository, Engineering-API inherits that repository access through the parent-child team relationship. Members who receive access through Engineering-API can therefore view and clone the repository, subject to the capabilities of GitHub's Read repository role.

This inheritance model supports clear organizational structures while retaining centralized control of baseline access. The parent team can represent a wider engineering function, while the child team represents a focused API group that may later receive additional access to other repositories as needed. GitHub documents that child teams inherit the repository access permissions of their parent teams, and that nested teams can also inherit parent-team mentions. For details, see GitHub Docs: Nested teams and GitHub Docs: Repository roles for an organization.

QUESTION NO: 6

Why is a GitHub App generally preferred to a personal access token (PAT) for authenticating automated machines and integrations?

A. GitHub Apps are required to pass SAML assertions

B. GitHub Apps have time-limited installation tokens with scoped access

C. PATs cannot be used in GitHub Actions

D. PATs support fewer GitHub APIs than Apps

ANSWER: B

Explanation:

GitHub Apps have time-limited installation tokens with scoped access because they provide an identity specifically intended for an integration or automation workload, rather than using credentials associated with an individual user. An administrator can install an app only on selected repositories and grant only the permissions needed for its function. For example, an automation service can be authorized to read repository contents or manage issues without being granted unrelated permissions. This supports least-privilege access and gives organizations a clearer, more manageable way to govern an integration's repository access.

After the app authenticates, it requests an installation access token for a particular installation. GitHub states that installation access tokens expire after one hour. The short-lived token is limited by the app installation's authorized repositories and configured permissions, which reduces the risk and operational burden of relying on a broadly usable, long-lived machine credential. This model also allows access to be changed centrally by adjusting the GitHub App installation or its permissions as the integration evolves. For GitHub's implementation details, see Authenticating as a GitHub App installation and Deciding when to build a GitHub App.

QUESTION NO: 7

Which command should you run from your local computer to generate and save a support bundle for the GitHub Enterprise Server instance named `ghe.avocado.corp`?

A. ssh -p 122 admin@ghe.avocado.corp -- 'ghe-support-bundle -o' > support-bundle.tgz

B. ssh -p 122 adming@ghe.avocado.corp – 'ghe-diagnostics' > support-bundle.tgz

C. curl -u admin https://ghe.avocado.corp/diagnostics/support-bundle.tgz -o

D. ssh -p 122 adming@ghe.avocado.corp -- 'ghe-config generate-support-bundle' > support-bundle.tgz

ANSWER: A

Explanation:

`ssh -p 122 admin@ghe.avocado.corp -- 'ghe-support-bundle -o' > support-bundle.tgz` is the documented command pattern for creating a GitHub Enterprise Server support bundle and downloading it to the computer on which the command is run. GitHub Enterprise Server provides its administrative shell through SSH on port `122`. Connecting with the `admin` account allows an appliance administrator to invoke the built-in `ghe-support-bundle` utility. The `-o` switch writes the generated bundle to standard output, and the local shell redirection operator saves that output as `support-bundle.tgz`. This provides a portable compressed archive that can be supplied to GitHub Support for troubleshooting system health, configuration, and operational issues. Because a support bundle can include logs and diagnostic details from the appliance, it should be stored and transferred according to the organization's security procedures. GitHub documents this exact SSH-based workflow for generating the archive locally; see Creating a support bundle. Details about the administrative SSH shell and its required port are available in Accessing the administrative shell (SSH).

QUESTION NO: 8

An organization needs a security specialist to review and manage code scanning, Dependabot, and secret scanning alerts across all organizational repositories, without granting broad organization-administration permissions. Which organization role should be assigned?

A. Organization owner

B. Security manager

C. Team maintainer

D. Repository administrator

ANSWER: B

Explanation:

Security manager is the appropriate role because it is specifically designed for centralized management of an organization's security posture without granting the extensive administrative capabilities of an organization owner. Organization owners can assign this role to individuals or teams that need to work with security findings across the organization. A security manager has the access needed to view and manage security alerts, including alerts produced by code scanning, Dependabot, and secret scanning, across repositories in the organization. This enables the specialist to triage findings, investigate affected repositories, coordinate remediation, and oversee security-alert workflows at organizational scope. The role also provides read access to repositories in the organization, including private repositories,

when required to support that security-review responsibility. Importantly, the role follows least-privilege practice: it provides security-focused access rather than general control over organization settings, membership, billing, or other owner-level administrative functions. GitHub documents the security manager role and its permissions in its guidance on using organization roles. For details about organization-wide security alert management, see GitHub's code security documentation.

QUESTION NO: 9

An organization uses proprietary token formats that are not included in GitHub's supported secret-scanning patterns. The security team needs GitHub to identify these tokens and block new exposures in selected repositories. Which two actions should administrators take?

A. Create a custom secret scanning pattern that matches the proprietary token format.

B. Enable push protection for the custom pattern in the selected repositories.

C. Add the token format to `.github/dependabot.yml`.

D. Create a repository security policy that lists the token format.

E. Configure a CODEOWNERS rule for files that might contain the tokens.

ANSWER: A B

Explanation:

Administrators should create a custom secret scanning pattern that matches the proprietary token format. GitHub custom patterns use regular expressions and optional additional requirements, such as a required prefix or a checksum, to identify secrets that are specific to an organization. This extends secret scanning beyond GitHub's predefined partner, non-provider, and generic secret patterns, allowing the organization to detect its own credentials in repository content.

They should also enable push protection for the custom pattern in the selected repositories. Once push protection is enabled for a custom pattern, GitHub scans commits before they are accepted and blocks a push that contains a detected match. This provides preventive protection rather than merely creating an alert after the secret has been committed. Administrators can scope enablement to the repositories that handle the proprietary format, matching the stated requirement. GitHub documents how to define, test, and publish organization-level custom patterns at Defining custom patterns for secret scanning. For the preventive control and its repository configuration, see Protecting pushes with secret scanning.