Установка, настройка и работа с Hashicorp Vault

Обновлено: 28.05.2021 Опубликовано: 19.05.2021

Используемые термины: Hashicorp Vault

В данной инструкции попробуем охватить как можно больше примеров работы с Hashicorp Vault. Мы выполним установку на системы Linux, настоим сервер, сохраним несколько секретов и попробуем получить доступ к данным секретам из различных систем. В качестве Linux рассмотрим Debian и CentOS 7 и 8.

Подготовка

Прежде чем начать, приведем наш сервер в готовность. Установим необходимые пакеты, настроим время и систему безопасности.

Установка пакетов

Нам понадобятся некоторые утилиты. Команды для их установки зависят от используемой системы.

а) Debian:

```
apt-get install wget chrony curl apt-transport-https
```

б) CentOS:

```
yum install wget chrony curl
```

** где:*

- ***wget —** утилита для загрузки файлов.*
- ***chrony —** сервис для синхронизации времени.*
- ***curl —** утилита для отправки POST и GET запросов на веб-сервер.*
- ***apt-transport-https —** дополнение для возможности использовать репозитории по https.*

Настройка времени

Для корректного получения токенов необходимо, чтобы время на сервере было правильное. Задаем необходимый нам часовой пояс:

```
timedatectl set-timezone Europe/Moscow
```

** полный перечень вариантов можно посмотреть командой **timedatectl list-timezones**.*

Если у нас в сети есть свой сервер синхронизации времени, открываем на редактирование файл настройки chrony.

а) Debian:

```
vi /etc/chrony/chrony.conf
```

б) CentOS:

```
vi /etc/chrony.conf
```

Нам нужно поменять источник, с которым мы будем синхронизировать наше время. Данная опция отличается в зависимости от версии системы или дистрибутива.

а) в Debian или CentOS 8:

```
pool dmosk.local
#pool ...
```

** в нашем примере мы комментируем тот адрес **pool**, который был в конфигурации и подставляем адрес нашего сервера **dmosk.local**.*

а) в CentOS 7:

```
server dmosk.local
#server 0.centos.pool.ntp.org iburst
#server 1.centos.pool.ntp.org iburst
#server 2.centos.pool.ntp.org iburst
#server 3.centos.pool.ntp.org iburst
```

** в нашем примере мы комментируем адреса **server**, которые были в конфигурации и подставляем адрес нашего сервера **dmosk.local**.*

Разрешаем автоматический запуск для сервиса синхронизации времени и перезапускаем его.

а) для Debian:

```
systemctl enable chrony

systemctl restart chrony
```

б) для CentOS:

```
systemctl enable chronyd
```

```
systemctl restart chronyd
```

Настройка брандмауэра

Для корректной работы сервиса нам необходимо открыть порт 8200. В зависимости от используемой утилиты управления netfilter мы должны применять разные инструменты.

а) Iptables (как правило, Debian):

```
iptables -I INPUT -p tcp --dport 8200
-j ACCEPT
```

Для сохранения правила можно воспользоваться утилитой iptables-persistent:

```
apt-get install iptables-persistent

netfilter-persistent save
```

б) Firewalld (как правило, для CentOS):

```
firewall-cmd --permanent --add-
port=8200/tcp

firewall-cmd --reload
```

Установка и запуск

Программный продукт поддерживает различные варианты установки. Мы рассмотрим установку из репозитория.

Подключаем официальный репозиторий и устанавливаем пакет vault.

а) Debian:

```
curl -fsSL
https://apt.releases.hashicorp.com/gpg
```

```
| sudo apt-key add -

echo "deb [arch=amd64]
https://apt.releases.hashicorp.com
$(lsb_release -cs) main" >
/etc/apt/sources.list.d/hashicorp.list

apt-get update

apt-get install vault
```

б) CentOS:

```
wget
https://rpm.releases.hashicorp.com/RHE
L/hashicorp.repo -O
/etc/yum.repos.d/hashicorp.repo

yum install vault
```

Разрешаем автозапуск службы и если она не запущена, стартуем ее:

```
systemctl enable vault --now
```

Установка выполнена. При попытке зайти по адресу https://<IP-адрес сервера Vault>:8200/ мы должны увидеть страницу начальной настройки мастер-ключей.

Идем дальше.

Настройка рабочего окружения

При попытке выполнить любую операцию в командной строке, мы получим ошибку:

```
Error checking seal status: Get
"https://127.0.0.1:8200/v1/sys/seal-
status": x509: cannot validate
```

```
certificate for 127.0.0.1 because it
doesn't contain any IP SANs
```

Это значит, что мы пытаемся подключиться к нашему серверу по https с неправильным сертификатом. На этапе первичной настройки не хочется заниматься получением и настройкой валидного сертификата. В официальной документации сказано, что нужно ввести команду:

```
export
VAULT_ADDR=http://127.0.0.1:8200
```

Она укажет текущему окружению подключаться к серверу по http. Однако, это приведет к другой ошибке:

```
Error checking seal status: Error
making API request.

URL: GET
http://127.0.0.1:8200/v1/sys/seal-
status
Code: 400. Raw Message:

Client sent an HTTP request to an
HTTPS server.
```

Она говорит, что мы подключаемся к серверу по незащищенному каналу, когда сервер требует безопасного соединения.

Для решения проблемы открываем файл:

```
vi /etc/vault.d/vault.hcl
```

И снимаем комментарии со следующих строк, а также меняем значение для поля **address**:

```
listener "tcp" {
  address = "127.0.0.1:8201"
  tls_disable = 1
}
```

** так как для https уже используется порт **8200**, мы должны поменять предложенный по умолчанию вариант на свой, чтобы не возникало конфликтов при запуске сервиса.*

Перезапускаем службу vault:

```
systemctl restart vault
```

Обновляем системную переменную VAULT_ADDR:

```
export
VAULT_ADDR=http://127.0.0.1:8201
```

** обратите внимание, что мы поменяли порт подключения на **8201**.*

Пробуем вывести на экран статус:

```
vault status
```

Мы должны получить что-то на подобие:

```
Key                Value
---                -----
Seal Type          shamir
Initialized        false
Sealed             true
Total Shares       0
Threshold          0
Unseal Progress    0/0
Unseal Nonce       n/a
Version            1.7.1
Storage Type       file
HA Enabled         false
```

Чтобы данная системная переменная создавалась каждый раз при входе пользователя в систему, открываем файл:

```
vi /etc/environment
```

И добавляем:

```
VAULT_ADDR=http://127.0.0.1:8201
```

Распечатывание

После установки, сервер Vault находится в запечатанном (sealed) состоянии. То есть, он не знает, как ему расшифровывать секреты, которые будут храниться в базе.

При попытке выполнить любую операцию с хранилищем секретов мы получим ошибку:

```
* Vault is sealed
```

Чтобы исправить ситуацию, нужно выполнить инициализацию сервера — мы получим ключи для распечатывания (Unseal Keys). После необходимо ввести эти ключи и можно будет авторизоваться в системе.

Инициализация, распечатывание и вход

Для начала, нам необходимо инициализировать наш сервер. Это выполняется командой:

```
vault operator init
```

Команда нам вернет 5 ключей. Любые 3 из них являются ключами для распечатывания сервера (Unseal Key). Также нам будет предоставлен токен для root (Initial Root Token), с помощью которого можно будет войти в систему vault.

И так, распечатаем наш сервер, введя по очереди 3 команды.

Для первого ключа:


```
vault operator unseal
```

Для второго:

```
vault operator unseal
```

И третьего:

```
vault operator unseal
```

После выполнения каждой команды система будет запрашивать ключ. Необходимо ввести любые 3 из сгенерированных ранее. При правильном вводе ключа, мы будем видеть общую информацию по ключу. А при вводе третьего ключа мы должны увидеть:

```
Sealed          false
```

Это значит, что сервер больше не запечатан и с ним можно работать.

Теперь необходимо залогиниться в систему командой:

```
vault login
```

... и ввести ключ root, который мы получили после инициализации.

Мы можем выполнять команды для работы с Hashicorp Vault.

Автоматическое распечатывание

После перезагрузки нашего сервера он опять становится запечатанным. Предполагается, что полученных 5 ключей необходимо выдать пяти ответственным за безопасность сотрудникам. После перезагрузки, трое из них должны будут ввести свои данные. Это правильно с точки зрения безопасности.

Однако, если у нас есть причины автоматически поднимать сервер после перезагрузки, рассмотрим, как это сделать.

> Данный способ противоречит безопасности, однако, реальная эксплуатация систем не всегда вписывается в рамки идеальных концепций.

Выполним настройку в несколько шагов.

1. Скрипт.

Создадим каталог для хранения скриптов:

```
mkdir /scripts
```

Создадим скрипт:

```
vi /scripts/unseal.sh
```

```
#!/bin/bash
PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbi
n:/usr/local/bin:/usr/local/sbin

sleep 10
vault operator unseal
w1SgHSWyXm+7kwmYk4bFX2rBLG5jKxIn01DMkj
57071D
vault operator unseal
38s4+FkxKTTANFZgCwEPFOgJIMwTvLca1j36yY
Pc3gdx
vault operator unseal
4xlpKVwPuNlskydM/qmCmW22x7WZdfuiFu92HG
RNOa8o
```

** где*
***w1SgHSWyXm+7kwmYk4bFX2rBLG5jKxIn01DMkj57071D**,*
38s4+FkxKTTANFZgCwEPFOgJIMwTvLca1j36yYPc3gdx
и
4xlpKVwPuNlskydM/qmCmW22x7WZdfuiFu92HGRNOa8o
— любых 3 токена (из 5 сгенерированных).

*Обратите внимание, мы задерживаем выполнение скрипта на 10 секунд — на практике, сервис vault может не успеть запуститься, и мы тогда получим ошибку при выполнении команды **vault operator unseal**.*

Разрешаем запуск скрипта на выполнение:

```
chmod +x /scripts/unseal.sh
```

Можно, даже, выполнить скрипт:

```
/scripts/unseal.sh
```

В итоге, мы распечатаем наш сервер.

2. Автозапуск скрипта при старте системы:

Большинство современных серверных систем работает на основе systemd. Рассмотрим автозапуск с помощью последней.

Создаем юнит:

```
vi /etc/systemd/system/vault-
unseal.service
```

```
[Unit]
Description=Vault Auto Unseal Service
After=network.target
After=vault.service

[Service]
Environment="VAULT_ADDR=http://127.0.0
.1:8201"
ExecStart=/scripts/unseal.sh
Type=oneshot
RemainAfterExit=no

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

** в данном примере мы выполняем одну команду при запуске сервиса — запуск скрипта /scripts/unseal.sh.*

*Перед этим мы создаем системную переменную **VAULT_ADDR**, чтобы при выполнении команд в скрипте система понимала, к какому серверу подключаться.*

Перечитаем конфигурацию для systemd:

```
systemctl daemon-reload
```

Разрешаем автозапуск созданного сервиса:

```
systemctl enable vault-unseal
```

Пробуем перезагрузить сервер — vault должен оказаться распечатанным.

Работа с секретами

Наша система полностью готова, чтобы ей пользоваться.

Для начала дадим разрешение на хранение секретов по пути secret:

```
vault secrets enable -path=secret/ kv
```

Мы должны получить ответ:

```
Success! Enabled the kv secrets engine
at: secret/
```

Теперь выполним простую операцию по созданию секрета. Введем команду, предложенную на официальном сайте разработчика:

```
vault kv put secret/hello foo=world
```

** команда **kv** взаимодействует с хранилищем Vault. В данном примере мы добавляем пару ключ-значение, соответственно **foo-world**.*

Мы должны получить ответ:

```
Success! Data written to: secret/hello
```

Посмотреть содержимое можно командой:

```
vault kv get secret/hello
```

На что мы получим:

```
===== Data =====
Key         Value
---         -----
foo         world
```

Также можно внести за раз множество значений:

```
vault kv put secret/hello foo=world
excited=yes
```

Посмотреть список созданных секретов можно командой:

```
vault kv list secret
```

Теперь удалим секрет командой:

```
vault kv delete secret/hello
```

Полный перечень команд можно увидеть так:

```
vault -help
```

После чего можно будет получить помощь по конкретной команде vault, например:

```
vault kv -help
```

Данной информации достаточно, чтобы познакомиться с продуктом. Пойдем дальше.

Версионность и метаданные

Hashicorp Vault поддерживает версионность данных, то есть, мы можем посмотреть определенную версию данных внутри секрета. Однако, по умолчанию, сервис устанавливается с kv версии 1, которая не поддерживает данной операции. При попытке выполнить любую команду, захватывающую версионность, мы получим ошибку:

```
Metadata not supported on KV Version 1
```

Для решения необходимо создание механизма kv с поддержкой версии 2. Это делается командой:

```
vault secrets enable -version=2 kv
```

Или для ранее созданного пути включаем ведение версионности командой:

```
vault kv enable-versioning secret/
```

Теперь занесем 2 раза данные:

```
vault kv put secret/hello foo=world
```

```
vault kv put secret/hello foo=world2
```

Посмотреть данные определенной версии можно командой:

```
vault kv get -version=1 secret/hello
```

В нашем примере мы увидим:

```
=== Data ===
Key    Value
---    -----
foo    world
```

** то есть значение, которое вводилось первой командой.*

Посмотреть метаданные секрета можно командой:

```
vault kv metadata get secret/hello
```

Для удаления определенной версии секрета вводим:

```
vault kv destroy -versions=1
secret/hello
```

Динамические секреты

Мы можем настроить автоматическое создание временных пользователей в различных системах. Это удобно для автоматического назначения прав приложению. При этом, через некоторое время пароль уже не будет действовать, что повышает безопасность системы.

Рассмотрим пример настройки динамических секретов для базы данных MariaDB/MySQL.

Первым делом, создадим пользователя в СУБД с правами создавать других пользователей. Подключаемся к базе данных:

```
mysql -uroot -p
```

Создаем учетную запись, под которой будет подключаться vault к СУБД:

```
> CREATE USER 'vaultuser'@'localhost'
IDENTIFIED BY 'vaultpass';
```

Дадим права созданной учетной записи создавать других пользователей и назначать им права:

```
> GRANT CREATE USER ON *.* TO
'vaultuser'@'localhost' WITH GRANT
OPTION;
```


Выходим из оболочки SQL:

```
> exit;
```

Теперь разрешаем механизм секретов базы данных (database secrets engine):

```
vault secrets enable database
```

Мы должны увидеть:

```
Success! Enabled the database secrets
engine at: database/
```

** также мы можем получить ошибку path is already in use at database/, если данный механизм уже разрешен в нашей системе.*

Создадим настройку для подключения к нашей базе:

```
vault write database/config/test \
 plugin_name=mysql-database-plugin \
 connection_url="{{username}}:
{{password}}@tcp(127.0.0.1:3306)/" \
 allowed_roles="test-role" \
 username="vaultuser" \
 password="vaultpass"
```

** предполагается, что:*

- ***test —** имя базы, для которой vault сможет создать временную учетную запись.*
- ***plugin_name —** имя плагина для создания учетной записи. Важный параметр, который должен правильно быть подобран в зависимости от версии СУБД. Ниже будет комментарий, с какой ошибкой мы можем столкнуться, выбрав неправильный плагин.*
- ***test-role —** роль, которая будет использоваться vault (ее создадим ниже).*
- ***vaultuser —** пользователь, под которым мы подключаемся к СУБД (создали выше).*
- ***vaultpass —** пароль, с которым мы подключаемся к серверу баз данных.*

Создадим роль, по сути, зададим команду для создания временной учетной записи:

```
vault write database/roles/test-role \
 db_name=test \
 creation_statements="CREATE USER
'{{name}}'@'%' IDENTIFIED BY
'{{password}}'; GRANT SELECT ON *.* TO
'{{name}}'@'%';" \
 default_ttl="1h" \
 max_ttl="24h"
```

** где:*

- ***db_name —** имя базы данных, для которой мы будем создавать пользователя.*
- ***creation_statements —** команды SQL для создания нового пользователя. {{name}} и {{password}} являются специальными переменными, вместо которых vault подставит сгенерированные значения. Обратите внимание, что в данном примере создается пользователь с правами подключения с любого хоста, однако, в зависимости от версии MariaDB, именно с локального хоста подключиться под данным пользователем будет нельзя. При такое необходимости, меняем **%** на **localhost**.*
- ***default_ttl —** время, в течение которого будет действовать пароль.*
- ***max_ttl —** время, в течение которого пользователю можно будет обновлять пароль.*

И так, теперь давайте создадим пользователя и пароль:

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

** после ввода команды vault сгенерирует и создаст в базе нового пользователя и пароль.*

Мы должны увидеть что-то на подобие:

```
Key                Value
---                -----
lease_id
database/creds/test-
role/JpEmEh2MJV115Lr4S4Lx5UHW
lease_duration     1h
lease_renewable    true
password           7v9jC-
XHXJjQ2sicLI42
username           v-test--gVPavnGVr
```

** Если мы увидим ошибку, на подобие **Error 1470: String 'v-root-test-role-KHWyA2IwdoaUth7c' is too long for user name (should be no longer than 16)** мы столкнулись с ограничением на стороне СУБД (нельзя создавать имя пользователя с длиной более 16 символов). Чтобы решить проблему мы должны пересоздать подключение **database/config/test** с новым плагином **mysql-legacy-database-plugin**.*

Также для создания пользователя мы можем использовать API запрос:

```
curl --header "X-Vault-Token:
s.e0YKJEOHlpfgQiWqZlzVuUbY"
http://127.0.0.1:8201/v1/database/cred
s/test-role
```

** где X-**Vault-Token —** токен с доступом. Для теста можно использовать тот, что был получен для root после инициализации системы.*

Теперь давайте проверим, что созданные учетные записи имеют нужный нам доступ. Для начала, под пользователем root можно проверить, что записи созданы:

```
> SELECT user, host FROM mysql.user;
```

Мы должны увидеть в списках нужную запись (в нашем примере, **v-test--gVPavnGVr**).

Теперь зайдем под учетной записью, которая была создана с помощью vault:

```
mysql -h192.168.0.15 -u'v-test--
gVPavnGVr' -p'7v9jC-XHXJjQ2sicLI42'
```

** в моем случае из-за доступа %, подключение с localhost было запрещено. Поэтому я проверяю доступ с другого хоста в локальной сети, добавив опцию **-h192.168.0.15** (где 192.168.0.15 — сервер с СУБД, доступ к которой предоставлялся через vault).*

Мы должны подключиться к базе.

Отменить доступ можно командой:

```
vault lease
revoke database/creds/test-
role/JpEmEh2MJV115Lr4S4Lx5UHW
```

** где **database/creds/test-role/JpEmEh2MJV115Lr4S4Lx5UHW —** значение lease_id, которое нам вернула система при создании временной учетной записи.*

Посмотреть список всех идентификатором можно командой:

```
vault list
sys/leases/lookup/database/creds/test-
role
```

** где **test-role —** имя созданной нами роли.*

Продлить lease_duration или default_ttl (пароль) можно с помощью команды:

```
vault lease renew database/creds/test-
role/JpEmEh2MJV115Lr4S4Lx5UHW
```


Аутентификация и политики

Hashicorp Vault позволяет управлять доступами с помощью политик и токенов авторизации.
Рассмотрим процесс настройки.

Работа с токенами

Для начала научимся управлять токенами. Простая команда для создания нового:

```
vault token create
```

Система сгенерирует новый ключ и сделает вывод на экран.

Также мы можем создать временный токен:

```
vault token create -period=1h
```

** на один час.*

Посмотреть информацию о токене можно через его аксессор. Сначала получим список аксессоров:

```
vault list auth/token/accessors
```

После смотрим токен по аксессору:

```
vault token lookup -
accessor uW9Ajr8VzFiCwHzHWn75qWVe
```

Войти в систему с помощью токена можно командой:

```
vault login
```

После вводим наш токен. Или одной командой:

```
vault login s.Db9j6Q4TvyFDr3j2aQmXttrX
```

Посмотреть информацию о токене, под которым мы зарегистрировались в системе, можно командой:

```
vault token lookup
```

А данной командой мы создаем пользователя и привязываем его к политике **my-policy**:

```
vault token create -policy=my-policy
```

Если политики нет в системе, то мы получим предупреждение:

```
WARNING! The following warnings were
returned from Vault:

  * Policy "my-policy" does not exist
```

Нас это не должно заботить — на следующем шаге мы ее создадим.

При необходимости привязать токен к нескольким политикам, перечисляем из в опциях policy:

```
vault token create -policy=my-policy -
policy=my-policy2
```

Работа с политиками

Выше мы создали токен и привязали его к политике **my-policy**. Создадим ее:

```
vault policy write my-policy - << EOF
path "secret/data/foo/*" {
  capabilities = ["read", "create",
"update"]
}

path "secret/data/hello/*" {
  capabilities = ["read", "update"]
}

path "secret/data/*" {
  capabilities = ["read"]
```

```
}
EOF
```

** в данной политике мы разрешаем чтение всех секретов в **secret**. Для ранее созданного секрета **secret/hello** мы разрешим чтение и обновление записей, а для секрета **secret/foo** также мы разрешаем создавать записи. Обратите внимание, что на самом деле, данные хранятся в **secret/data**...*

Посмотреть список политик можно командой:

```
vault policy list
```

Посмотреть информацию о конкретной политике можно командой:

```
vault policy read my-policy
```

Проверка политики

И так, мы создали токен, привязали его к политике, создали саму политику. Проверим, что наши права работают.

Если нужно, можно создать еще токен и привязать его к нашей политике:

```
vault token create -policy=my-policy
```

Зарегистрируемся с нужным нам токеном (который привязан к проверяемой политике):

```
vault login
```

Попробуем сначала создать секрет в ветке secret/foo и в нем указать пару ключ-значение:

```
vault kv put secret/foo/new foo=world
```

Команда должна выполниться успешно.

Теперь сделаем то же самое для **secret/hello**

```
vault kv put secret/hello/new
foo=world
```

Мы должны получить ошибку

```
Error writing data to
secret/data/hello/new: Error making
API request.

URL: PUT
http://127.0.0.1:8201/v1/secret/data/h
ello/new
Code: 403. Errors:

* 1 error occurred:
    * permission denied
```

Теперь логинимся под токеном с полными правами и добавим запись:

```
vault kv put secret/hello/new
foo=world
```

И снова заходим под токеном с ограниченными правами. Пробуем обновить запись:

```
vault kv put secret/hello/new
foo=world2
```

Команда должна выполниться успешно, так как мы разрешили политикой обновлять записи.

Аутентификация на основе пользователей

Также в Hashicorp Vault мы можем создать пользователя с паролем и привязать к нему

политику. Администратор сможем входить в систему под учетной записью, а не с помощью токена.

Разрешаем метод auth по пути userpass:

```
vault auth enable userpass
```

Создаем пользователя:

```
vault write auth/userpass/users/dmosk
password="test-pass" policies="my-
profile"
```

** данной командой мы создали пользователя dmosk с паролем **test-pass** и привязали его к политике **my-profile**.*

Войти в систему можно командой:

```
vault login -method=userpass
username=dmosk
```

Вводим пароль и попадаем в систему под пользователем dmosk. Можно выполнить тесты, которые мы делали выше.

Одноразовые SSH пароли

Рассмотрим варианты хранения секретов для авторизации по SSH с использованием одноразовых паролей или One-Time SSH Password (OTP).

Настройка на сервере Vault

Разрешаем механизм ssh:

```
vault secrets enable ssh
```

Создаем роль для создания одноразового пароля:

```
vault write ssh/roles/otp_key_role \
 key_type=otp \
 default_user=test \
 cidr_list=10.0.2.0/24
```

** в данном примере:*

- ***key_type —** тип создаваемого ключа. Оставляем otp.*
- ***default_user —** имя учетной записи.*
- ***cidr_list —** подсеть, для которой будет разрешен доступ. Это должна быть подсеть компьютеров, на которых мы будем проходить аутентификацию по одноразовому паролю.*

На сервере, пока, все. Переходим к настройке клиента.

Настройка на клиенте

На клиенте необходимо выполнить настройку, которая позволит выполнять две задачи:

1. Создавать одноразовые пароли на стороне Vault.
2. Осуществлять проверку подлинности при подключении по SSH.

Данной задачей занимается vault-ssh-helper. Рассмотрим процесс его установки и настройки.

Нам понадобится пакет для распаковки zip-архивов. В зависимости от системы ставим его одной из команд ниже.

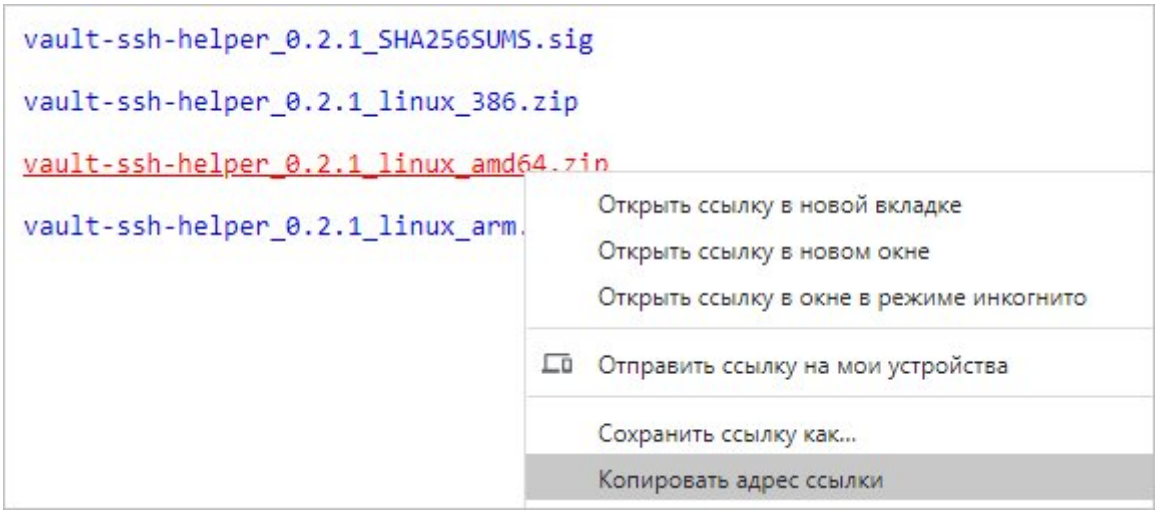
а) для Debian / Ubuntu:

```
apt-get install unzip
```

б) для CentOS / Red Hat / Fedora:

```
yum install unzip
```

Установка vault-ssh-helper выполняется простым копированием бинарника. Переходим на официальный сайт для загрузки и выбираем последнюю версию пакета. На следующей страницы копируем ссылку на нужный вариант архива, в зависимости от архитектуры нашего сервера:



С помощью скопированной ссылки загружаем архив:

```
wget
https://releases.hashicorp.com/vault-
ssh-helper/0.2.1/vault-ssh-
helper_0.2.1_linux_amd64.zip
```

Распаковываем его:

```
unzip vault-ssh-helper_*.zip -d
/usr/bin
```

Создадим каталог для конфигурационных файлов vault-ssh-helper:

```
mkdir /etc/vault-ssh-helper
```

Создадим конфигурационный файл:

```
vi /etc/vault-ssh-helper/config.hcl
```

```
vault_addr =
"https://192.168.0.20:8200"
```

Есть вопрос?

```
ssh_mount_point = "ssh"
tls_skip_verify = true
allowed_roles = "*"
```

** где:*

- ***vault_addr —** адрес нашего сервера секретов.*
- ***ssh_mount_point —** путь на стороне Vault, по которому будут храниться секреты. Настраивается при включении механизма SSH.*
- ***tls_skip_verify —** стоит ли пропускать проверку подлинности сертификата. Так как у нас еще не настроен валидный сертификат, устанавливаем значение в true.*
- ***allowed_roles —** на стороне сервера можно создавать разным пользователей от разных ролей. В данной настройке мы можем определить, пользователям с какой ролью будет разрешен вход по SSH.*

Открываем конфигурационный файл pamd для sshd:

```
vi /etc/pam.d/sshd
```

И в самый верх добавляем:

```
auth sufficient pam_exec.so quiet
expose_authtok log=/tmp/vaultssh.log
/usr/bin/vault-ssh-helper -
config=/etc/vault-ssh-
helper/config.hcl
auth optional pam_unix.so not_set_pass
use_first_pass nodelay
...
```

** в данной конфигурации мы, по-прежнему, сможем авторизоваться под учетными записями, созданными на самом сервере. Если необходимо это отключить, разрешив вход только с OTP, комментируем **@include common-auth** и меняем **sufficient** на **required**.*

Есть вопрос?

Наконец, редактируем конфигурационный файл sshd. Открываем его:

```
vi /etc/ssh/sshd_config
```

Выставляем следующие значения для опций:

```
ChallengeResponseAuthentication yes
...
UsePAM yes
```

** где **ChallengeResponseAuthentication** разрешает авторизацию с применением интерактивного ввода пароля; **UsePAM** разрешает использование модуля pam.*

Перезапустим сервис ssh:

```
systemctl restart sshd
```

Выполняем проверку нашей настройки командой:

```
vault-ssh-helper -verify-only -config
/etc/vault-ssh-helper/config.hcl
```

Мы должны получить ответ:

```
2021/05/18 14:07:14 [INFO] using SSH
mount point: ssh
2021/05/18 14:07:14 [INFO] using
namespace:
2021/05/18 14:07:14 [INFO] vault-ssh-
helper verification successful!
```

Значит настройка на клиенте выполнена коррекнто.

Создаем учетную запись на клиенте, под которой будем входить в систему с одноразовым паролем:

```
useradd test -m
```

** в нашей политике будет создаваться пароль для учетной записи **test**.*

Создаем одноразовый пароль

Вводим команду:

```
vault write ssh/creds/otp_key_role
ip=192.168.0.25
```

** в данном примере мы создаем одноразовый пароль для компьютера с IP-адресом 192.168.0.25.*

Мы должны получить, примерено, такой вывод:

```
Key                Value
---                -----
lease_id
ssh/creds/otp_key_role/5SYfW5VDZ3qGnaM
tPhUEHbNr
lease_duration     768h
lease_renewable    false
ip                 192.168.0.25
key                83a57021-74b0-3ce3-
8179-6fb92288c0ce
key_type           otp
port               22
username           test
```

** мы получили одноразовый пароль **83a57021-74b0-3ce3-8179-6fb92288c0ce**.*

Пробуем войти в систему:

```
ssh test@192.168.0.25
```

Вводим тот пароль, который нам выдала система (key). Войти мы сможем только один раз, после нужно будет уже генерировать новый пароль.

Настройка SSL

В инструкции выше мы настроили наше окружение для локального подключения по http. Данный метод временный, так как не позволит управлять системой с другого компьютера или придется жертвовать безопасностью. Рассмотрим процесс настройки запросов по защищенному протоколу https.

Первое, что нам нужно, это получить сертификат. Правильнее всего купить сертификат или запросить через Let's Encrypt. После прописать в конфигурационном файле vault путь до данного сертификата. Но мы рассмотрим процесс получения самоподписанного сертификата, но при этом, который примет система. Для этого необходимо выполнить несколько условий:

- Сертификат должен быть выдан для hostname, по которому мы будем отправлять запросы.
- Для сертификата мы должны указать альтернативное имя subjectAltName.

И так, создаем каталог для хранения сертификатов:

```
mkdir /etc/ssl/vault
```

Проверяем версию openssl:

```
openssl version
```

Она должна быть 1.1.1 и выше. В противном случае, необходимо выполнить обновление OpenSSL. Как правило, данное действие требуется только на CentOS 7.

Генерируем сертификат:

```
openssl req -new -x509 -days 1461 -
nodes -out /etc/ssl/vault/cert.pem -
keyout /etc/ssl/vault/cert.key -subj
"/C=RU/ST=SPb/L=SPb/O=Global
Security/OU=IT
Department/CN=vault.dmosk.local" -
```

```
addext "subjectAltName =
DNS:vault.dmosk.local"
```

** в данном примере мы сгенерируем необходимые ключи по пути **/etc/ssl/vault**; обязательно, нужно поменять значения **vault.dmosk.local** на имя сервера, который используется у вас.*

Если команда вернет ошибку, проверяем, что у нас обновленная версия openssl, которая поддерживает ключ addext.

Теперь откроем конфигурационный файл hashicorp vault:

```
vi /etc/vault.d/vault.hcl
```

Приведем секцию **HTTPS listener** к виду:

```
# HTTPS listener
listener "tcp" {
  address       = "0.0.0.0:8200"
  tls_cert_file =
"/etc/ssl/vault/cert.pem"
  tls_key_file  =
"/etc/ssl/vault/cert.key"
}
```

** необходимо поменять пути до файлов **tls_cert_file** и **tls_key_file**.*

Перезапускаем сервис:

> После перезагрузки сервиса, он станет запечатанным и нам нужно будет снова ввести 3 части ключа (команды **vault operator unseal**).

```
systemctl restart vault
```


Меняем в окружении переменную VAULT_ADDR:

```
export
VAULT_ADDR=https://vault.dmosk.local:8
200
```

** мы указываем протокол **https**, обращения должны выполняться по доменному имени, для которого мы получили сертификат; также указываем порт **8200**.*

Выполняем команду:

```
vault status
```

Мы должны получить состояние системы. Значит запросы по https работают.

И последнее, снова открываем файл:

```
vi /etc/environment
```

И также меняем значение для переменной VAULT_ADDR:

```
VAULT_ADDR=https://vault.dmosk.local:8
200
```

Запуск в виде контейнера Docker

В инструкции мы рассмотрели установку Hashicorp Vault как пакета. Также мы можем установить данный сервис в виде контейнера Docker из официального образа. Рассмотрим вкратце данный вопрос.

Для начала, необходимо установить в систему Docker. После загружаем образ

```
docker pull vault
```


Для запуска vault в режиме сервера вводим:

```
docker run --cap-add=IPC_LOCK --name
vault -d -p 8200:8200 -e
'VAULT_LOCAL_CONFIG={"backend":
{"file": {"path": "/vault/file"}},
"default_lease_ttl": "168h",
"max_lease_ttl": "720h", "listener":
{"tcp": {"address": "127.0.0.1:8200",
"tls_disable": "true"}}}' vault server
```

После заходим внутрь контейнера:

```
docker exec -it vault sh
```

Задаем системную переменную для подключения к vault по http:

```
export
VAULT_ADDR=http://127.0.0.1:8200
```

Инициализируем сервер:

```
vault operator init
```

... и так далее.

Установка клиента и удаленное подключение

Мы можем управлять нашим сервером с удаленного компьютера из командной строки. Для этого необходимо установить клиента и настроить системное окружение.

Установка клиента ничем не отличается от установки сервера. Только нам не нужно инициализировать систему и запускать ее в качестве сервиса. Операции отличаются в зависимости от установленной операционной системы.

а) для Debian / Ubuntu / Mint:

```
apt-get install wget apt-transport-
https

echo "deb [arch=amd64]
https://apt.releases.hashicorp.com
$(lsb_release -cs) main" >
/etc/apt/sources.list.d/hashicorp.list

apt-get update

apt-get install vault
```

б) для CentOS / Red Hat / Fedora:

```
yum install wget

wget
https://rpm.releases.hashicorp.com/RHE
L/hashicorp.repo -O
/etc/yum.repos.d/hashicorp.repo

yum install vault
```

После чего задаем переменную системного окружения

```
export
VAULT_ADDR=https://vault.dmosk.local:8
200
```

** обратите внимание, что мы подключаемся уже по **https**. Имя нашего сервера должно разрешаться в DNS или быть прописано в файле hosts на компьютере управления.*

Также, если у нас самоподписанный сертификат, вводим:

```
export VAULT_SKIP_VERIFY=true
```

** данная системная переменная указывает системе не проверять валидность сертификата.*

Пробуем посмотреть статус удаленного сервера:

```
vault status
```

Регистрируемся в системе:

```
vault login -method=userpass
username=dmosk
```

** в данном примере мы заходим под пользователем **dmosk**. Также мы можем войти с использованием токена.*

Можно выполнять действия согласно разрешениям.

Серверы # Безопасность # DevOps

Была ли полезна вам эта инструкция?

Да Нет

Дмитрий Моск — IT-специалист.
Настройка серверов, компьютерная помощь.

Инструкции

Как настроить почту для корпоративной среды на Debian

Установка и запуск менеджера управления проектами Taiga на Rocky Linux

Как собрать свой собственный deb-пакетов с нуля под Linux Debian

Примеры создания пакетов RPM из исходников или со своими файлами

Как установить и использовать сервер хранения секретов Hashicorp Vault

Установка и настройка файлового сервера Samba на Ubuntu

Установка и настройка кластера Kubernetes на Linux Ubuntu

Как настроить почту для корпоративной среды на Ubuntu Server

Установка, настройка и использование системы по сбору логов Grafana Loki на Linux

Как установить и настроить связку Asterisk + FreePBX на CentOS 8

Весь список ...

Нужна инструкция? Пишите:

Что хотите узнать...

Контактная эл. почта

Запросить инструкцию

Есть вопрос?

Настройка серверов

Есть вопрос?