HashiTalks

Why you should use Vault as your Consul Certificate Authority

HashiTalks 2023

Thomas Kula

Sr. Staff Solutions Engineer at HashiCorp
[he/him]

thomashashi

01 Why you should use Vault as your Consul Certificate Authority

CA Usage in Consul

Certificate Authority usage in Consul

There are two primary uses of a Certificate Authority in Consul

Certificate Authority usage in Consul

There are two primary uses of a Certificate Authority in Consul

- Consul client auto-encryption/auto-config

Certificate Authority usage in Consul

There are two primary uses of a Certificate Authority in Consul

- Consul client auto-encryption/auto-config
- Consul Service Mesh (Connect)

Certificate Authority usage in Consul

There are two primary uses of a Certificate Authority in Consul

- Consul client auto-encryption/auto-config
- Consul Service Mesh (Connect)

Certificate Authority usage in Consul

Consul Service Mesh CA Usage

The core of Consul Service Mesh is *mutual TLS (mTLS)*

https://developer.hashicorp.com/consul/docs/connect/connect-internals

Certificate Authority usage in Consul

Consul Service Mesh CA Usage

The core of Consul Service Mesh is *mutual TLS (mTLS)*

- Every instance of every service on the service mesh has a TLS certificate

https://developer.hashicorp.com/consul/docs/connect/connect-internals

Certificate Authority usage in Consul

Consul Service Mesh CA Usage

The core of Consul Service Mesh is *mutual TLS (mTLS)*

- Every instance of every service on the service mesh has a TLS certificate
- That is how, on the wire, services identify themselves to other services

https://developer.hashicorp.com/consul/docs/connect/connect-internals

Certificate Authority usage in Consul

Consul Service Mesh CA Usage

The core of Consul Service Mesh is *mutual TLS (mTLS)*

- Every instance of every service on the service mesh has a TLS certificate
- That is how, on the wire, services identify themselves to other services
- That service identity is used when controlling service access—*Intentions*

https://developer.hashicorp.com/consul/docs/connect/connect-internals

Certificate Authority usage in Consul

Consul Service Mesh CA Usage

Who are you?

Certificate Authority usage in Consul

Consul Service Mesh CA Usage

Who are you?

Are you allowed in?

02 Why you should use Vault as your Consul Certificate Authority

What we need to protect

What we need to protect

What we don't care about

What we need to protect

~~CERTIFICATES~~

What we need to protect

We don't care about certificates

- This may come as a mild surprise, but we actually don't care about certificates

What we need to protect

We don't care about certificates

- This may come as a mild surprise, but we actually don't care about certificates
- A certificate is only an identity *assertion:* "The presenter of this certificate is www.hashicorp.com"

What we need to protect

We don't care about certificates

- This may come as a mild surprise, but we actually don't care about certificates
- A certificate is only an identity *assertion:* "The presenter of this certificate is www.hashicorp.com"
- And instructions on how to verify this: "And here's how the presenter will prove this"

What we need to protect

We don't care about certificates

- We throw around certificates all the time—*they are public documents*
- Here's the certificate for `www.hashicorp.com`

```
openssl s_client -showcerts -servername www.hashicorp.com -connect
www.hashicorp.com:443 2>/dev/null < /dev/null | openssl x509 -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            03:20:0f:1c:ad:7f:5a:7e:06:81:09:93:1b:7d:3e:e7:9f:f9
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Let's Encrypt, CN=R3
        Validity
            Not Before: Jan  3 10:51:00 2023 GMT
            Not After : Apr  3 10:50:59 2023 GMT
        Subject: CN=www.hashicorp.com

[... truncated for brevity ...]
```

What we need to protect

But we do care about...

- But if we look further in that certificate

```
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:e3:5a:83:7e:04:ee:f7:4f:8b:c2:64:5f:ad:7e:
                    c9:94:39:27:c7:d9:62:f9:45:92:93:a2:44:05:57:
                    87:e9:0d:19:8d:da:aa:0f:5a:8b:a1:c6:59:4b:0c:
                    08:a7:dd:be:b1:fb:29:ca:eb:93:a5:3a:88:74:24:
                    53:6a:66:39:ae:f6:5c:05:26:8b:85:3a:f6:bf:87:
                    35:40:07:50:1c:15:46:47:6c:c4:a1:0f:d3:13:7f:
                    f7:14:25:26:7a:8e:80:39:65:dd:0e:b1:a9:2f:a9:
                    52:2b:7b:56:d4:88:73:81:0a:55:30:86:9f:1b:8a:
                    e2:93:f5:f4:10:c2:1c:da:c8:c9:7f:1c:e4:e6:0c:
                    36:ac:af:7f:6c:ac:7e:8e:2f:49:1c:ca:3f:d8:ab:
                    09:34:e6:1b:d3:f6:5a:f7:47:95:ff:bf:79:ad:16:
                    e3:c0:56:b4:26:be:25:35:3c:01:cb:38:f8:3e:88:
                    8c:37:9c:28:59:a9:68:15:55:22:ec:e9:9a:4f:da:
                    76:13:bd:ad:a2:61:55:56:5a:ff:1c:c6:fb:6d:61:
                    46:45:7c:e7:ea:fa:ae:ec:53:0b:72:8a:2b:39:a1:
                    f0:08:ac:a3:27:1a:da:5a:db:a7:fa:43:96:ae:70:
                    5b:a7:8a:3b:ff:df:0d:8f:4d:b0:e7:47:5b:31:d6:
                    26:41
                Exponent: 65537 (0x10001)
```

What we need to protect

What we care about

- A given Public Key has one and only one Private Key

What we need to protect

What we care about

- A given Public Key has one and only one Private Key
- The holder of a Private Key can perform a *signing* operation

What we need to protect

What we care about

- A given Public Key has one and only one Private Key
- The holder of a Private Key can perform a *signing* operation
- That signature can be *verified* with the corresponding Public Key

What we need to protect

What we care about

- A given Public Key has one and only one Private Key
- The holder of a Private Key can perform a *signing* operation
- That signature can be *verified* with the corresponding Public Key
- Allowing anyone with the Public Key to verify the signature was performed with the Private Key *without requiring any access to the Private Key*

What we need to protect

What we care about

- A given Public Key has one and only one Private Key
- The holder of a Private Key can perform a *signing* operation
- That signature can be *verified* with the corresponding Public Key
- Allowing anyone with the Public Key to verify the signature was performed with the Private Key *without requiring any access to the Private Key*
- TLS uses the certificate, and a signature using that Public/Private key pair of data exchanged during the handshake, to prove that identity assertion

What we need to protect

PRIVATE KEYS

What we need to protect

What we care about

- We care about Private Keys because that's what the presenter of that certificate uses to prove it is the legitimate holder of that certificate

What we need to protect

What we care about

- We care about Private Keys because that's what the presenter of that certificate uses to prove it is the legitimate holder of that certificate
- And not some random bystander

What we need to protect

What we care about

- But we only know about that public/private key pair because it was in the certificate presented

What we need to protect

What we care about

- But we only know about that public/private key pair because it was in the certificate presented
- How do we know *that* wasn't faked?

What we need to protect

What we care about

- But we only know about that public/private key pair because it was in the certificate presented
- How do we know *that* wasn't faked?
- *How can we trust the information in that certificate?*

What we need to protect

Trust Path

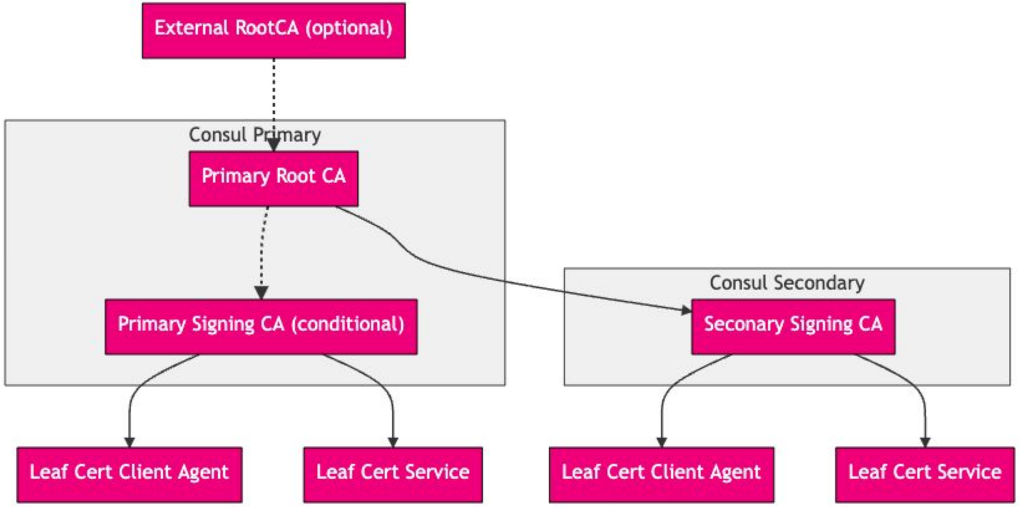
- Back in the certificate

```
[...]
Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=US, O=Let's Encrypt, CN=R3
[...]
X509v3 extensions:
    X509v3 Authority Key Identifier:
            keyid:14:2E:B3:17:B7:58:56:CB:AE:50:09:40:E6:1F:AF:9D:8B:14:C2:C6
[...]
Signature Algorithm: sha256WithRSAEncryption
     35:a6:d9:59:42:07:ce:64:e5:4c:53:15:54:04:ef:ce:e5:80:
     56:c4:81:0d:31:1a:55:b5:5c:96:92:56:a7:35:00:98:22:b0:
     28:39:d5:cc:09:7e:4a:92:6c:29:a8:80:51:38:64:72:7f:de:
     39:3d:08:4a:46:37:5a:40:fb:28:79:b7:cb:22:04:4d:36:4b:
     1c:a5:d3:8b:19:95:0b:7e:95:33:b5:21:4e:ee:41:9d:a6:93:
     6f:80:11:de:5d:db:f0:47:81:a1:bc:de:f2:60:b7:63:b1:3f:
     45:4b:f8:5e:b7:5b:09:c0:18:b8:34:17:ed:36:c5:bc:2a:f9:
     ad:db:11:5f:e8:10:6a:8a:ec:04:92:16:7c:7e:9f:49:44:f9:
     69:ed:cc:28:b8:be:8f:c7:60:5a:5f:76:14:04:53:05:b3:b9:
     e3:62:7d:52:5c:54:d6:ad:30:7e:39:6e:d6:96:71:35:88:1f:
     03:07:d5:ac:02:b1:e4:4d:53:38:35:84:fd:06:94:37:54:b7:
     c4:7b:25:35:4e:da:68:ff:34:d1:e4:5a:cc:0c:4b:67:8e:f4:
     fc:2a:8b:bc:32:26:96:e4:11:09:fa:23:70:7d:fb:fd:37:94:
     c7:33:8a:3d:c0:32:fa:2d:87:7d:3d:3f:bf:58:02:ee:95:ec:
     6d:fb:df:29
```

What we need to protect

Protecting Private Keys

- We want not only to be able to verify the certificate, but that *it came ultimately from somewhere we trust*

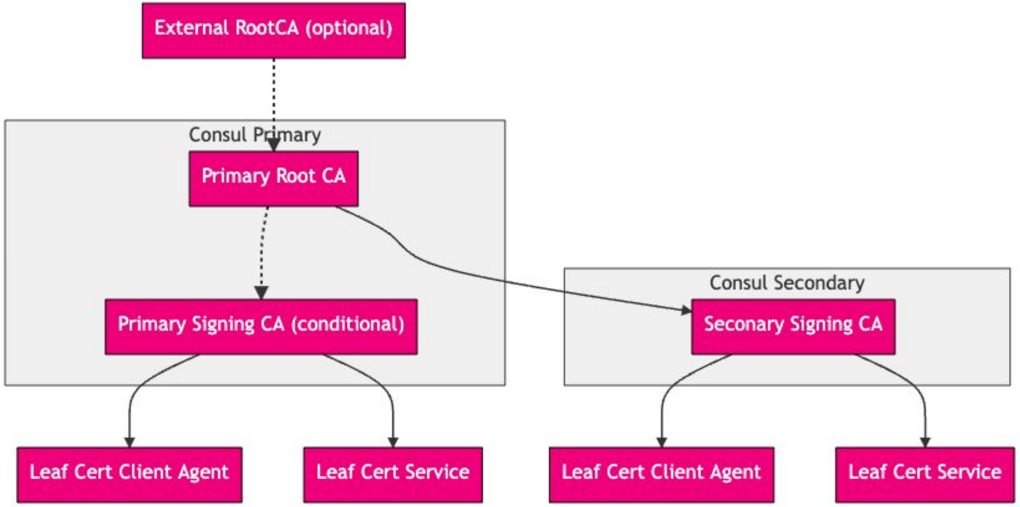


https://github.com/hashicorp/consul/tree/main/docs/service-mesh/ca

What we need to protect

Protecting Private Keys

- To do that, our leaf certificate is signed by a higher up CA—*Certificate Authorities*
- Which allows us to build a chain all the way up to something we ultimately trust



https://github.com/hashicorp/consul/tree/main/docs/service-mesh/ca

What we need to protect

Protecting Private Keys for Leaf Certificates

- In Consul, protecting the private keys of *Leaf Certificates* (i.e. certificates for actual service instances on the Service Mesh) is fairly easy

What we need to protect

Protecting Private Keys for Leaf Certificates

- In Consul, protecting the private keys of *Leaf Certificates* (i.e. certificates for actual service instances on the Service Mesh) is fairly easy
- When a given local Consul agent is asked to create a leaf certificate for a service, the Private Key is generated locally, and we ask the next layer up to sign our certificate

What we need to protect

Protecting Private Keys for Leaf Certificates

- In Consul, protecting the private keys of *Leaf Certificates* (i.e. certificates for actual service instances on the Service Mesh) is fairly easy
- When a given local Consul agent is asked to create a leaf certificate for a service, the Private Key is generated locally, and we ask the next layer up to sign our certificate
- The resultant certificate is public, so we don't care about that

What we need to protect

Protecting Private Keys for Leaf Certificates

- In Consul, protecting the private keys of *Leaf Certificates* (i.e. certificates for actual service instances on the Service Mesh) is fairly easy
- When a given local Consul agent is asked to create a leaf certificate for a service, the Private Key is generated locally, and we ask the next layer up to sign our certificate
- The resultant certificate is public, so we don't care about that
- The Private Key is only ever stored in memory—*never on disk*—even with the local agent caching that certificate and key in case the sidecar restarts and asks for the certificate while it is still valid

What we need to protect

Protecting Private Keys for CAs

- As we go up layers, we may encounter an intermediate Consul Signing CA

What we need to protect

Protecting Private Keys for CAs

- As we go up layers, we may encounter an intermediate Consul Signing CA
- Above those is the Primary Root CA

What we need to protect

Protecting Private Keys for CAs

- As we go up layers, we may encounter an intermediate Consul Signing CA
- Above those is the Primary Root CA
- Those are all just certificates, which happen to *sign other certificates*

What we need to protect

Protecting Private Keys

So if we have that key

We can sign anything

We can *be any service on the service mesh*

What we need to protect

Private Key Best Practices

- **NIST SP 800-57 Part 1 Revision 5 6.2.2.3 Confidentiality**: "One of the following mechanisms shall be used to provide confidentiality for secret key information in storage: 1. Encryption (or key wrapping) using an approved algorithm…."
- **CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates Version 1.8.6**: "The CA SHALL encrypt its Private Key with an algorithm and key-length that, according to the state of the art, are capable of withstanding cryptanalytic attacks for the residual life of the encrypted key or key part."

What we need to protect

Private Key Best Practices

- **NCSC Design and build a privately hosted Public Key Infrastructure Version 1**: "Securely storing your private keys will reduce the *likelihood* of compromise."

03 **Why you should use Vault as your Consul Certificate Authority**

Consul CA

How the Consul CA Treats Private Keys

Configuration

- These are the important bits of configuration
- Full configuration is in the presentation repository

```
datacenter = "dc-consul-ca"
data_dir ="/opt/consul/data"
connect {
  enabled = true
  ca_provider = "consul"
}
```

How the Consul CA Treats Private Keys

Examine Root Certificate

Examine Root Certificate

How the Consul CA Treats Private Keys

Raft Storage

- Consul uses the *Raft* protocol to do distributed consensus, and maintain distributed state

How the Consul CA Treats Private Keys

Raft Storage

- Consul uses the *Raft* protocol to do distributed consensus, and maintain distributed state
- When we say Consul is a highly available, replicated service, *that distributed state is what is replicated*

How the Consul CA Treats Private Keys

Raft Storage

- Consul uses the *Raft* protocol to do distributed consensus, and maintain distributed state
- When we say Consul is a highly available, replicated service, *that distributed state is what is replicated*
- In my configuration, `/opt/consul/data/raft/raft.db` is where Consul keeps its *Raft Log*

How the Consul CA Treats Private Keys

Raft Storage

- Consul uses the *Raft* protocol to do distributed consensus, and maintain distributed state
- When we say Consul is a highly available, replicated service, *that distributed state is what is replicated*
- In my configuration, `/opt/consul/data/raft/raft.db` is where Consul keeps its *Raft Log*
- There may also be snapshots in `/opt/consul/data/raft/snapshots`

How the Consul CA Treats Private Keys

Raft Storage

- Consul uses the *Raft* protocol to do distributed consensus, and maintain distributed state
- When we say Consul is a highly available, replicated service, *that distributed state is what is replicated*
- In my configuration, `/opt/consul/data/raft/raft.db` is where Consul keeps its *Raft Log*
- There may also be snapshots in `/opt/consul/data/raft/snapshots`
- Those combined hold *all* Consul data for that datacenter

How the Consul CA Treats Private Keys

Raft Storage

What can we find in there?

How the Consul CA Treats Private Keys

What does this mean?

- I have some private key that I found in the Consul server Raft log

How the Consul CA Treats Private Keys

What does this mean?

- I have some private key that I found in the Consul server Raft log
- That private key has one and only one corresponding public key, which we printed out

How the Consul CA Treats Private Keys

What does this mean?

- I have some private key that I found in the Consul server Raft log
- That private key has one and only one corresponding public key, which we printed out
- Which matches *exactly* the public key in the Consul Primary Root CA certificate

How the Consul CA Treats Private Keys

What does this mean?

- I have some private key that I found in the Consul server Raft log
- That private key has one and only one corresponding public key, which we printed out
- Which matches *exactly* the public key in the Consul Primary Root CA certificate
- Which means ***this is the key which signs everything***

How the Consul CA Treats Private Keys

What does this mean?

- Remember, Consul Service Mesh is an *identity-based networking tool*

How the Consul CA Treats Private Keys

What does this mean?

- Remember, Consul Service Mesh is an *identity-based networking tool*
- We connect services together based on their identity

How the Consul CA Treats Private Keys

What does this mean?

- Remember, Consul Service Mesh is an *identity-based networking tool*
- We connect services together based on their identity
- Which is proven on the wire *with certificates*

How the Consul CA Treats Private Keys

What does this mean?

- Remember, Consul Service Mesh is an *identity-based networking tool*
- We connect services together based on their identity
- Which is proven on the wire *with certificates*
- Which means that anyone who holds the signing key *can be* ***any service they want*** *on the Consul Service Mesh*

How the Consul CA Treats Private Keys

What does this mean?

With this key an attacker can be ***any service they want*** *on the Consul Service Mesh*

03 **Why you should use Vault as your Consul Certificate Authority**

Vault CA

How the Vault CA Treats Private Keys

Configuration

- These are the important bits of configuration
- Full configuration is in the presentation repository

```
datacenter = "dc-vault-ca"
data_dir ="/opt/consul/data"
connect {
  enabled = true
  ca_provider = "vault"
  ca_config {
    [...]
}
```

How the Vault CA Treats Private Keys

Examine Root Certificate

Examine Root Certificate

How the Vault CA Treats Private Keys

Consul Raft Storage

What can we find in there?

How the Vault CA Treats Private Keys

What Changes with Vault as your CA?

- If I set up Consul to use the Vault CA provider, all certificate signing operations are handled with Vault

How the Vault CA Treats Private Keys

What Changes with Vault as your CA?

- If I set up Consul to use the Vault CA provider, all certificate signing operations are handled with Vault
- The keying material—which is what we care about—is stored inside of Vault's cryptographic barrier

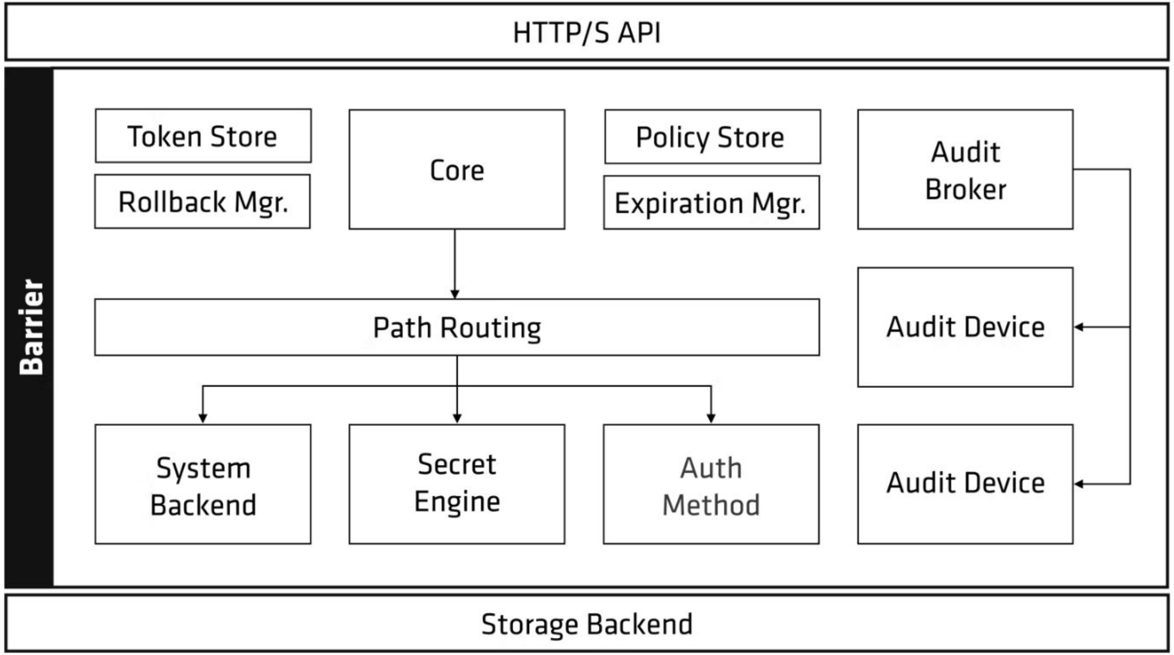
How the Vault CA Treats Private Keys

What Changes with Vault as your CA?

- If I set up Consul to use the Vault CA provider, all certificate signing operations are handled with Vault
- The keying material—which is what we care about—is stored inside of Vault's cryptographic barrier
- No keys ever live on disk unencrypted

How the Vault CA Treats Private Keys

What Changes with Vault as your CA?



https://developer.hashicorp.com/vault/docs/internals/architecture

How the Vault CA Treats Private Keys

What's in Vault storage?

What can we find in *there*?

How the Vault CA Treats Private Keys

What Changes with Vault as your CA?

- We get the security and logging of Vault signing our certificates when we leverage it in Consul

How the Vault CA Treats Private Keys

What Changes with Vault as your CA?

- We get the security and logging of Vault signing our certificates when we leverage it in Consul
- And we keep sensitive keying material safely protected

How the Vault CA Treats Private Keys

In Summary

Consul CA Provider

Private Key stored
unencrypted on disk

How the Vault CA Treats Private Keys

In Summary

Vault CA Provider

Private Key stored ***encrypted*** on disk

04 Why you should use Vault as your Consul Certificate Authority

Caveats to using Vault as your Consul CA

Caveats to using Vault as your Consul CA

Vault Access Tokens

- Astute viewers may point out that the `token` parameter (or `auth_method` parameters, if you use that) are stored on disk

```
connect {
  enabled = true
  ca_provider = "vault"
  ca_config {
    address = "http://127.0.0.1:8200"
    token =
"hvs.NNNNOOOOOOOOOOOOOOOOOOOOOOOOOO"
    [...]
  }
}
```

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

- Use an `auth_method` which makes sense for your environment—one which doesn't hardcode credentials on disk

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

- But if an attacker on the Consul server, and have either the token, or can use the `auth_method` configured, can't I just ask Vault to do things for me, the attacker?

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

- But if an attacker on the Consul server, and have either the token, or can use the `auth_method` configured, can't I just ask Vault to do things for me, the attacker?
- Potentially, yes, but you have also created additional opportunities for audit logging and analysis to detect anomalous behavior

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

- You are also adding a layer of defence for attacks on things like backups

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

- You are also adding a layer of defence for attacks on things like backups
- Any reasonable action you can take to make an attack more complicated, and increase the likelihood of detection, helps with your defence

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

- You are also adding a layer of defence for attacks on things like backups
- Any reasonable action you can take to make an attack more complicated, and increase the likelihood of detection, helps with your defence
- It may also provide additional data you can use post-attack to determine the full scope of the compromise

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

Defence in Depth

Caveats to using Vault as your Consul CA

Mitigating Vault Access Tokens

- Depending on your industry, you may have regulatory requirements which mandate that a private encryption key is never persisted to storage in an unencrypted form
- Or this may be part of your corporate risk mitigation strategy
- Using Vault as your Consul CA provides a path to meet those requirements

Caveats to using Vault as your Consul CA

Additional Complexity

- Using Vault as your Consul CA does mean "more moving parts"

Caveats to using Vault as your Consul CA

Additional Complexity

- Using Vault as your Consul CA does mean "more moving parts"
- Consul now has a dependency on Vault

Caveats to using Vault as your Consul CA

Additional Complexity

- Using Vault as your Consul CA does mean "more moving parts"
- Consul now has a dependency on Vault
- If Consul is unable to access Vault it isn't immediately fatal

Caveats to using Vault as your Consul CA

Additional Complexity

- Using Vault as your Consul CA does mean "more moving parts"
- Consul now has a dependency on Vault
- If Consul is unable to access Vault it isn't immediately fatal
- Certificates which are still valid will continue to work

Caveats to using Vault as your Consul CA

Additional Complexity

- Using Vault as your Consul CA does mean "more moving parts"
- Consul now has a dependency on Vault
- If Consul is unable to access Vault it isn't immediately fatal
- Certificates which are still valid will continue to work
- Anything requiring new certificates, however, will break
    - Normal certificate rotation
    - Generating certificates for new instances of a service

Caveats to using Vault as your Consul CA

Mitigating Additional Complexity

- Vault, just like Consul, can run in a highly-available cluster
- No single point of failure
- There is a management overhead, however

05 **Why you should use Vault as your Consul Certificate Authority**

Should I do it?

Should I use Vault as my Consul CA?

- That is a question only you can answer

Should I use Vault as my Consul CA?

- That is a question only you can answer
- I am certainly an advocate for it

Should I use Vault as my Consul CA?

- That is a question only you can answer
- I am certainly an advocate for it
- At minimum, I hope you have a good understanding now of the implications of *either* decision

Should I use Vault as my Consul CA?

- That is a question only you can answer
- I am certainly an advocate for it
- At minimum, I hope you have a good understanding now of the implications of *either* decision
- So you can make an *informed decision* to satisfy your particular situation and requirements

Thank You

thomashashi/vault-as-consul-ca

hugs@hashicorp.com | learn.hashicorp.com | discuss.hashicorp.com