Vault hashicorp

Vault es una herramienta que sirve para almacenar y controlar el acceso a tokens, contraseñas, certificados, claves de cifrado y otros datos confidenciales.

Características

- Vault es accesible por otros servicios y aplicaciones mediante APIs.
- Vault tiene mecanismos integrados que lo hacen resistente a fallas.
- Debido a la tecnología de replicación, Vault es muy escalable y puede proporcionar altas tasas de rendimiento para satisfacer la mayoría de las necesidades.
- Vault es capaz de cifrar/descifrar datos sin almacenarlos. La principal implicación de esto es que si ocurre una intrusión, el atacante no tendrá acceso a las claves reales.
- Vault puede almacenar claves por un período definido. Cuando este período expira, las claves se revocan automáticamente. Además, Vault admite una revocación manual flexible.
- Vault mantiene un historial de interacción con el y sus secretos.
- Vault admite la autenticación mediante tokens, lo cual es conveniente y seguro.
- Es posible conectar varios complementos a Vault para ampliar su funcionalidad.
- Vault tiene una interfaz gráfica de usuario basada (web), que se puede usar para interactuar con el sistema.

Instalación

- Deshabilitar el historial de la consola (bash):

```
history -c
echo "set +o history" >> ~/.bashrc
```

- Descargar https://www.vaultproject.io/downloads/
- Descomprimir y copiar al directorio de ejecutables

```
unzip vault-XYZ.zip
chown root:root vault
mv vault /usr/local/bin/
rm -f vault-XYZ.zip
```

- Instalar autocompletado

```
vault -autocomplete-install
```

- Crear directorios de datos de Vault

```
sudo mkdir /etc/vault
sudo mkdir -p /var/lib/vault/data
```

- Crear usuario llamado Vault

```
sudo useradd --system --home /etc/vault --shell /bin/false vault
sudo chown -R vault:vault /etc/vault /var/lib/vault/
```

- Crear el archivo de servicio para Vault

```
cat <<EOF | sudo tee /etc/systemd/system/vault.service
[Unit]
Description="HashiCorp Vault - A tool for managing secrets"
Documentation=https://www.vaultproject.io/docs/
Requires=network-online.target
After=network-online.target
ConditionFileNotEmpty=/etc/vault/config.hcl

[Service]
User=vault
Group=vault
ProtectSystem=full
ProtectHome=read-only
PrivateTmp=yes
PrivateDevices=yes
SecureBits=keep-caps
AmbientCapabilities=CAP_IPC_LOCK
NoNewPrivileges=yes
ExecStart=/usr/local/bin/vault server -config=/etc/vault/config.hcl
ExecReload=/bin/kill --signal HUP
KillMode=process
KillSignal=SIGINT
Restart=on-failure
RestartSec=5
```

```
TimeoutStopSec=30
StartLimitBurst=3
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
EOF
```

- Generar el certificado SSL y la llave privada (puede usar letsencrypt)
  - Certificado: `/etc/letsencrypt/live/entidad.gob.bo/cert.pem`
  - Llave privada: `/etc/letsencrypt/live/entidad.gob.bo//privkey.pem`
- Cree el archivo configh.hcl de Vault

```
cat <<EOF | sudo tee /etc/vault/config.hcl
disable_cache = true
disable_mlock = true
ui = false
plugin_directory="/etc/vault/plugins"
listener "tcp" {
  address          = "0.0.0.0:8200"
  tls_cert_file = "/etc/letsencrypt/live/entidad.gob.bo/cert.pem"
  tls_key_file  = "/etc/letsencrypt/live/entidad.gob.bo//privkey.pem"
}
storage "file" {
  path  = "/var/lib/vault/data"
 }
api_addr         = "https://0.0.0.0:8200"
max_lease_ttl         = "10h"
default_lease_ttl    = "10h"
cluster_name         = "vault"
raw_storage_endpoint     = true
disable_sealwrap     = true
disable_printable_check = true
EOF
```

- Inicie y habilite el servicio de Vault para iniciar con el arranque del sistema

```
sudo systemctl daemon-reload
sudo systemctl enable --now vault
systemctl status vault
```

Iniciando el servidor

Vault es una herramienta cliente-servidor que consta de un componente de cliente que utilizan los desarrolladores para las aplicaciones y un componente de servidor que se encarga de proteger los datos en servidores remotos.

- Exporte la variable de entrono VAULT_ADDR antes de inicalizar el servidor Vault

```
export VAULT_ADDR=https://127.0.0.1:8200
echo "export VAULT_ADDR=https://127.0.0.1:8200" >> ~/.bashrc
```

- Inicializar vault con 4 llaves compartidas de las cuales 3 serán requeridas para “abrir” el contenedor de llaves. Este comando también generara el token del root que se usa para la configuración inicial.

```
sudo rm -rf  /var/lib/vault/data/*
vault operator init -key-shares=4 -key-threshold=3
```

El comando generara las 4 llaves compartidas que deberán ser entregadas a diferentes personas para poder “abrir” vault y tener acceso a los datos privados que se almacenen en ella. Las personas a cargo de la custodia de estas llaves compartidas deberán almacenarlas de manera segura usando PGP (Pretty Good Privacy), pueden usar un servicio como https://keybase.io

- Para abrir Vault ejecute este comando 3 veces y copie una a una 3 de las 4 las llaves compartidas

```
vault operator unseal
```

- Autenticarnos con el token del root generado con el comando anterior

```
vault login <<token root>>
```

- Habilitar pistas de auditoria

```
vault audit enable syslog tag="vault" facility="AUTH"
```

- Instalar syslog

```
apt-get install syslog-ng
```

- Agregar al archivo `/etc/syslog-ng/syslog-ng.conf`

```
source s_net { udp(ip(0.0.0.0) port(514)); };
filter f_vault { host( "0.0.0.0" ); };
destination df_vault { file("/var/log/vault.log"); };
log { source ( s_net ); filter( f_vault ); destination ( df_vault ); };
```

Almacenamiento de datos privados

- Habilitar el motor de secretos llave-valor (KV)

```
vault secrets enable -version=1 kv
```

- Listar los motores de secretos

```
vault secrets list
```

- Almacenar una API

```
vault kv put kv/apikey/Google llave=AAaaBBccDDeeOTXzSMT1234BB_Z8JzG7JkSVxI
```

- Leer la llave almacenada

```
vault kv get kv/apikey/Google
```

Código de ejemplo de una aplicación en python que se comunica con Vault usando un token para obtener el API key:

```
    response = requests.get(
    'https://127.0.0.1:8200/v1/kv/apikey/Google',
    params={'q': 'requests+language:python'},
    headers={'X-Vault-Token': 's.VjOyZANgP89UEQSVcXNKdZOi'},
    )
    json_response = response.json()
    APIKey = json_response['data']['llave']
```

- Almacenar el certificado cert.pem

```
vault kv put kv/cert/mysql certificado=@cert.pem
```

- Leer el certificado almacenado

```
vault kv get -field=certificado kv/cert/mysql
```

Instalar módulo de generación de contraseñas robustas

- Descargar y descomprimir el último plu-in binario

```
mkdir /etc/vault/plugins/
cd /etc/vault/plugins/
wget https://github.com/sethvargo/vault-secrets-gen/releases/download/v0.0.6/vault-secrets-
gen_0.0.6_linux_amd64.tgz
tar -zxvf vault-secrets-gen_0.0.6_linux_amd64.tgz
```

- Habilite mlock para que el complemento se pueda habilitar y deshabilitar de forma segura

```
setcap cap_ipc_lock=+ep /etc/vault/plugins/vault-secrets-gen
```

- Calcule el SHA256 del complemento y regístrelo en el catálogo de complementos de Vault

```
export SHA256=$(shasum -a 256 "/etc/vault/plugins/vault-secrets-gen" | cut -d' ' -f1)

vault plugin register \
    -sha256="${SHA256}" \
    -command="vault-secrets-gen" \
    secret secrets-gen
```

- Monta el motor de secretos

```
vault secrets enable \
    -path="gen" \
    -plugin-name="secrets-gen" \
    plugin
```

- Generar un password

```
vault write gen/password length=36
```

Secretos dinámicos

Para incrementar la seguridad de una aplicación se puede usar la funcion de “secretos dinámicos” de Vault que permite crear dinámicamente usuarios y contraseñas temporales de la base de datos para que sean usados por una aplicación y así no exponer el usuario y contraseña.

Este ejemplo asume que mongdb tiene habilitado la autenticación para las credenciales:

usuario="sam"
password="test123

- Habilitar el módulo de base de datos

```
vault secrets enable database
```

- Configure Vault para que se conecte a mongoDB

```
vault write database/config/my-mongodb-database \
    plugin_name=mongodb-database-plugin \
    allowed_roles="my-role" \
    connection_url="mongodb://{{username}}:{{password}}@127.0.0.1:27017/admin" \
    username="sam" \
    password="test123"
```

- Configure Vault para que cree una credencial en la base de datos y expire cada hora (default_ttl)

```
vault write database/roles/my-role \
    db_name=my-mongodb-database \
    creation_statements='{ "db": "admin", "roles": [{ "role": "readWrite" }] }' \
    default_ttl="1h" \
    max_ttl="24h"
```

- Probar manualmente la generación de credenciales

```
vault read database/creds/my-role
```

Código de ejemplo de una aplicación en python que se comunica con Vault usando un token para obtener un usuario y contraseña de la base de datos:

```
response = requests.get(
'http://127.0.0.1:8200/v1/database/creds/my-role',
params={'q': 'requests+language:python'},
headers={'X-Vault-Token': 's.VjOyZANgP89UEQSVcXNKdZOi'},
)
json_response = response.json()
Database.USER = json_response['data']['username']
Database.PASSWORD = json_response['data']['password']
```

En el siguiente punto se muestra como generar tokens que puede tener restricciones como:

1. Tiempo de vida
2. Permiso granular de acceso a Vault
3. Uso máximo

Habilitar AppRole para la autenticación de aplicaciones con vault

El método AppRole permite que aplicaciones se autentiquen con roles definidos por Vault.

Habilitar la autenticación para aplicaciones

```
vault auth enable approle
```

Crear el archivo "database.hcl” que describe la política para la aplicación

```
cat <<EOF | sudo tee /etc/vault/database.hcl
path "database/creds/my-role" {
 capabilities = ["read", "list"]
}
EOF
```

Cargar la política a vault

```
vault policy write database /etc/vault/database.hcl
```

- Crear un rol con la política “database” y con las siguientes restricciones

1. Tiempo de vida del token 10 minutos
2. Tiempo de vida del secret-id 1 minuto
3. Uso máximo de tokens: 3 veces

```
vault write auth/approle/role/databaseApp secret_id_ttl=1m secret_id_num_uses=1 token_num_uses=3
token_ttl=10m token_max_ttl=30m policies=database
```

- Revisar los parámetros configurados

```
vault read auth/approle/role/databaseApp
```

- Obtener el role-id

```
vault read auth/approle/role/databaseApp/role-id
```

- Generar el secret-id

```
vault write -f auth/approle/role/databaseApp/secret-id
```

- Con el role-id y secret-id obtenido con los comandos anteriores podemos obtener un token a la que se le aplico la política creada anteriormente

```
vault write auth/approle/login role_id="XXXXX" secret_id="YYYYYYY"
```

- Revocar el token del root

```
vault token revoke s.V6T0DxxIg5FbBSre61y1WLgmalid
```

Hardening

- Vault siempre debe usarse con TLS en producción
- Vault debería ser el único proceso principal que se ejecuta en una máquina. Preferentemente debe ejecutarse en un servidor físico, sino es posible tener un servidor dedicado usar una máquina virtual. No es recomendable usar contenedores.
- Use un firewall local para restringir todo el tráfico entrante y saliente a Vault y servicios esenciales del sistema como NTP
- Los usuarios nunca deben acceder a la máquina directamente. En cambio, deberían acceder a Vault a través de su API a través de la red.
- Para el funcionamiento normal existen datos confidenciales en memoria. El riesgo de exposición debe minimizarse deshabilitando la memoria SWAP

- Vault está diseñado para ejecutarse como un usuario sin privilegios, y no hay razón para ejecutar Vault con privilegios de administrador o root
- Desactivar volcados de memoria (Core Dump)
- Evita los tokens de root. Este token debe usarse solo para configurar el sistema inicialmente
- El uso de mecanismos adicionales como SELinux y AppArmor puede ayudar a proporcionar capas adicionales de seguridad
- El acceso al backend de almacenamiento debe restringirse solo a Vault para evitar accesos no autorizados.
- Deshabilitar el historial de comandos de Shell

Revision #3
Created 7 marzo 2023 09:32:41 by Franz Rojas
Updated 10 abril 2024 11:05:56 by Vladimir Urquiola