HashiCorp

HCVA0-003 Exam

HashiCorp Certified: Vault Associate (003) Exam

Questions & Answers
Demo

Version: 4.0

Topic 1, Exam Pool A

Question: 1

Select the policies below that permit you to create a new entry of environment=prod at the path /secrets/apps/my_secret (select three).

A. path "secrets/+/my_secret" { capabilities = ["create"] allowed_parameters = { "*" = [] } }
B. path "secrets/apps/my_secret" { capabilities = ["update"] }
C. path "secrets/apps/my_secret" { capabilities = ["create"] allowed_parameters = { "environment" = [] } }
D. path "secrets/apps/*" { capabilities = ["create"] allowed_parameters = { "environment" = ["dev", "test", "qa", "prod"] } }

Answer: A, C, D

Explanation:
Comprehensive and Detailed in Depth
This question requires identifying Vault policies that allow creating a new entry with environment=prod at the specific path /secrets/apps/my_secret. Vault policies define permissions using paths, capabilities, and parameter constraints. Let's evaluate each option:
Option A: path "secrets/+/my_secret" { capabilities = ["create"] allowed_parameters = { "*" = [] } }
The + wildcard matches any single segment in the path, so this policy applies to /secrets/apps/my_secret. The create capability permits creating new entries at this path. The allowed_parameters = { "*" = [] } means any parameter (including environment) can be set to any value. This satisfies the requirement to create an entry with environment=prod. Thus, this policy is correct.
Option B: path "secrets/apps/my_secret" { capabilities = ["update"] }
This policy targets the exact path /secrets/apps/my_secret but only grants the update capability. According to Vault's documentation, update allows modifying existing entries, not creating new ones. Since the question specifies creating a new entry, this policy does not meet the requirement and is incorrect.
Option C: path "secrets/apps/my_secret" { capabilities = ["create"] allowed_parameters = { "environment" = [] } }
This policy explicitly matches /secrets/apps/my_secret and grants the create capability, which allows new entries to be written. The allowed_parameters = { "environment" = [] } specifies that the environment parameter can take any value (an empty list means no restriction on values). This permits setting environment=prod, making this policy correct.
Option D: path "secrets/apps/*" { capabilities = ["create"] allowed_parameters = { "environment" = ["dev", "test", "qa", "prod"] } }
The * wildcard matches any path under secrets/apps/, including /secrets/apps/my_secret. The create

capability allows new entries, and the allowed_parameters restricts environment to dev, test, qa, or prod. Since prod is an allowed value, this policy permits creating an entry with environment=prod and is correct.
Overall Explanation from Vault Docs:
Vault policies control access via paths and capabilities (create, read, update, delete, list). The create capability is required to write new data. Parameter constraints (allowed_parameters) further restrict what key-value pairs can be written. An empty list ([]) allows any value, while a populated list restricts values to those specified. A deny takes precedence over any allow, but no deny is present here.
Reference: https://developer.hashicorp.com/vault/docs/concepts/policies#parameter-constraints

Question: 2

Below is a list of parent and child tokens and their associated TTL. Which token(s) will be revoked first?

A. ├──hvs.y4fUERqCtUV0xsQjWLJar5qX - TTL: 4 hours
B. ├──hvs.FNiIFU14RUxxUYAl4ErLfPVR - TTL: 6 hours
C. ├──hvs.Jw9LMpu7oCQgxiKbjfyzyg75 - TTL: 4 hours (child of B)
D. ├──hvs.3IrlhEvcerEGbae11YQf9FvI - TTL: 3 hours
E. ├──hvs.hOpweMVFvqfvoVnNgvZq8jLS - TTL: 5 hours (child of D)

Answer: D

Explanation:
Comprehensive and Detailed in Depth
Vault tokens have a Time-To-Live (TTL) that determines their expiration time, after which they are revoked. Parent-child relationships mean that revoking a parent token also revokes its children, regardless of their TTLs. Let's analyze:
A: TTL 4 hours - Expires after 4 hours, no children listed.
B: TTL 6 hours - Expires after 6 hours, parent to C.
C: TTL 4 hours (child of B) - Expires after 4 hours or if B is revoked earlier.
D: TTL 3 hours - Expires after 3 hours, parent to E.
E: TTL 5 hours (child of D) - Expires after 5 hours or if D is revoked earlier.
Analysis:
Shortest TTL is D (3 hours), so it expires first unless a parent above it (none listed) is revoked sooner.
E (5 hours) is a child of D. If D is revoked at 3 hours, E is also revoked, despite its longer TTL.
A and C (4 hours) expire after D.
B (6 hours) expires last among parents.
The question asks which token(s) are revoked first based on TTL alone, not manual revocation. D has the shortest TTL (3 hours) and will be revoked first. E's revocation depends on D, but the question focuses on initial expiration. Thus, only D is revoked first based on its TTL.
Overall Explanation from Vault Docs:
Tokens form a hierarchy where child tokens inherit revocation from their parents. "When a parent token is revoked, all of its child tokens—and all of their leases—are revoked as well." TTL dictates automatic expiration unless overridden by manual revocation or parent revocation. Here, D's 3-hour TTL is the

shortest, making it the first to expire naturally.
Reference: https://developer.hashicorp.com/vault/docs/concepts/tokens#token-hierarchies-and-orphan-tokens

Question: 3

Your company's security policies require that all encryption keys must be rotated at least once per year. After using the Transit secrets engine for a year, the Vault admin issues the proper command to rotate the key named ecommerce that was used to encrypt your dat
a. What command can be used to easily re-encrypt the original data with the new version of the key?
A. vault write -f transit/keys/ecommerce/rotate <old data>
B. vault write -f transit/keys/ecommerce/update <old data>
C. vault write transit/encrypt/ecommerce v1:v2 <old data>
D. vault write transit/rewrap/ecommerce ciphertext=<old data>

Answer: D

Explanation:
Comprehensive and Detailed in Depth
The Transit secrets engine in Vault manages encryption keys and supports key rotation. After rotating the ecommerce key, existing ciphertext (encrypted with the old key version) must be re-encrypted (rewrapped) with the new key version without exposing plaintext. Let's evaluate:
A: vault write -f transit/keys/ecommerce/rotate <old data>
This command rotates the key, creating a new version, but does not re-encrypt existing data. It's for key management, not data rewrapping. Incorrect.
B: vault write -f transit/keys/ecommerce/update <old data>
There's no update endpoint in Transit for re-encrypting data. This is invalid and incorrect.
C: vault write transit/encrypt/ecommerce v1:v2 <old data>
The transit/encrypt endpoint encrypts new plaintext, not existing ciphertext. The v1:v2 syntax is invalid. Incorrect.
D: vault write transit/rewrap/ecommerce ciphertext=<old data>
The transit/rewrap endpoint takes existing ciphertext, decrypts it with the old key version, and re-encrypts it with the latest key version (post-rotation). This is the correct command. For example, if <old data> is vault:v1:cZNHVx+..., the output might be vault:v2:kChHZ9w4....
Overall Explanation from Vault Docs:
"Vault's Transit secrets engine supports key rotation… The rewrap endpoint allows ciphertext encrypted with an older key version to be re-encrypted with the latest key version without exposing the plaintext." This operation is secure and efficient, using the keyring internally.
Reference: https://developer.hashicorp.com/vault/tutorials/encryption-as-a-service/eaas-transit-rewrap

Question: 4

From the unseal options listed below, select the options you can use if you're deploying Vault on-premises (select four).

A. Certificates
B. Transit
C. AWS KMS
D. HSM PKCS11
E. Key shards

Answer: B, C, D, E

Explanation:
Comprehensive and Detailed in Depth
Vault requires unsealing to access encrypted data, and on-premises deployments support various unseal mechanisms. Let's assess:
A: Certificates
Certificates secure communication (e.g., TLS), not unsealing. Vault's seal/unseal process uses cryptographic keys, not certificates. Incorrect.
B: Transit
The Transit secrets engine can auto-unseal Vault by managing encryption keys internally. Ideal for on-premises setups avoiding external services. Correct.
C: AWS KMS
AWS KMS can auto-unseal Vault if the on-premises cluster has internet access to AWS APIs. Common in hybrid setups. Correct.
D: HSM PKCS11
Hardware Security Modules (HSM) with PKCS11 support secure key storage and auto-unsealing on-premises. Correct.
E: Key shards
Shamir's Secret Sharing splits the master key into shards, the default manual unseal method for all Vault clusters. Correct.
Overall Explanation from Vault Docs:
"Vault supports multiple seal types… Key shards (Shamir) is the default… Auto-unseal options like Transit, AWS KMS, and HSM (PKCS11) are viable for on-premises if configured with access to required services."
Certificates are not an unseal mechanism.
Reference: https://developer.hashicorp.com/vault/docs/configuration/seal

Question: 5

How does the Vault Secrets Operator (VSO) assist in integrating Kubernetes-based workloads with Vault?

A. By enabling a local API endpoint to allow the workload to make requests directly from the VSO
B. By using client-side caching for KVv1 and KVv2 secrets engines
C. By injecting a Vault Agent directly into the pod requesting secrets from Vault
D. By watching for changes to its supported set of Custom Resource Definitions (CRD)

Answer: D

Explanation:
Comprehensive and Detailed in Depth

The Vault Secrets Operator (VSO) integrates Kubernetes workloads with Vault by syncing secrets. Let's evaluate:
A: VSO doesn't create a local API endpoint for direct requests; it syncs secrets to Kubernetes Secrets. Incorrect.
B: Client-side caching is a Vault Agent feature, not VSO's primary function. VSO can use caching, but it's not the main integration method. Incorrect.
C: VSO doesn't inject Vault Agents; that's a separate Vault Agent Sidecar approach. Incorrect.
D: VSO watches Custom Resource Definitions (CRDs) to sync Vault secrets to Kubernetes Secrets dynamically. This is its core mechanism. Correct.
Overall Explanation from Vault Docs:
"VSO operates by watching for changes to its supported set of CRDs… It synchronizes secrets from Vault to Kubernetes Secrets, ensuring applications access them natively."
Reference: https://developer.hashicorp.com/vault/docs/platform/k8s/vso