Switzerland has bunkers, we have Vault

Christophe Tafani-Dereeper
09.11.2018

BLACK ALPS
CYBER SECURITY CONFERENCE

~$ whoami

Christophe Tafani-Dereeper
(christophetd)

Hacknowledge

➢ Security engineer @ Hacknowledge

➢ Threat hunting, SOC analyst, Infra, Dev(Sec)Ops

Goal of the talk

- Present the concepts and features of Hashicorp Vault

- Demonstrate how Vault can be used in the real-world scenarios

Challenges of secret management

- What is a secret?
- Secrets sprawled everywhere
- Hard to know where secrets are, who has access to them
- Hard to log accesses to secrets

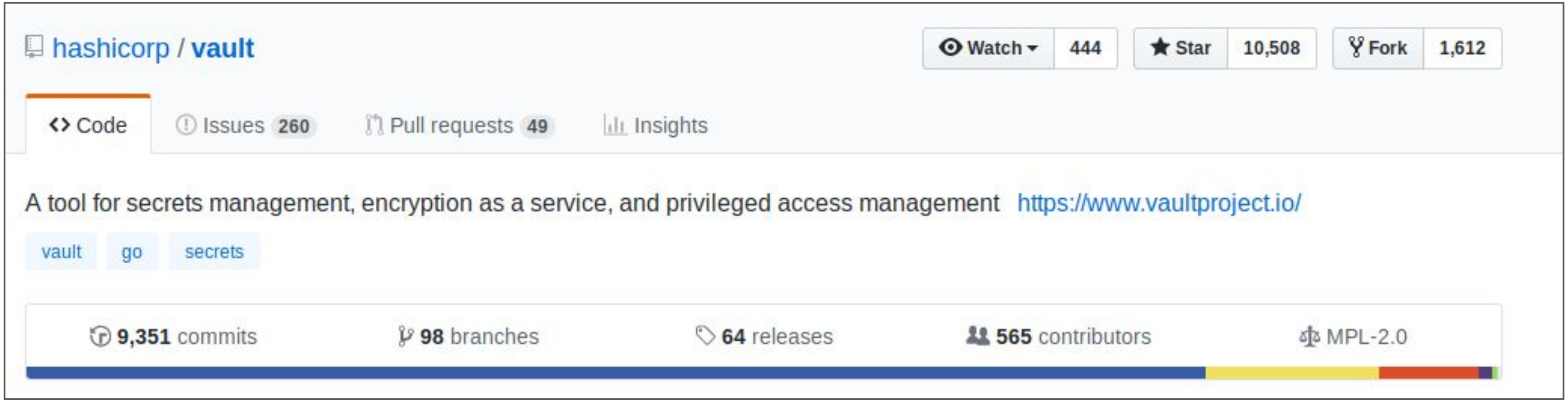
- What secrets were accessed?
- When?
- By whom?
- How to revoke them?

The Vault way

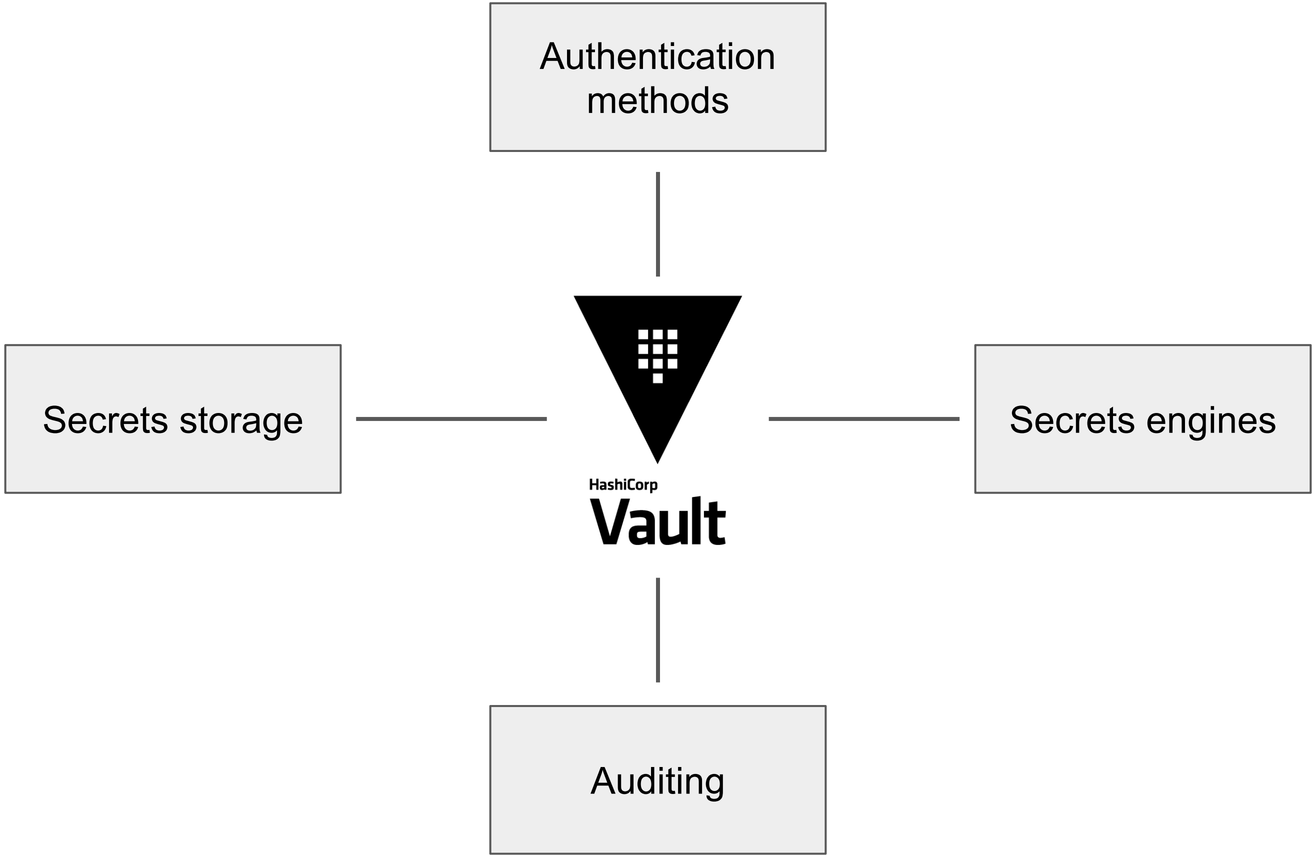
- Secrets are **centralized** in Vault
- Secrets are **short-lived** and **revokable**
- **Role-based ACLs** for granular access control
- Audit trail for strong **accountability** and non-repudiation

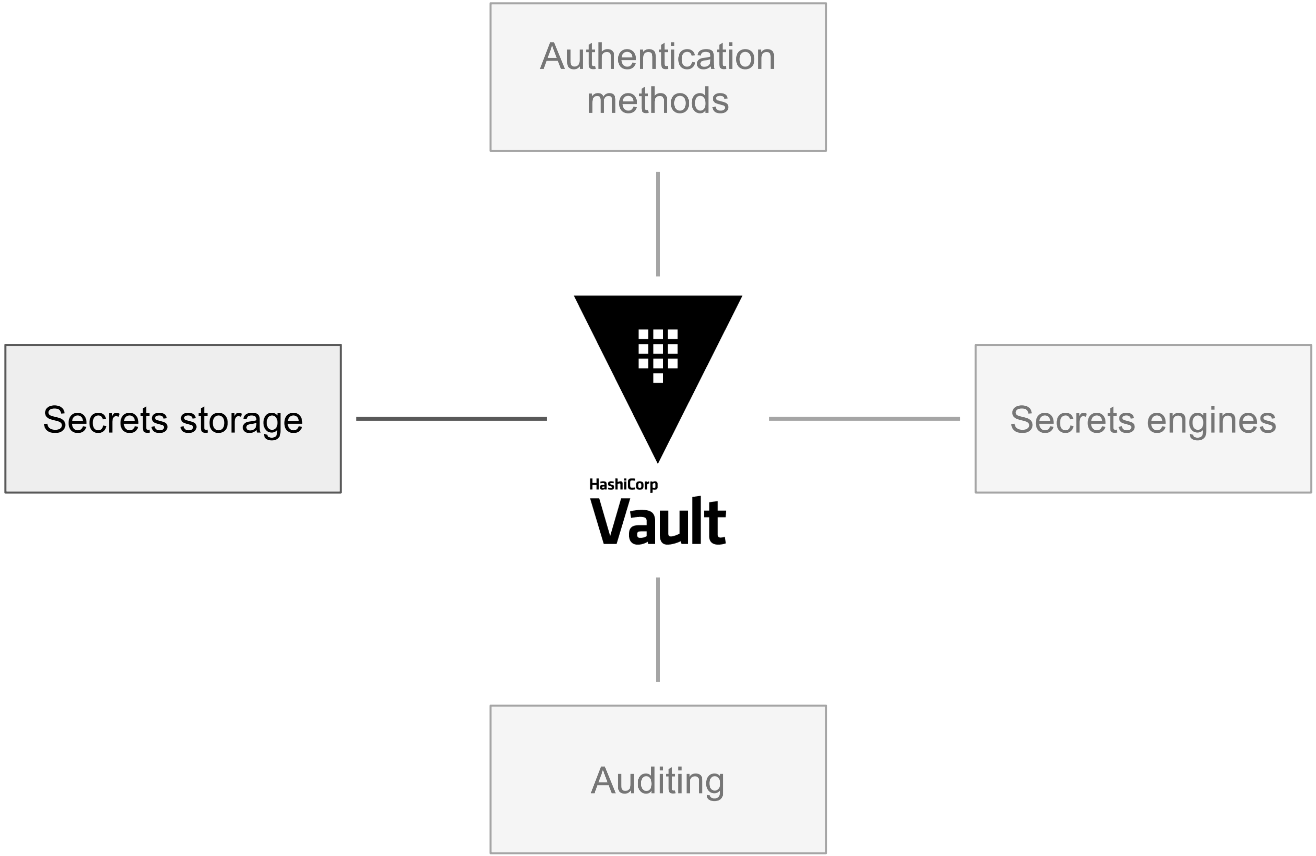
Vault 101

Hashicorp Vault



- First version in April 2015, 1.0 released on October 23rd 2018
- Free, with paid advanced features (not discussed in this talk)
- REST API, CLI

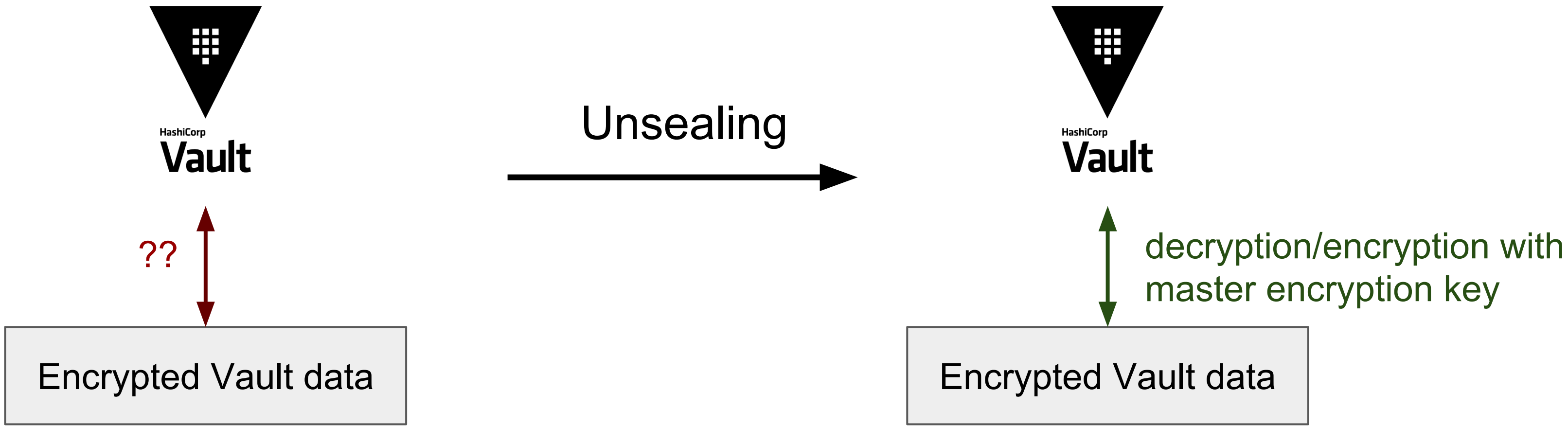

Authentication methods
Secrets storage
HashiCorp
Vault
Secrets engines
Auditing


Authentication methods
Secrets storage
HashiCorp
Vault
Secrets engines
Auditing

Secrets storage

- Secrets are stored encrypted in a storage backend of your choice
  - Filesystem, MySQL, S3, etcd, Consul…
- Storage backends are untrusted
  - Compromising the storage doesn’t allow to compromise the secrets stored in Vault
  - Authenticated encryption (AES GCM)
- How does Vault know how to decrypt its storage?
  ⇒ *Unsealing process*

Unsealing



HashiCorp
Vault
??
Encrypted Vault data
Unsealing
HashiCorp
Vault
decryption/encryption with
master encryption key
Encrypted Vault data

Master key splitting

- The master encryption key is split using Shamir's Secret Sharing algorithm
- The different parts are distributed to several trusted individuals
- The master key can be reconstructed with a certain number of key shares
    - num_shares = 3, threshold = 2
      ⇒ Any combination of 2 administrators can unseal Vault

```
$ vault operator init -key-shares=3 -key-threshold=2

Unseal Key 1: uCLmRwheyiBjI38so2ayYtearJyENppycC6XU//oRcHp
Unseal Key 2: 7VrbOoxN6y2X/ieTKhAz4BILTnenFMOYj2IzvVISd4ga
Unseal Key 3: ZkNnWwYnj20VGF+Ib9brR7oeHY+3dfkWdtaw2HgGwAv5
```

```
$ vault operator init \
    -key-shares=3 \
    -key-threshold=2 \
    -pgp-keys=keybase:christophetd,keybase:milkmix,keybase:lbarman

Unseal Key 1: (encrypted unseal key 1)
Unseal Key 2: (encrypted unseal key 2)
Unseal Key 3: (encrypted unseal key 3)
```

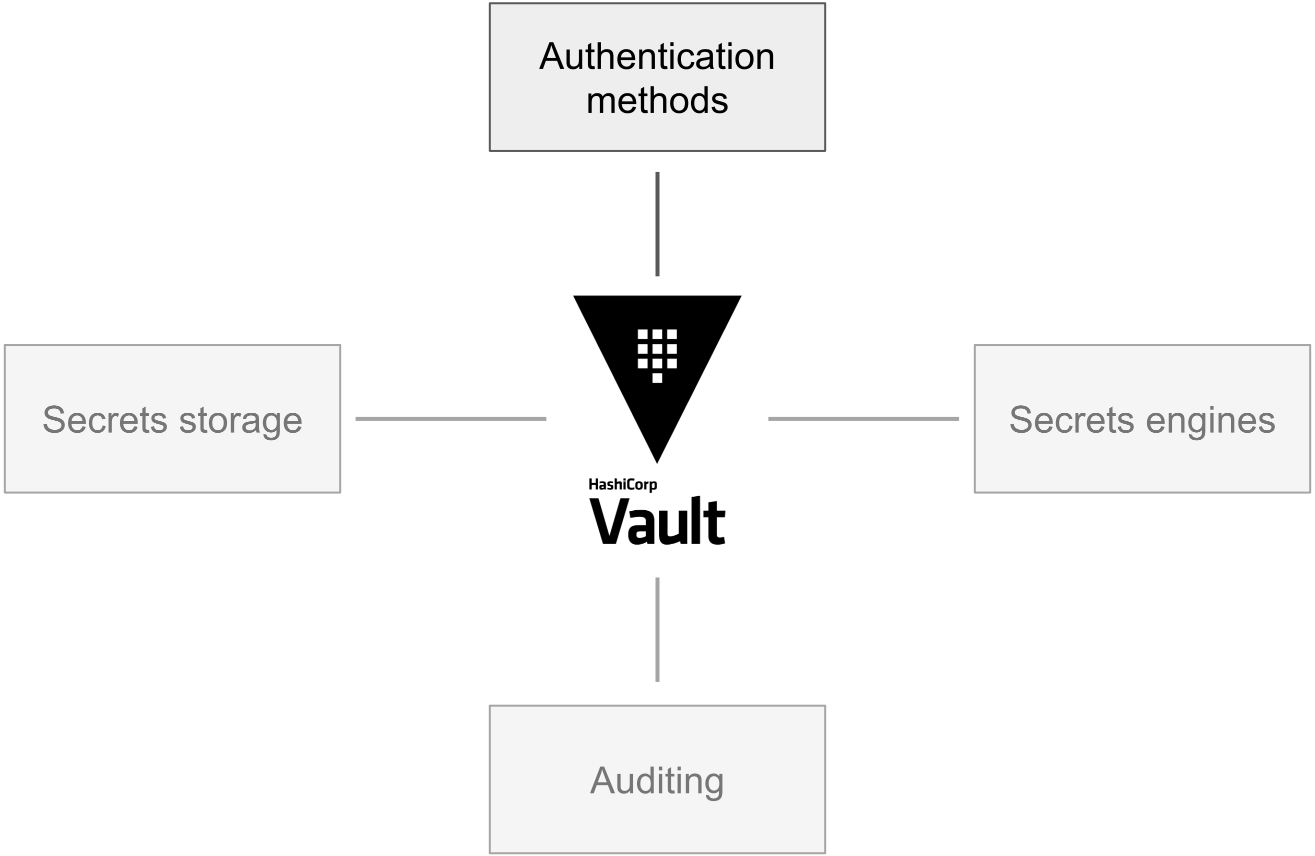
Unsealing process

Admin 1

```
$ vault operator unseal unsealing_key_1

Key                 Value
---                 -----
Seal Type           shamir
Sealed              true
Total Shares        3
Threshold           2
Unseal Progress     1/2
```

Admin 2

```
$ vault operator unseal unsealing_key_2


Key                 Value
---                 -----
Seal Type           shamir
Sealed              false
```

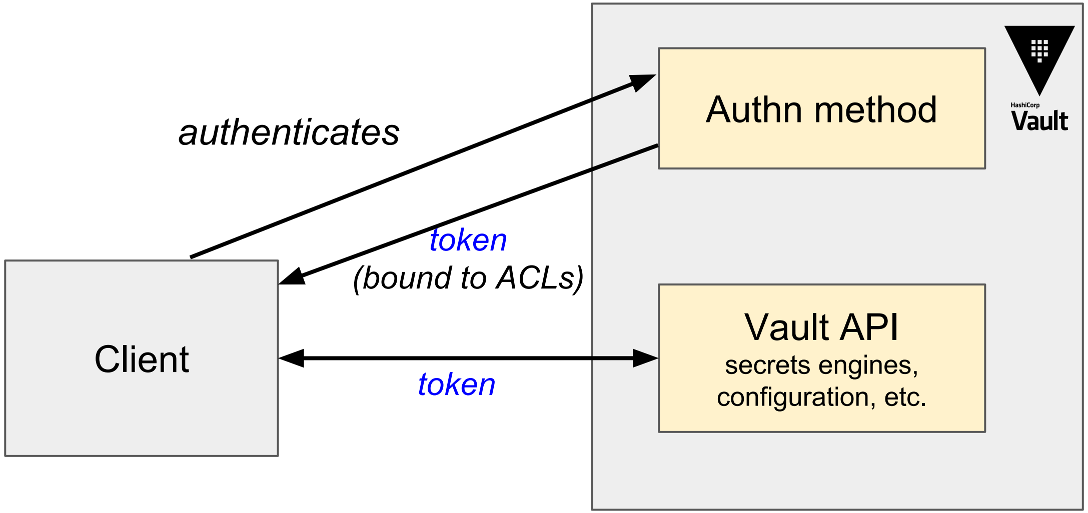

Authentication methods
Secrets storage
HashiCorp
Vault
Secrets engines
Auditing

Authentication & authorization

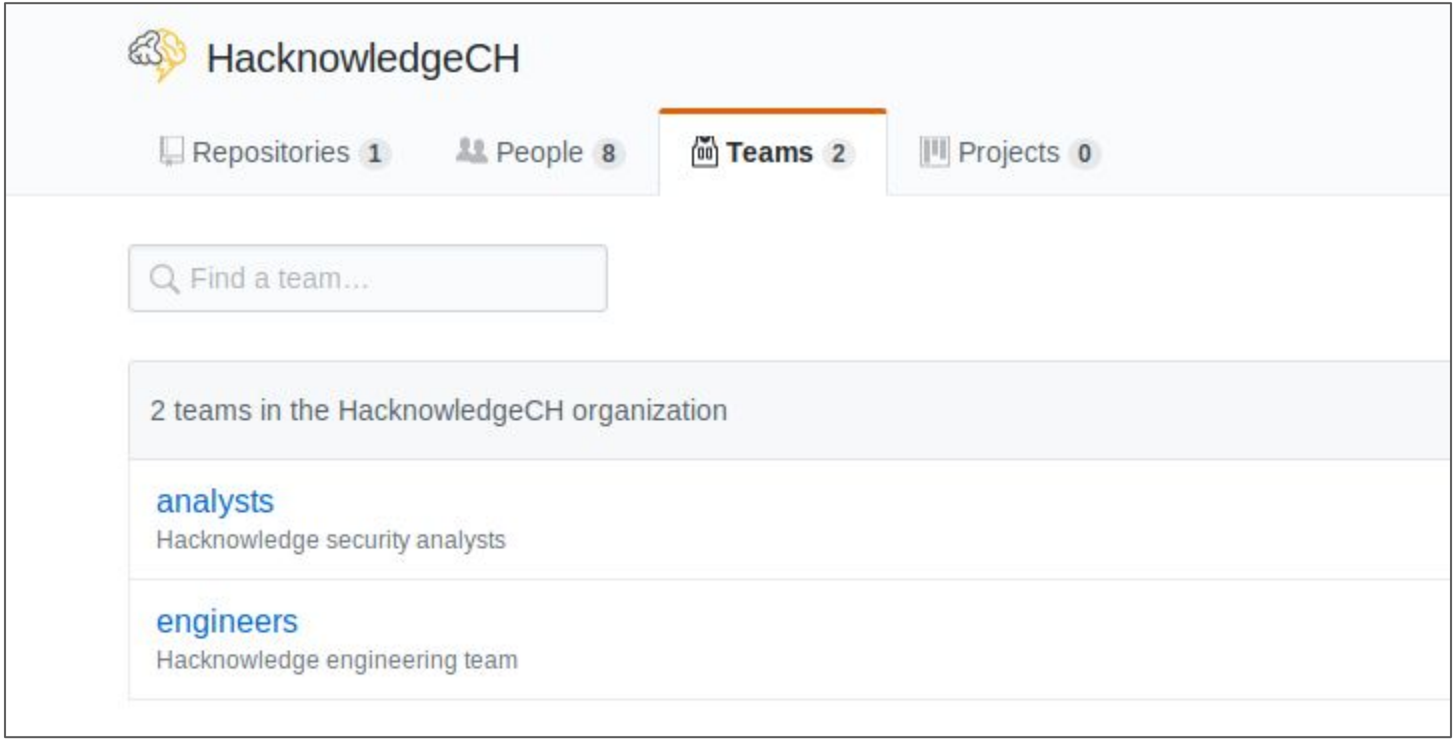
- Clients authenticate to Vault using an authentication method
  - For humans: LDAP, RADIUS, Github, username/password
  - For applications: AppRole, Kubernetes RBAC, AWS instance role

- Authorization:
  - Operators define ACLs on secret paths
  - Authentication engine configured to map authentication-method-specific user groups to Vault ACLs
    e.g. Users of the “engineering” team on Github map to the ACL “engineer” in Vault

Authentication & authorization



Example: Github authentication



Example: Github authentication

Initial setup

1. Create an ACL ("policy") for the engineering team
2. Enable the Github authentication method
3. Map Github users from the *engineering* team to the engineering team ACL

Usage

1. User authenticates to Vault using a Github access token
2. Vault returns a token (bound to the ACL for the engineering team)
3. User can use this token to interact with Vault

Example: Github authentication

Initial setup (Operator): Create an ACL ("policy") for the engineering team:

```
$ vault policy write engineers-policy - <<POLICY

path "static/engineering/*" {
  capabilities = ["create", "read", "update", "list"]
}

POLICY
```

Example: Github authentication

Initial setup (Operator): Set up the Github authentication method

```
$ vault auth enable github

$ vault write auth/github/config organization=Hacknowledge

$ vault write auth/github/map/teams/engineers value=engineers-policy
```

Example: Github authentication

Usage (normal user):

```
$ vault login -method=github

GitHub Personal Access Token: ***************

Success! You are now authenticated.

Key                    Value
---                    -----
token                  24GKildxW2aOy0cBFpNOEmsY
[...]
token_policies         [default engineers-policy]
```

Example: Github authentication

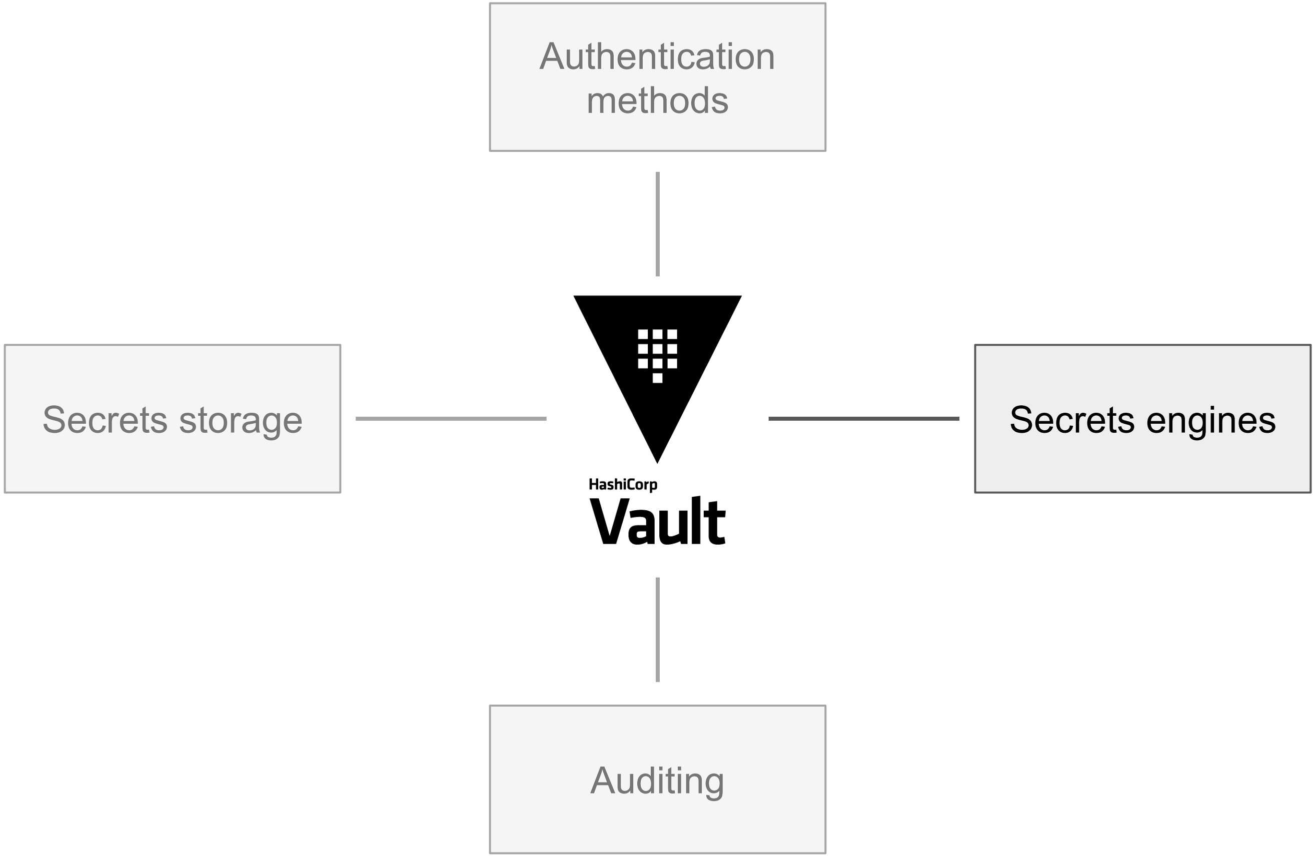
Usage (normal user):

```
$ vault read static/engineering/secret
=== Data ===
Key  Value
---  -----
foo  bar
```

```
$ vault read static/ops/secret
Error reading static/ops/secret: Error making API request.
permission denied
```

Authentication and authorization wrap-up

- **Policies** define the permissions each client has
- **Authentication methods** allow to map external identities to a set of policies


Authentication methods
Secrets storage
HashiCorp Vault
Secrets engines
Auditing

Secrets engines

- Secrets engines are at the core of Vault
    - allow to store, generate, and manage all kind of secrets

- Lots of different secrets engines
    - Key-value (example on previous slide)
    - MySQL, PostgreSQL
    - AWS, Azure
    - SSH
    - PKI

Secrets engines

- *Everything is a path*: secrets engines can be mounted (enabled) and unmounted (disabled) in Vault

```
$ vault secrets enable -path=static kv
Success! Enabled the kv secrets engine at: static/
```

- Each secret has a path within the engine they belong to
  (e.g. *static/banking/credit-card*)

Static secrets engine: Key Value

- Most basic secret engine - can store arbitrary key-value pairs

```
$ vault write static/banking/credit-card number=123456 exp=01/2021
Success! Data written to: static/banking/credit-card
```

```
$ vault read static/banking/credit-card
Key               Value
---               -----
exp               01/2021
number            123456
```

Dynamic secrets engines

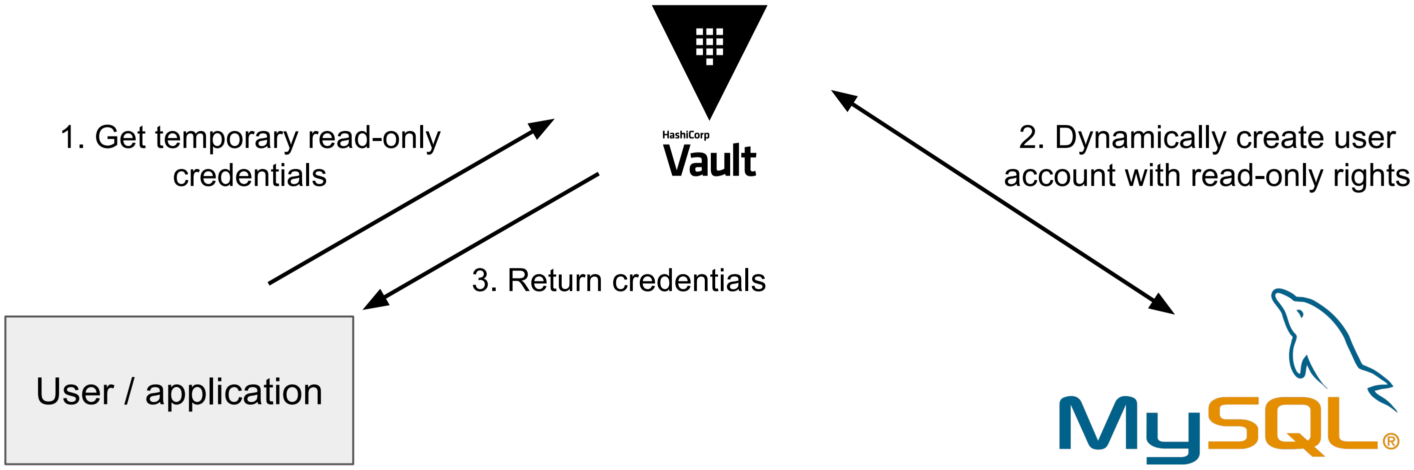
- A dynamic secret engine generates secrets on-the-fly
  - MySQL/PostgreSQL: create user account
  - AWS: generate IAM credentials
  - PKI: sign certificate

- Dynamic secrets are supposed to be short-lived and revokable

Dynamic secret engine example: MySQL

- Vault holds root MySQL credentials
- Vault dynamically generates MySQL credentials with specific rights
- Credentials are limited in time and can be revoked

Dynamic secret engine example: MySQL



Dynamic secret engine example: MySQL

Setup (operator):

```
$ vault secrets enable -path=db database

$ vault write db/config/mysql-prod \
   plugin_name="mysql-database-plugin" \
   connection_url="{{username}}:{{password}}@tcp(127.0.0.1:3306)/" \
   username="root" \
   password="my-secret-pw" \
   allowed_roles="mysql-prod-readonly"

$ vault write db/roles/mysql-prod-readonly \
    db_name=mysql-prod \
    creation_statements="CREATE USER '{{name}}'@'%' IDENTIFIED BY '{{password}}';\
      GRANT SELECT ON *.* TO '{{name}}'@'%';" \
    default_ttl="10m"
```

Dynamic secret engine example: MySQL

Usage (user or application):

```
$ vault read db/creds/mysql-prod-readonly

Key                  Value
---                  -----
lease_id             db/creds/mysql-prod-readonly/4bFeHHV4fzJSs6T9xLQrFhdH
lease_duration       10m
lease_renewable      true
password             A1a-HpsqK2m547gSP5I0
username             v-root-mydb-reado-6akkZS1xkOsm2P
```

Access control with an ACL!

Dynamic secrets: leases

- Most dynamic secrets have a **lease**

```
lease = { id, time_to_live, is_renewable }
```

- In our previous MySQL example, we had:

```
$ vault read db/creds/mydb-readonly

Key                 Value
---                 -----
lease_id            db/creds/mysql-prod-readonly/4bFeHHV4fzJSs6T9xLQrFhdH
lease_duration      10m
lease_renewable     true
password            A1a-HpsqK2m547gSP5I0
username            v-root-mydb-reado-6akkZS1xkOsm2P
```

Dynamic secrets: leases

- Leases can be renewed

```
$ vault lease renew db/creds/mysql-prod-readonly/4bFeHHV4fzJSs6T9xLQrFhdH

Key                 Value
---                 -----
lease_id            db/creds/mysql-prod-readonly/4bFeHHV4fzJSs6T9xLQrFhdH
lease_duration      10m
lease_renewable     true
```

(makes the previously obtained credentials valid for 10 more minutes)

Dynamic secrets: leases

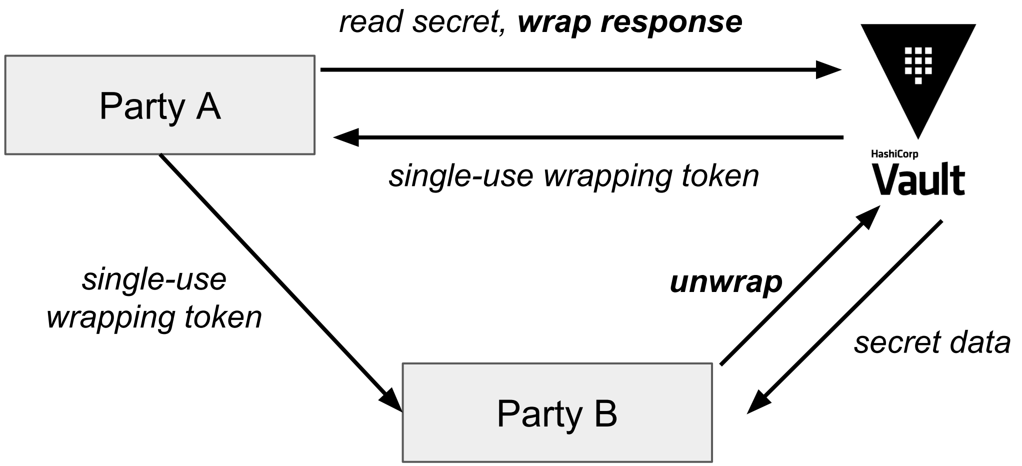
- Leases can be revoked by operators, individually or by prefix

```
$ vault lease revoke db/creds/mysql-prod-readonly/4bFeHHV4fzJSs6T9xLQrFh

Success! Revoked lease: db/creds/mysql-prod-readonly/4bFeHHV4fzJSs6T9xLQrFhdH
```

```
$ vault lease revoke -prefix db/creds/mysql-prod-readonly

Success! Revoked any leases with prefix: db/creds/mysql-prod-readonly
```

Additional concept: Response Wrapping

- Building block that can be used in more complex workflows
- When party A needs to communicate a secret to party B over an insecure channel



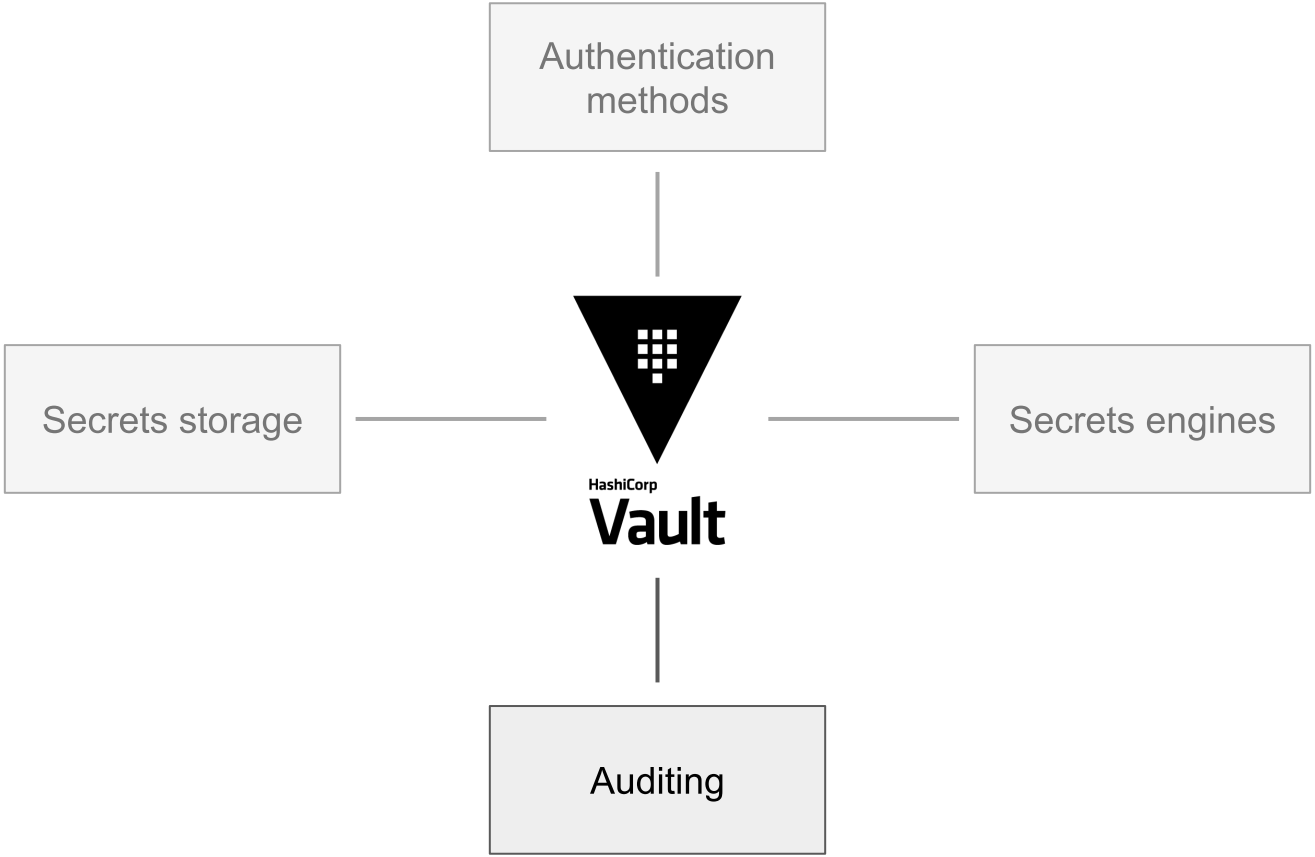
Additional concept: Response Wrapping

- **Coverage**: the transmitted information is only a *reference* to the actual secret

- **Malfeasance detection**: party B detects if the communication has been intercepted
  - Vault will tell it that the wrapping token is not valid
  - Party B can then raise an alert

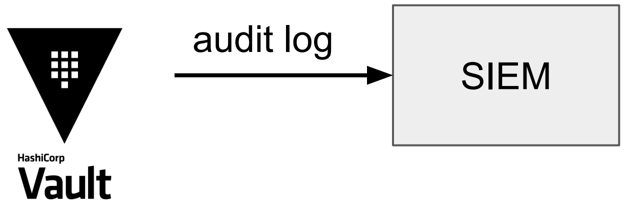
- **Limits exposure lifetime**
  - wrapping token typically expires very quickly
  - its lifetime is independant than the TTL of the secret it wraps

- (Does *not* provide confidentiality)


Authentication methods
Secrets storage
HashiCorp
Vault
Secrets engines
Auditing

Audit log

- Vault has an audit log for every request / response
- Can be shipped to syslog, or local file



Audit log


```
{
  "time": "2018-02-31T13:37:37.123Z",
  "type": "request",
  "auth": {
     "display_name": "github-christophetd",
     "policies": [
       "default",
       "engineers-policy"
     ],
     "metadata": {
     "org": "Hacknowledge",
     "username": "christophetd"
     },
  },
  "request": {
     "id": "97166a54-6b7b-f577-749a-96f191c9a10c",
     "operation": "read",
     "path": "secret/supersecret",
     "remote_address": "10.0.1.47",
  },
  "error": "1 error occurred:\n\n* permission denied"
}
```

Audit log use-cases

- Anomaly detection
  - Access denied errors
  - Failed authentications
- Logs correlation
- “Honey secrets”
  - Give an application access to *secret/honey*
  - Allow the application to read the policy attached to its token (*sys/policy/app-policy*)
  - Raise alert if *secret/honey* is accessed - can indicate an attacker enumerating its privileges

Hands-on with Vault

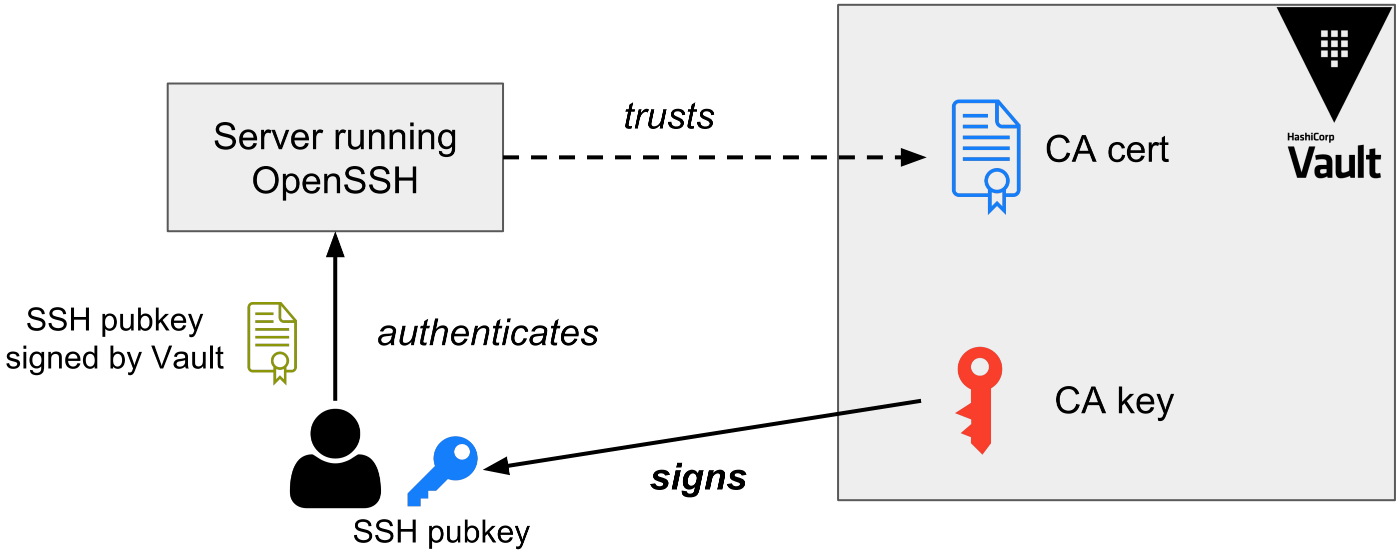
Scenario #1: SSH access management

- Context:
  - You have a fleet of Linux servers
  - You want to provide SSH accesses to your team in a scalable way
- Approaches
  - 1 Linux user per employee per machine
  - 1 user on all machines, employees' public keys in the *authorized_keys* on each machine
  - PAM
  - Vault's SSH secret backend

Scenario #1: SSH access management

- Vault holds a SSH CA key, signs employees' public keys
- Linux servers trust Vault's CA certificate
- Built-in OpenSSH feature!
  - 0 additional software to install
  - 0 communication needed between Linux servers and Vault

Scenario #1: SSH access management



Scenario #1: Initial setup phase

- Enable Vault's SSH secret backend

```
$ vault secrets enable ssh
Success! Enabled the ssh secrets engine at: ssh/
```

- Generate a SSH CA certificate and key (only stored in Vault)

```
$ vault write ssh/config/ca generate_signing_key=true

Key           Value
---           -----
public_key    ssh-rsa AAAAB3NzaC…..
```

Scenario #1: Initial setup phase

- Deploy Vault's SSH CA certificate as a trusted SSH CA on Linux machines

/etc/ssh/sshd_config

```
TrustedUserCAKeys /etc/ssh/vault-ssh-ca.crt
```

Scenario #1: Initial setup phase

- Create a role in the SSH secrets engine, specifying…
  - A TTL: for how much time should Vault sign users' public keys?
  - A remote user to allow connection as
  - (optionally) A CIDR list from which access should be allowed
  - (optionally) SSH features to allow (PTY, port forwarding, etc)

```
$ vault write -f ssh/roles/developer - <<EOF
{
  "ttl": "10m",
  "allowed_users": "developer,tomcat",
  "default_user": "developer",
  "default_critical_options": { "source-address": "10.0.0.0/24" },
  "default_extensions": { "permit-pty": "", "permit-port-forwarding": "" },
  "allow_user_certificates": true,
  "key_type": "ca"
}
EOF
```

Scenario #1: Usage

- Ask Vault to sign our SSH public key

```
$ vault write ssh/sign/developer \
   public_key=@.ssh/id_rsa.pub \
   valid_principals=developer

Key             Value
---             -----
serial_number   458e609f5eed0a8a
signed_key      ssh-rsa-cert-v01@openssh.com AAAA...
```

- Connect to a Linux server trusting Vault’s SSH CA

```
$ ssh -i signed_key.pub developer@10.0.0.31

Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.13.0-36-generic x86_64)
developer@server:~$
```

Scenario #1: Usage (wrapper)

- *vault ssh* wrapper can do both in a single command

```
$ vault ssh -mode=ca -role=developer developer@10.0.0.31
```

Scenario #1: TTL

- Once the TTL is over, the signed key is not valid anymore

```
$ ssh -i .ssh/id_rsa -i signed_key.pub developer@10.0.0.31

developer@10.0.0.31: Permission denied (publickey).
```

- What TTL to use?
  - Tradeoff between performance / availability and easy revokation

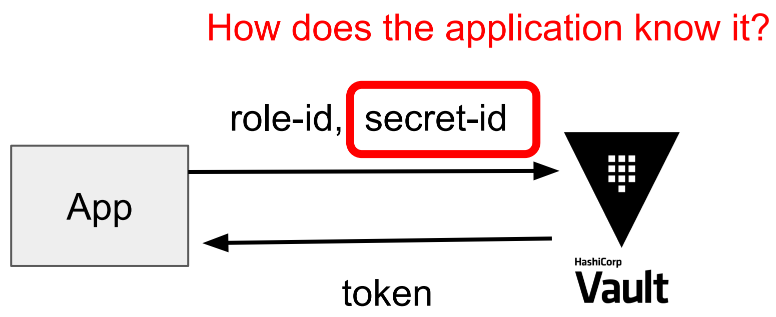
Scenario #2: Authenticating applications

- Authentication easy for humans, harder for applications

- Our requirements:
  - Applications should be deployable automatically (e.g. via a CI/CD pipeline)
  - Each application should have a dedicated policy only allowing it to retrieve its own secrets

- Most of the time, AppRole authentication method is the way to go
  - but it only provides a building block

Scenario #2: Authenticating applications with AppRole



- Tentative 1: Hardcode the secret-id on the VM/container where the app runs
  ⇒ But how do you initially get the secret-id?

- Tentative 2: Have the CI/CD inject the secret-id in the VM/container at deployment time
  ⇒ But how can the CI/CD authenticate to Vault to have access to the secret-id?

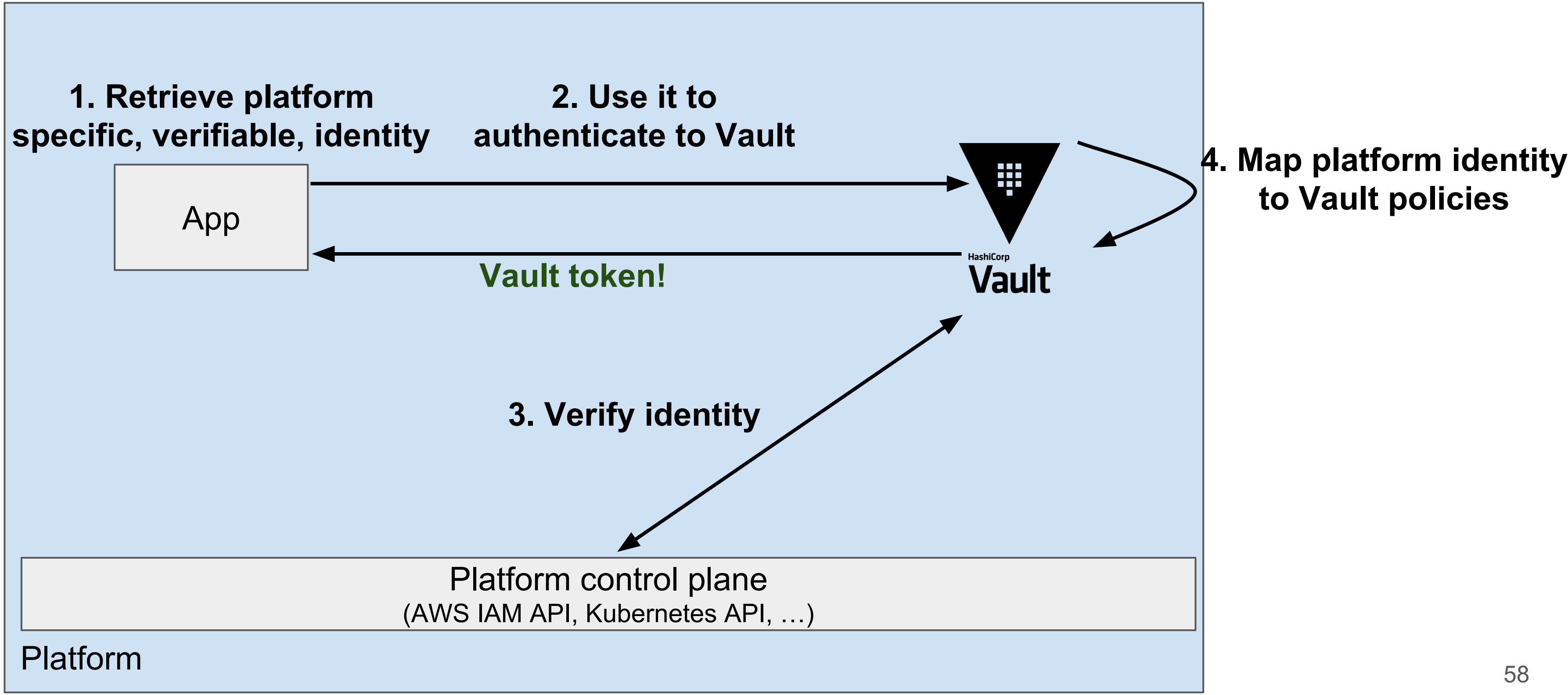
Option 1: Platform integration

- The platform assigns a cryptographic and verifiable identity to each application instance
  - AWS: IAM EC2/ECS instance role
  - Kubernetes: Pod service account

- At runtime, the platform gives an easy way to the application to prove its identity
  - AWS: Metadata service running on `169.254.169.254` (only accessible from the instance)
  - Kubernetes: Injected in a volume `/var/run/secrets/`

- Vault has several authentication engines to allow application authentication with their platform-specific identity
  - AWS, Azure, AliCloud, Google Cloud, Kubernetes secrets engines

Option 1: Platform integration



Option 2: No platform integration

- e.g. your applications run in VMs on an on-prem ESXi cluster

- How do you pass the authentication secret (secret-id) to your applications?

- Challenging problem - no silver bullet
  - highly dependent on the environment and technologies in use
  - hard to have a solution as secure as with platform integration

Option 2: Trusted orchestrator

- **Trusted orchestrator**: We extends our trust to an additional component
  e.g. Jenkins, Gitlab CI
- Orchestrator is authenticated to Vault
- Orchestrator passes the AppRole secret-id to application it deploys

Option 2: Trusted orchestrator

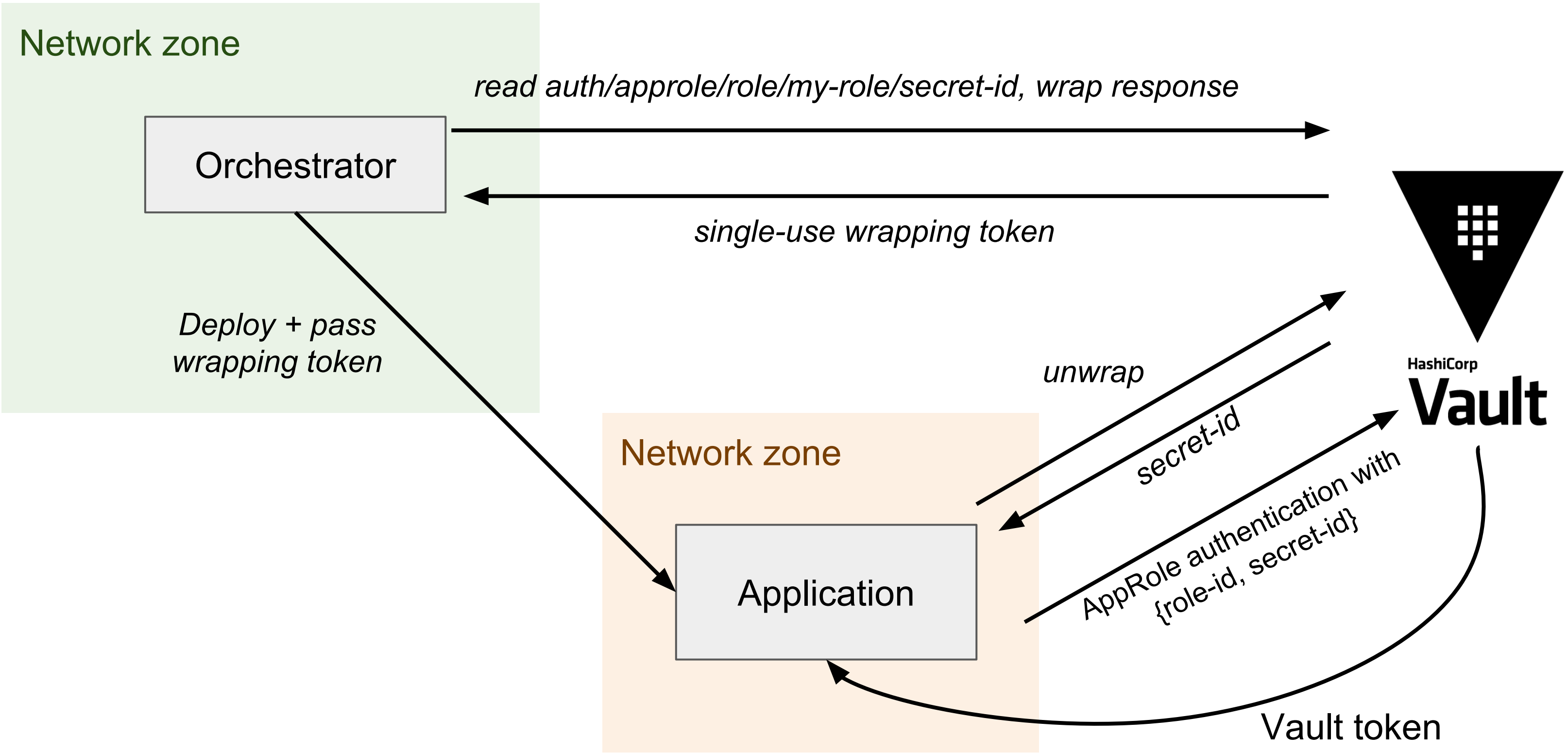
- Orchestrator:
  - Can only retrieve the application's AppRole secret-id (cannot read application secrets)

```
path "auth/approle/role/my-app/secret-id" {
    capabilities = ["create", "update"]
}
```

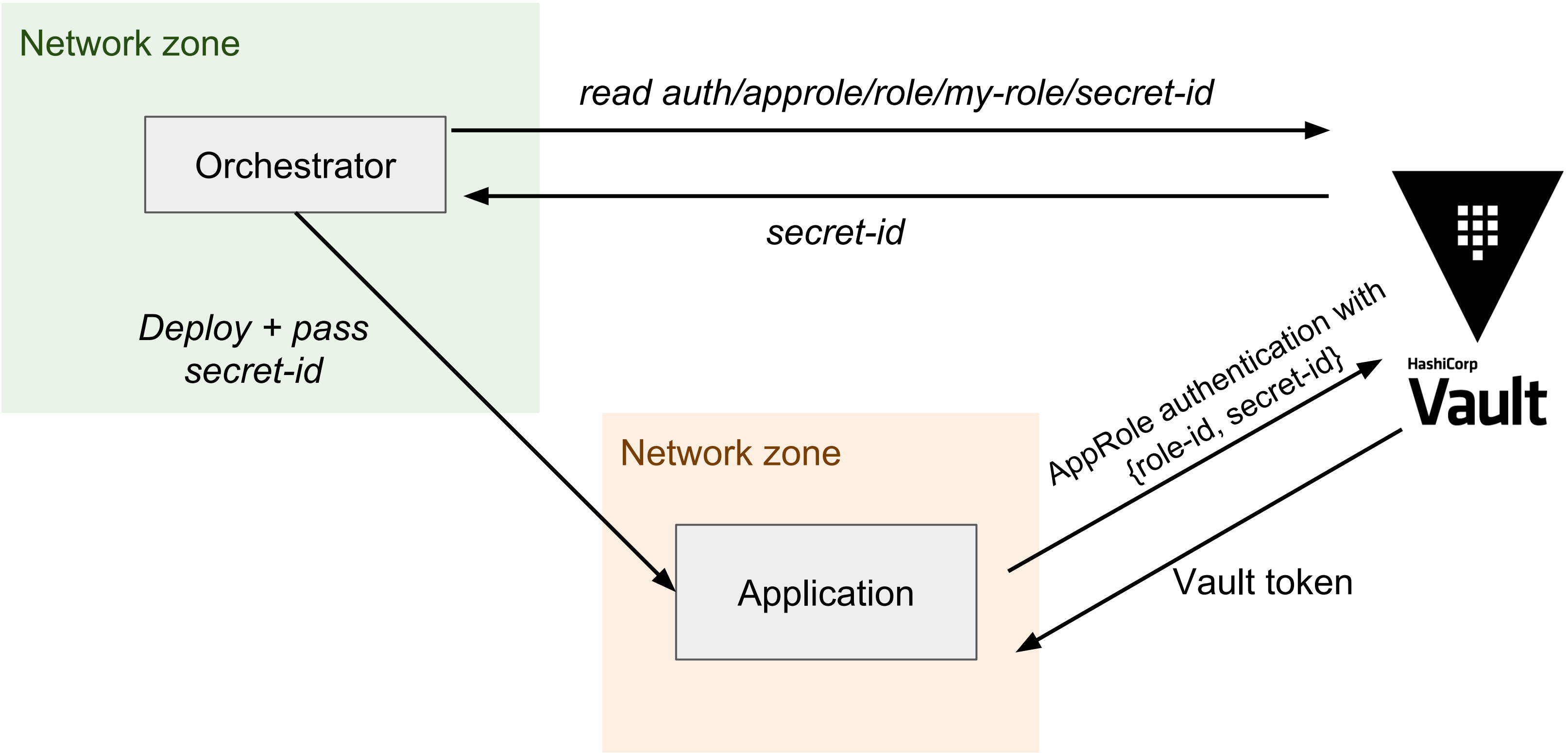
  - Is in a different network than the applications it deploys

- Applications:
  - authenticate using a dedicated AppRole
  - AppRole is configured to only allow authentications from the apps network
  - can only read their own secrets

Trusted orchestrator scenario (with response wrapping)



Trusted orchestrator scenario



Trusted orchestrator scenario: result #1

Good:

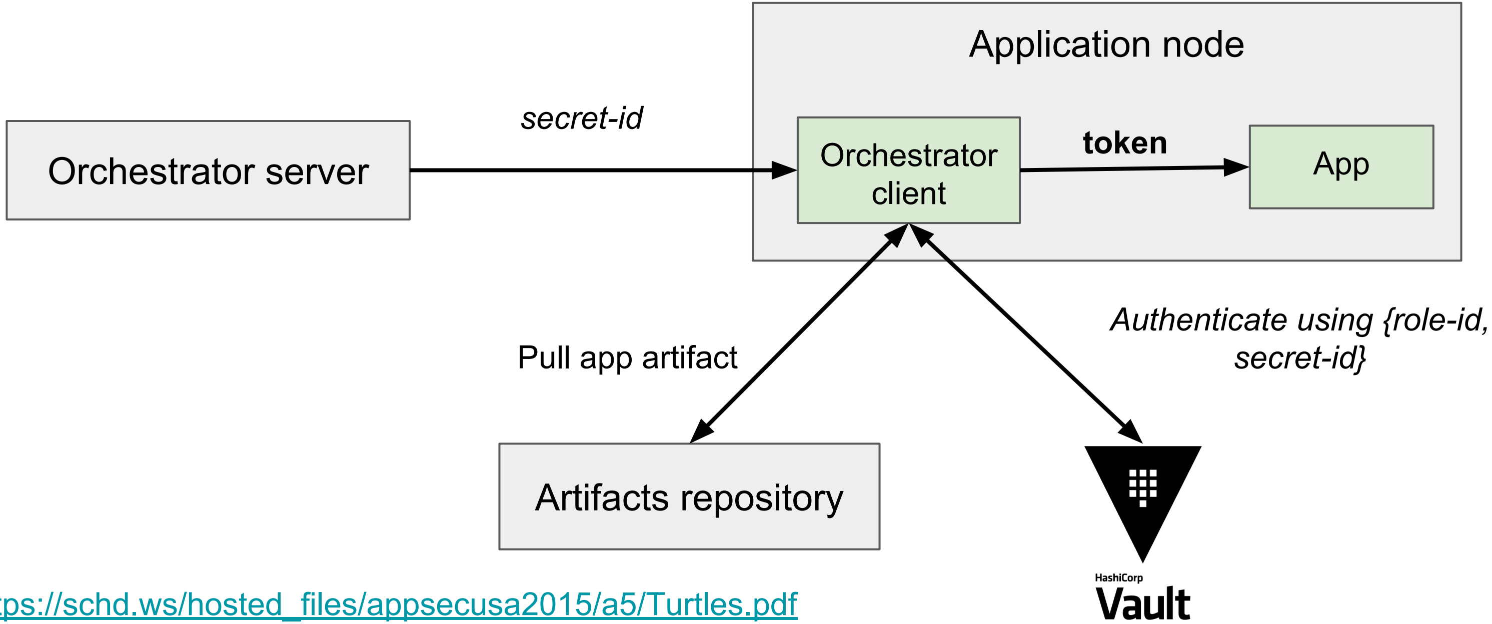
- Applications can only access secrets as defined by their AppRole policy
- Orchestrator cannot access applications' secrets
    - It cannot authenticate using the application's secret-id (CIDR restrictions)

Bad:

- A compromised orchestrator can be used to deploy a backdoored application that leaks secrets

Can we do better?

- Problem: Orchestrator has total control over the nodes where the apps run
- Consequence: Compromised orchestrator ⇒ compromised apps secrets
- Potential solution:



https://schd.ws/hosted_files/appsecusa2015/a5/Turtles.pdf

Trusted orchestrator scenario: result #2

- Orchestrator cannot deploy backdoored applications anymore

- It must still authenticate to Vault by some way (e.g. hardcoded token)
  - … but compromising the orchestrator becomes much less interesting!

- Potential improvement: response wrapping

Wrapping up - Other Vault capabilities

- Transit secret backend: Encryption As a Service
- PKI secrets backend
- High-Availability mode
- Web UI

Wrapping up - Vault limitations

- Unsealing process hard to automate
- Can easily become a single point of failure
- Not all secrets can be dynamic
- Added complexity

Wrapping up - Vault alternatives

- Provider-dependent solutions:
    - AWS KMS
    - Google Cloud KMS
    - Azure Key Vault
- Hardware Security Modules
- Software Solutions
    - Square's KeyWhiz
    - Pinterest's Knox

Readings and resources

- Hashicorp Learning center
  https://learn.hashicorp.com/vault/

- "*Secrets at Scale: Automated Bootstrapping of Secrets & Identity in the Cloud*" (Netflix)
  https://www.youtube.com/watch?v=15H5uCj1hlE

- "*The Secure Introduction Problem: Getting Secrets Into Containers*"
  https://slideshare.net/DynamicInfraDays/containerdays-nyc-2016-the-secure-introduction-problem-getting-secrets-into-containers-jeff-mitchell

- "*Secret Security Turtles*"
  https://blog.alanthatcher.io/vault-security-turtles

Thank you!

Twitter @christophetd
Email christophe@tafani-dereeper.me
Web https://christophetd.fr