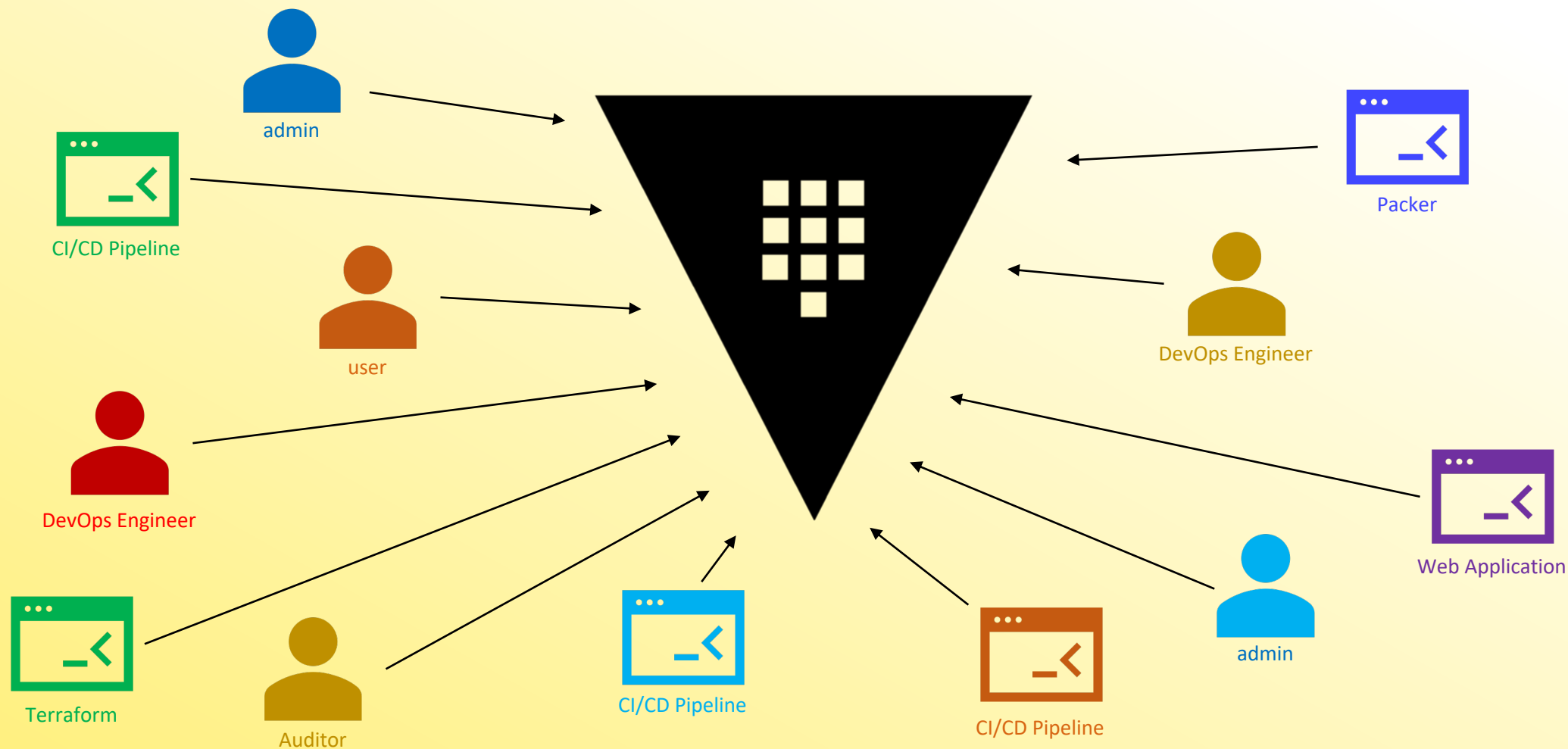




Vault Policies



How Do We Determine Who Should Access Secrets



Vault Policies



- Vault policies provide operators a way to permit or deny access to certain paths or actions within Vault (RBAC)
 - Gives us the ability to provide granular control over who gets access to secrets
- Policies are written in declarative statements and can be written using JSON or HCL
- When writing policies, always follow the principal of least privilege
 - In other words, give users/applications only the permissions they need



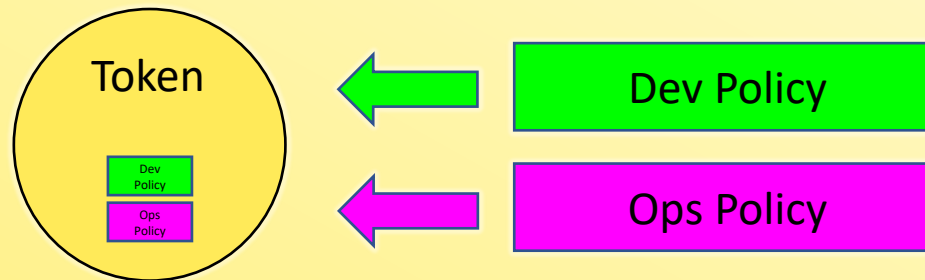
Vault Policies



- Policies are Deny by Default (implicit deny) - therefore you must explicitly grant to paths and related capabilities to Vault clients

No policy = no authorization

- Policies support an explicit DENY that takes precedence over any other permission
- Policies are attached to a token. A token can have multiple policies
 - Policies are cumulative and capabilities are additive



Out of the Box Policies



- **root** policy is created by default – superuser with all permissions
 - You cannot change nor delete this policy
 - Attached to all root tokens
- **default** policy is created by default – provides common permissions
 - You can change this policy, but it cannot be deleted
 - Attached to all non-root tokens by default (can be removed if needed)



Out of the Box Policies

```
Terminal
$ vault policy list
default
root
```

```
Terminal
$ vault policy read root
No policy named: root
```

```
Terminal
$ vault policy read default

# Allow tokens to look up their own properties
path "auth/token/lookup-self" {
  capabilities = ["read"]
}

# Allow tokens to renew themselves
path "auth/token/renew-self" {
  capabilities = ["update"]
}

# Allow tokens to revoke themselves
path "auth/token/revoke-self" {
  capabilities = ["update"]
}

# Allow a token to look up its own capabilities on a path
path "sys/capabilities-self" {
  capabilities = ["update"]
}

.....
```



Out of the Box Policies



The root policy does not contain any rules but can do anything within Vault. It should be used with extreme care.

```
path "*" {  
  capabilities = ["read", "create", "update", "delete", "list", "sudo"]  
}
```



If it *did* have rules, it would probably look something like this....





Managing Policies Using the CLI



Managing Policies in Vault

Command Line Interface (CLI)



Use the `vault policy` command

- `delete`
- `fmt`
- `list`
- `read`
- `write`

Terminal

```
$ vault policy list  
admin-policy  
default  
root
```

Terminal

```
$ vault policy write admin-policy /tmp/admin.hcl  
Success! Uploaded policy: admin-policy
```



Managing Policies in Vault

Command Line Interface (CLI)



```
vault policy write webapp /tmp/webapp.hcl
```

Type of Vault
object you want
to work with

Subcommand

Define the name
of the policy you
want to create

The location of the file
containing the pre-written
policy



Managing Policies in Vault

Command Line Interface (CLI)



TERMINAL

```
$ vault policy write webapp -<< EOF
path "kv/data/apps/*" {
  capabilities = ["read","create","update","delete"]
}
path "kv/metadata/*" {
  capabilities = ["read","create","update","list"]
}
EOF
```





Managing Policies Using the UI

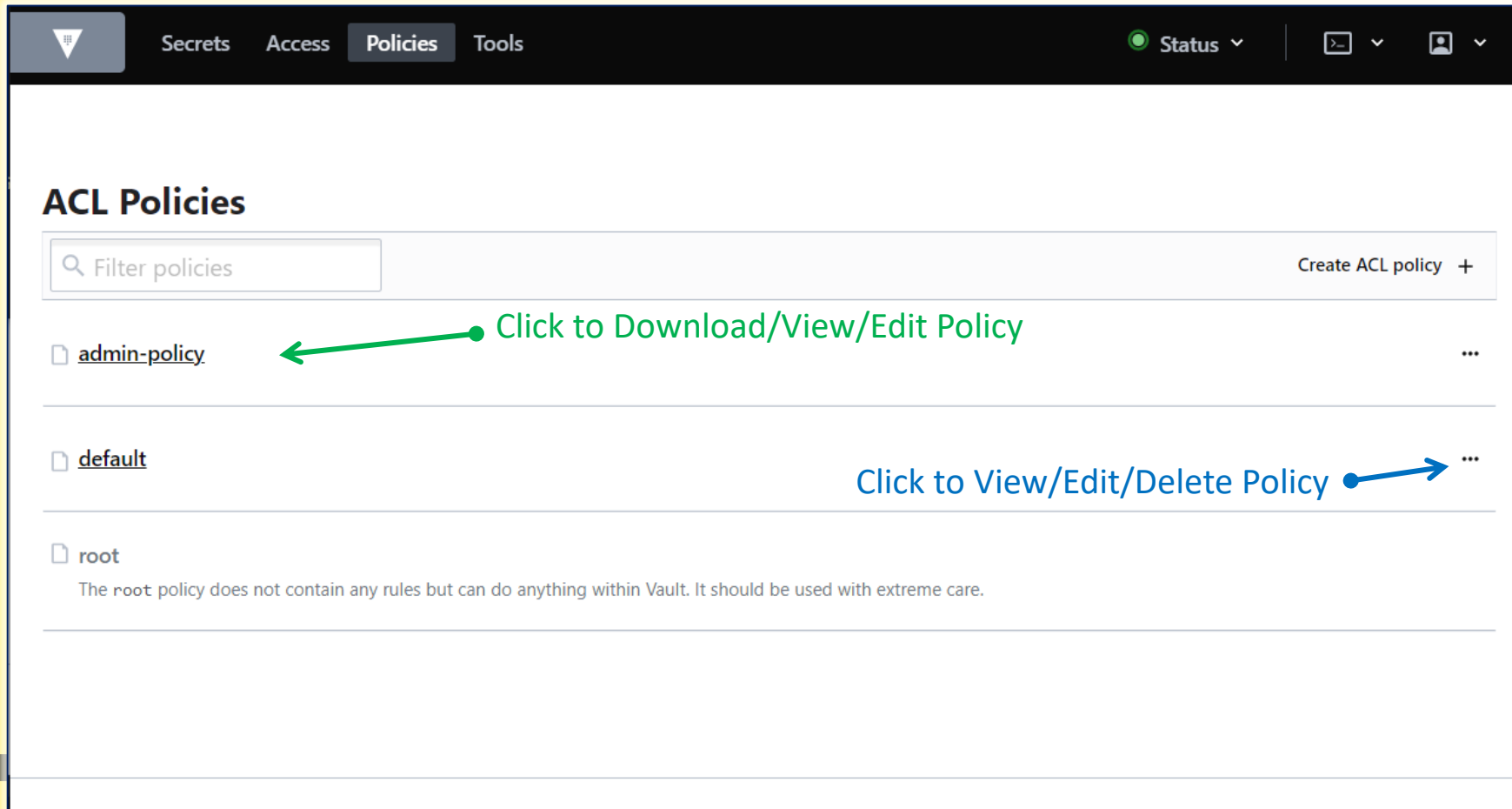


Managing Policies in Vault

User Interface (UI)



Create a New Policy





Managing Policies Using the API



Managing Policies in Vault

HTTP API



Creating a new Vault policy

- Method: POST

Don't forget you need
a valid token

Terminal

```
$ curl \
  --header "X-Vault-Token: hvs.bCEo8HFNIIR8wRGazwUk" \
  --request PUT \
  --data @payload.json \
  http://127.0.0.1:8200/v1/sys/policy/webapp
```

Create
Vault
Policy

API Endpoint

Name of the new policy



Managing Policies in Vault

HTTP API



Payload File:

```
payload.json
{
  "policy": "path \"kv/apps/webapp\" { capabilities... "
}
```





Anatomy of a Vault Policy



Anatomy of a Vault Policy



- Remember: Everything in Vault is path based
 - Policies grant or forbid access to those paths and operations

Two key parts to a Vault policy:

```
path "<path>" {  
    capabilities = ["<list of permissions>"]  
}
```



Anatomy of a Vault Policy



```
path "<path>" {  
    capabilities = ["<list of permissions>"]  
}  
path "<path>" {  
    capabilities = ["<list of permissions>"]  
}  
path "<path>" {  
    capabilities = ["<list of permissions>"]  
}
```



Anatomy of a Vault Policy



```
path "kv\data\apps\jenkins" {
  capabilities = ["read","update","delete"]
}
path "sys/policies/*" {
  capabilities = ["create","update","list","delete"]
}
path "aws/creds/web-app" {
  capabilities = ["read"]
}
```





Vault Polices - Path



Vault Policies - Path



- Path: we already know what a path is
 - see Vault Architecture and Pathing Structure in Section 1 for a review
- Examples of paths:
 - sys/policy/vault-admin
 - kv/apps/app01/web
 - auth/ldap/group/developers
 - database/creds/prod-db
 - secrets/data/platform/aws/tools/ansible/app01
 - sys/rekey



The Details are in the Path



Path of an Object

secrets/data/platform/aws/tools/ansible

Path where the
secrets engine
is mounted

Required
for a KV v2
secrets
engine

Higher-Level Paths
(data could be stored at each
one if needed)

Where the
key/value pairs are
stored and
retrieved



Vault Policies - Path



- Root-Protected Paths
 - Many paths in Vault require a root token or sudo capability to use
 - These paths focus on important/critical paths for Vault or plugins
- Examples of root-protected paths:
 - `auth/token/create-orphan` (create an orphan token)
 - `pki/root/sign-self-issued` (sign a self-issued certificate)
 - `sys/rotate` (rotate the encryption key)
 - `sys/seal` (manually seal Vault)
 - `sys/step-down` (force the leader to give up active status)



Vault Policies - Path



- Examples of root-protected paths:
 - sys/rotate (rotate the encryption key)
 - sys/seal (manually seal Vault)
 - sys/step-down (force the leader to give up active status)

```
admin-policy.hcl

path "sys/rotate" {
  capabilities = ["sudo"]
}
path "sys/seal" {
  capabilities = ["sudo"]
}
path "sys/step-down" {
  capabilities = ["sudo"]
}
```





Vault Polices - Capabilities



Vault Policies - Capabilities



- Capabilities define what can we do?
 - Capabilities are specified as a list of strings (yes, even if there's just one)

<u>Capability</u>	<u>HTTP Verb</u>
create	POST/PUT
read	GET
update	POST/PUT
delete	DELETE
list	LIST

<u>Capability</u>	<u>Description</u>
sudo	Allows access to paths that are root-protected
deny	Disallows access regardless of any other defined capabilities

create = if the key does not yet exist

update = if the key exists and you want to replace/update it



Vault Policies - Capabilities



- **Create** – create a new entry
- **Read** – read credentials, configurations, etc
- **Update** – overwrite the existing value of a secret or configuration
- **Delete** – delete something
- **List** – view what's there (doesn't allow you to read)
- **Sudo** – used for root-protected paths
- **Deny** – deny access – always takes precedence over any other capability

Note: Write is not a valid capability



Vault Policy - Example



Requirement:

- Access to generate database credentials at [database/creds/db01](#)
- Create, Update, Read, and Delete secrets stored at [kv/apps/dev-app01](#)

```
path "database/creds/dev-db01" {  
    capabilities = ["read"]  
}  
path "kv/apps/dev-app01" {  
    capabilities = ["create", "read", "update", "delete"]  
}
```

One Policy
With
Multiple Rules



Vault Policy - Example

Requirements:

- Access to read credentials after the path `kv/apps/webapp`
- Deny access to `kv/apps/webapp/super-secret`

```
path "kv/apps/webapp/*" {  
    capabilities = ["read"]  
}  
path "kv/apps/webapp/super_secret" {  
    capabilities = ["deny"]  
}
```

Tree

```
--kv  
  |--apps  
    |--webapp  
      |--super_secret ❌  
      |--api_token ✅  
      |--host_name ✅  
    |--mid-tier  
    |--database  
  |--cloud  
    |--aws  
      |--prod  
    |--gcp  
  |--dev
```

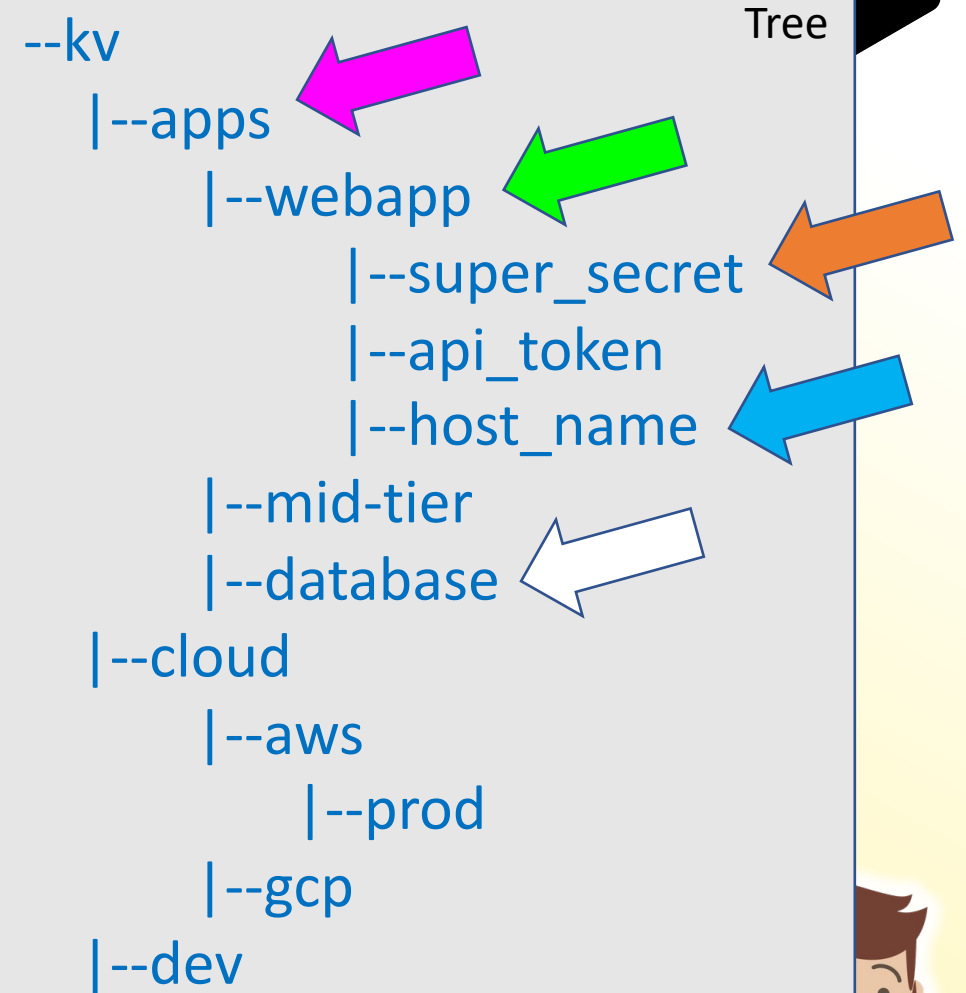
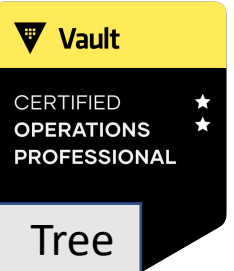


Pop Quiz

Q: Does this policy permit access to kv/apps/webapp?

A: No, because the policy only permits access to secrets AFTER kv/apps/webapp

```
path "kv/apps/webapp/*" {  
  capabilities = ["read"]  
}  
path "kv/apps/webapp/super_secret" {  
  capabilities = ["deny"]  
}
```



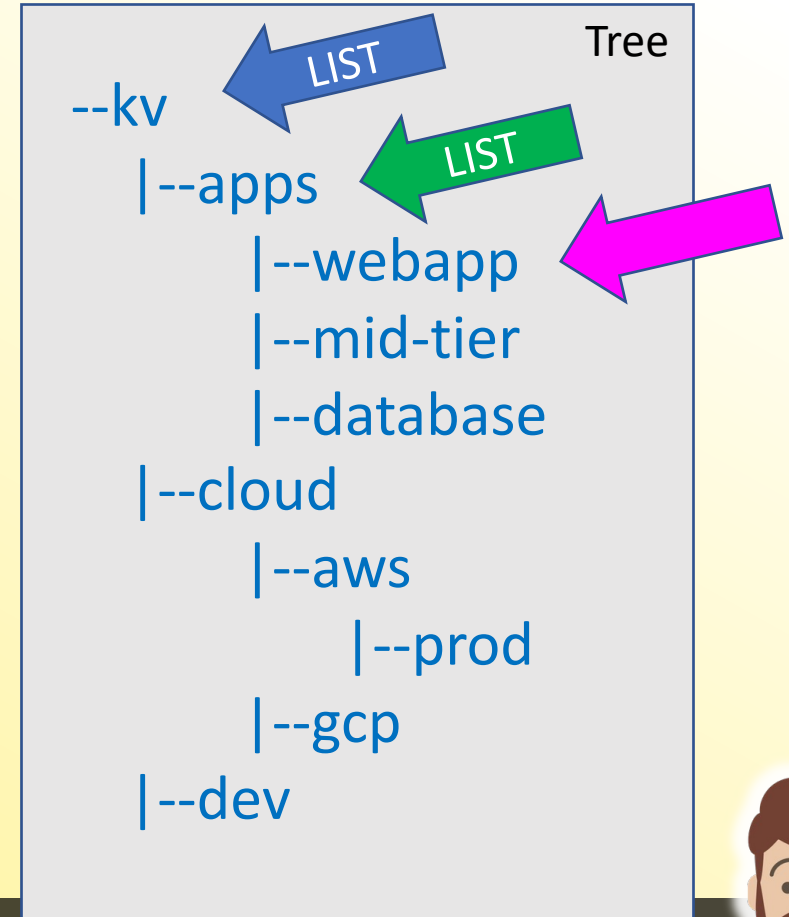
Pop Quiz



Q: Does this policy permit you to browse to kv/apps/webapp in the UI?

A: No, because the policy only permits list at the listed path, not the paths leading up to the desired path

```
path "kv/apps/webapp/*" {  
  capabilities = ["read", "list"]  
}
```





Customizing the Path



Using the * to Customize the Path



- The glob (*) is a wildcard and can only be used at the end of a path
- Can be used to signify anything "after" a path or as part of a pattern
- Examples:
 - secret/apps/application1/* - allows any path after application1
 - kv/platform/db-* - would match kv/platform/db-2 but not kv/platform/db2



The Details are in the Path



secret/apps/application1/*

Path where the
secrets engine
is mounted

Path created on the secrets engine
called secret

Apply capabilities on
anything AFTER
application1



Does it Match?



`secret/apps/application1/*`

Path must start with this – nothing else

Must ALSO include something beyond application1

Paths that Match

- ✓ `secret/apps/application1/db`
- ✓ `secret/apps/application1/data/production`
- ✓ `secret/apps/application1/web-app`
- ✓ `secret/apps/application1/keys/api_key`

Paths that Do Not Match


- X `secret/apps/database`
- X `secret/apps/application2`
- X `secret/data/front-end`
- X `kv/secret/app/application`




Pop Quiz



Given the policy:



```
path "secret/apps/application1/*" {  
  capabilities = ["read"]  
}
```



required

Can I read from the following path?

secret/apps/application1

Answer: No, because the policy only permits read access for anything AFTER application1, not the path secret/apps/application1 itself



Pop Quiz



If we wanted to ALSO read from `secret/apps/application1`, the policy would look like this:

```
path "secret/apps/application1/*" {  
  capabilities = ["read"]  
}
```

```
path "secret/apps/application1" {  
  capabilities = ["read"]  
}
```



NEW



Using the + to Customize the Path



- The plus (+) supports wildcard matching for a single directory in the path
- Can be used in multiple path segments (i.e., secret/+/+/db)
- Examples:
 - secret/+/db - matches secret/db2/db or secret/app/db
 - kv/data/apps/+/webapp – matches the following:
 - kv/data/apps/dev/webapp
 - kv/data/apps/qa/webapp
 - kv/data/apps/prod/webapp



The Details are in the Path



secret/data/+ /apps/webapp

Path where the
secrets engine
is mounted

Used for KV V2
Secrets Engine

Can be ANY
value

Remaining path



Does it Match?



secret/data/+/apps/webapp

Path must start with
this – nothing else

Can be
ANY value

Path must end with
this – nothing else

Paths that Match

- ✓ secret/data/production/apps/webapp
- ✓ secret/data/dev1/apps/webapp
- ✓ secret/data/team-abc/apps/webapp
- ✓ secret/data/456/apps/webapp

Paths that Do Not Match

- X secret/data/apps/webapp
- X secret/app123/dev
- X secret/data/front-end/apps
- X secret/dev/apps/webapp



Example Policy



Using multiple + in a policy

```
path "secret/+//webapp" {  
  capabilities = ["read", "list"]  
}
```

```
path "secret/apps/+/team-*" {  
  capabilities = ["create", "read"]  
}
```

Combining the * and + in a policy



ACL Templating



- Use variable replacement in some policy strings with values available to the token
- Define policy paths containing double curly braces: `{{<parameter>}}`

Example: Creates a section of the key/value v2 secret engine to a specific user

```
path "secret/data/{{identity.entity.id}}/*" {
    capabilities = ["create", "update", "read", "delete"]
}

path "secret/metadata/{{identity.entity.id}}/*" {
    capabilities = ["list"]
}
```



ACL Templating



Parameter	Description
<code>identity.entity.id</code>	The entity's ID
<code>identity.entity.name</code>	The entity's name
<code>identity.entity.metadata.<<metadata key>></code>	Metadata associated with the entity for the given key
<code>identity.entity.aliases.<<mount accessor>>.id</code>	Entity alias ID for the given mount
<code>identity.entity.aliases.<<mount accessor>>.name</code>	Entity alias name for the given mount
<code>identity.entity.aliases.<<mount accessor>>.metadata.<<metadata key>></code>	Metadata associated with the alias for the given mount and metadata key
<code>identity.groups.ids.<<group id>>.name</code>	The group name for the given group ID
<code>identity.groups.names.<<group name>>.id</code>	The group ID for the given group name
<code>identity.groups.names.<<group id>>.metadata.<<metadata key>></code>	Metadata associated with the group for the given key
<code>identity.groups.names.<<group name>>.metadata.<<metadata key>></code>	Metadata associated with the group for the given key





Working with Policies



What Policies are Attached?



Create a new token with "web-app" policy attached:

```
$ vault token create -policy="web-app"
```

Key	Value
token	s.7uBlZwXSxOg31uGXIUetEdXD
token_accessor	18r88muoe3x1xEqVqXdlTMwJ
token_duration	768h
token_renewable	true
token_policies	["default" "web-app"]
identity_policies	[]
token_policies	[default web-app]

Every token gets the **default** policy plus the assigned policy or policies



Testing Policies



Test to make sure the policy fulfills the requirements

Example Requirements:

- Clients must be able to request AWS credential granting read access to a S3 bucket
- Read secrets from secret/apikey/Google

TERMINAL

```
$ vault token create -policy="web-app"

# Authenticate with the newly generated token
$ vault login <token>

# Make sure that the token can read
$ vault read secret/apikey/Google

# This should fail
$ vault write secret/apikey/Google key="ABCDE12345"

# Request a new AWS credentials
$ vault read aws/creds/s3-readonly
```

Administrative Policies



- Permissions for Vault backend functions live at the `sys/` path
- Users/admins will need policies that define what they can do within Vault to administer Vault itself
 - Unsealing
 - Changing policies
 - Adding secret backends
 - Configuring database configurations



Administrative Policies

Licensing

Setup New Vault Cluster

Configure UI

Rotate Keys

Seal Vault

```
# Configure License
path "sys/license" {
  capabilities = ["read", "list", "create", "update", "delete"]
}
# Initialize Vault
path "sys/init" {
  capabilities = ["read", "update", "create"]
}
# Configure UI in Vault
path "sys/config/ui" {
  capabilities = ["read", "list", "update", "delete", "sudo"]
}
# Allow rekey of unseal keys for Vault
path "sys/rekey/*" {
  capabilities = ["read", "list", "update", "delete"]
}
# Allows rotation of master key
path "sys/rotate" {
  capabilities = ["update", "sudo"]
}
# Allows Vault seal
path "sys/seal" {
  capabilities = ["sudo"]
}
```





END OF SECTION

 Vault

CERTIFIED
OPERATIONS
PROFESSIONAL

