?????????? ??????????

- Методы аутентификации в HashiCorp Vault

?????? ?????????????? ? HashiCorp Vault

Методы аутентификации: Данные методы представляют собой компоненты в системе Vault, которые осуществляют аутентификацию и отвечают за назначение пользователю соответствующих прав и набора политик. Чаще всего Vault делегирует администрирование и принятие решений по аутентификации какому-либо настроенному внешнему методу аутентификации (например, AWS, GCP, GitHub, Kubernetes и т.д.). Наличие нескольких методов аутентификации позволяет вам использовать тот метод аутентификации, который наиболее подходит для вашего варианта использования Vault в вашей организации. Таким образом, каждый метод аутентификации имеет определенный вариант использования. Например, разработчикам проще всего использовать метод аутентификации GitHub, но для конечных серверов рекомендуется использовать метод AppRole. Прежде чем клиент сможет взаимодействовать с Vault в полной мере, он должен пройти аутентификацию с использованием одного из существующих методов аутентификации. При аутентификации генерируется специальный токен. Этот токен, концептуально, аналогичен идентификатору сеанса в веб-сайте. К токену может быть прикреплена политика, которая отображается во время аутентификации. Перейдем к более подробному ознакомлению с некоторыми методами аутентификации в Vault.

Аутентификация с помощью Token: Однако прежде, чем мы приступим, стоит немного пройтись по тому, как вообще устроены токены в Vault и какую функцию они выполняют. Итак, токен – это основной метод аутентификации в Vault. Токены можно использовать как напрямую, так и посредством использования методов аутентификации для динамического создания токенов на основе внешних прав. В первой части, когда мы устанавливали и запускали Vault, вы, вероятно, могли заметить, что сервер выводит изначальный корневой токен с бесконечным временем жизни. Это первый метод аутентификации для Vault. Это также единственный метод аутентификации, который нельзя отключить. Как указано в концепции об аутентификации, все внешние механизмы аутентификации, такие как GitHub, сопоставляются с динамически создаваемыми токенами. Эти токены имеют все те же свойства, что и обычные токены, созданные вручную. В Vault токены соответствуют информации. Наиболее важная информация, сопоставленная с токеном, представляет собой набор из одной или нескольких прикрепленных политик. Эти политики контролируют, что разрешено или запрещено делать держателю токена в Vault. Другая сопоставленная информация включает метаданные, которые можно просмотреть и добавить в журнал аудита, как, например, время создания, время последнего обновления и т. д.:

903f7f5e70ed3d090a606.png

В свою очередь корневой токен – это токен, к которому прикреплена корневая политика. Корневые токены обладают полными привилегиями и могут делать что угодно в Vault. Кроме того, это единственный тип токенов в Vault, срок действия которого может быть

неограниченным без необходимости продления. Создание корневого токена является несколько затруднительным процессом. На самом деле существует только три способа создания корневого токена:

> Исходный корневой токен, созданный во время инициализации оператора хранилища. Такой токен не имеет срока действия.

> Использование другого корневого токена. Корневой токен с истекающим сроком действия не может создать корневой токен, срок действия которого никогда не истечет.

> С помощью vault operator generate-root с разрешением кворума держателей Unseal Key.

Корневые токены полезны при разработке, но в производственной среде их следует тщательно охранять. Фактически, Vault рекомендует использовать корневые токены только для достаточной первоначальной настройки (обычно для настройки методов и политик аутентификации, необходимых для того, чтобы администраторы могли получать более ограниченные токены) или в чрезвычайных ситуациях, после решения которых такой токен должен быть сразу же отозван. Если необходим новый корневой токен, для его создания на лету можно использовать команду vault operator generate-root, упомянутая ранее, и связанную с ней конечную точку API. Вообще, начиная с Vault версии 1.0, существует два типа токенов: сервисные токены и пакетные токены. Рассмотрим каждый из типов токенов более подробно:

> Service Tokens: Сервисные токены представляют собой тот тип токена, который пользователи обычно называют «обычными» токенами Vault. Они поддерживают все функции, такие как продление, отзыв, создание дочерних токенов и многое другое.

> Batch Tokens: Пакетные токены представляют собой зашифрованные BLOB-объекты, содержащие достаточно информации, которую можно использовать для действий в Vault, но для их отслеживания не требуется место на диске. В результате чего они являются чрезвычайно легкими и масштабируемыми, но лишены большинства функций сервисных токенов.

fea000ff61da83aa374a1.png

15b3d1a2380f0872c8654.png

Обычно, когда держатель токена создает новый токен, такой токен становится дочерним токеном. В свою очередь, если этот дочерний токен породит ещё один токен, то уже тот

токен станет дочерним токеном для только что созданного токена. Когда родительский токен отзывают, все его дочерние токены также аннулируются. Это гарантирует, что пользователь не сможет избежать отзыва токена, просто создав бесконечное дерево дочерних токенов. Часто такое поведение нежелательно, поэтому пользователи с соответствующим доступом могут создавать бесхозные токены. У таких токенов нет родителя – они являются корнем для собственного дерева токенов. Эти бесхозные токены могут быть созданы одним из следующих способов:

> Через доступ на запись к конечной точке auth/token/create-orphan.

> Имея sudo или root-доступ к auth/token/create и устанавливая для параметра no_parent значение true.

> Через роли хранилища токенов.

> Войдя в систему с помощью любого другого метода аутентификации.

Примечание: Пользователи с соответствующими правами также могут использовать конечную точку auth/token/revoke-orphan, которая отменяет данный токен, однако при этом все дочерние токены продолжат свою работу, став бесхозными. Данный метод стоит использовать с крайней осторожностью.

Итак, как вам уже известно, метод аутентификации посредством токена встроен в Vault и автоматически доступен по пути /auth/token. Если выполнить вот такую команду:

> vault auth list

```
Path        Type       Accessor                Description
----        ----       --------                -----------
token/      token      auth_token_ed139e53     token based credentials
```

То мы получим список всех активных методов аутентификаций в Vault. В данном случае речь идет только о методе аутентификации через токен. Когда любой другой метод аутентификации возвращает ответ, ядро Vault вызывает метод Token, чтобы создать новый уникальный токен для этого метода аутентификации. Хранилище токенов также можно использовать для обхода любого другого метода аутентификации: вы можете создавать токены напрямую, а также выполнять с токенами множество других операций, таких как обновление, их отзыв и т.д. На текущий момент я прошел аутентификацию в Vault, поскольку взаимодействовал с ним на основе предыдущих статей, но давайте сделаем это ещё раз. Выполним данную команду и вставим корневой токен, который был получен при установке Vault:

> vault login

```
Token (will be hidden):
WARNING! The VAULT_TOKEN environment variable is set! This takes precedence over the value set b

Success! You are now authenticated. The token information displayed below is already stored in t

Key                  Value
---                  -----
token                hvs.GtiNt0B0AOrgMH9zdTMrGwA8
token_accessor       Rl0NmTmxXBfRS2wANmG8ZuJg
token_duration       ∞
token_renewable      false
token_policies       ["root"]
identity_policies    []
policies             ["root"]
```

В моём случае показано предупреждение, поскольку до этого я добавил токен в экспорт переменной VAULT_TOKEN. По-хорошему, мы либо проходим этап аутентификации с помощью входа, либо используем значение переменной, но сейчас это не так важно. Далее нас оповещают о том, что мы успешно вошли, но самое интересное расположено ниже. Нам выводят базовую информацию о токене, и в данном случае мы видим, что время жизни нашего токена бесконечно, а политика, которая привязано к этому токену, является корневой. Это значит, что с этим токеном у нас в распоряжении полные привилегии на взаимодействие с Vault. К слову, полный список команд в контексте токена выглядит следующим образом:

```
Usage: vault token <subcommand> [options] [args]

This command groups subcommands for interacting with tokens. Users can create, lookup, renew, ar

Please see the individual subcommand help for detailed usage information.

Subcommands:
   capabilities   Print capabilities of a token on a path
   create         Create a new token
   lookup         Display information about a token
   renew          Renew a token lease
   revoke         Revoke a token and its children
```

Теперь давайте создадим дочерний токен на основе корневого. Сделать это можно с помощью следующей команды:

> vault token create

```
Key                  Value
---                  -----
```

```
token               hvs.i2rhRxBVGS14bUTM0i2fwg8C
token_accessor      5tLgv5pGJKVpx1bZA6hnaTQc
token_duration      ∞
token_renewable     false
token_policies      ["root"]
identity_policies   []
policies            ["root"]
```

Как вы могли заметить, новосозданный токен также обладает корневой политикой, а его время никогда не истечет. Так происходит потому, что данный токен является дочерним по отношению к корневому токену и по умолчанию наследует все политики от своего родительского токена. Такой вариант нам не очень подходит, поскольку ранее я уже упоминал о том, что корневые токены крайне опасны и должны находиться в распоряжении у ограниченного числа людей. Так что давайте данный токен отзовем с помощью следующей команды:

> vault token revoke hvs.i2rhRxBVGS14bUTM0i2fwg8C

```
Success! Revoked token (if it existed)
```

И несколько видоизменим команду для его повторного создания:

> vault token create -policy=default

```
Key                 Value
---                 -----
token               hvs.CAESIPZuIZ6i3om5kzu0jY8kRyIw2de4A7K6uMG4yliZpknbGh4KHGh2cy5QN2wxcDN4VmJy
token_accessor      vP6R3UBCQl2PRv4v01OHPIE8
token_duration      768h
token_renewable     true
token_policies      ["default"]
identity_policies   []
policies            ["default"]
```

Длина нашего токена была изменена, поскольку к нему была добавлена политика Default. Сейчас это не столь важно и в будущем мы отдельно затронем тему политик в Vault. Пока что стоит знать, что эта политика доступа в Vault по умолчанию. Она не предоставляет полные права, но кое-что всё же можно будет сделать. Также стоит обратить внимание на то, что время жизни токена изменилось. Теперь оно составляет 768 часов, что равно примерно одному месяцу. Если мы хотим явным образом указать желаемое время жизни для того или иного токена, необходимо выполнить следующую команду:

> vault token create -policy=default -period=100h

```
Key                Value
---                -----
token              hvs.CAESIO3-prlzLywGxXMYzX-FjUgkVucLwPu5M7o5Dnl9jDuxGh4KHGh2cy5hcVNzdjJ5UHdT
token_accessor     m0nohVowhqNjvhFozJiBI4Fh
token_duration     100h
token_renewable    true
token_policies     ["default"]
identity_policies  []
policies           ["default"]
```

Однако, если мы попытаемся превысить данное значение, то получим предупреждение о том, что это невозможно, поскольку max_ttl равен 768 часам, что является значением по умолчанию:

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

 * period of "1000h" exceeded the effective max_ttl of "768h"; period value is capped accordingl
```

Для того чтобы изменить данное состояние, необходимо открыть конфигурационный файл Vault и добавить следующие два параметра:

```
max_lease_ttl = "10000h"
default_lease_ttl = "72h"
```

В общем случае конфигурационный файл будет выглядеть следующим образом:

```
ui = true

max_lease_ttl = "10000h"
default_lease_ttl = "72h"

storage "file" {
 path  = "/etc/vault/data"
}

listener "tcp" {
 address = "0.0.0.0:443"
 tls_cert_file = "/etc/letsencrypt/live/vault.kitezh.online/fullchain.pem"
 tls_key_file = "/etc/letsencrypt/live/vault.kitezh.online/privkey.pem"
 tls_disable = "false"
}
```

Далее потребуется перезапуск Vault с помощью systemctl, а также его распаковка с помощью ключей Unseal. Параметр default_lease_ttl говорит о том, что теперь срок жизни каждого не корневого токена, который будет создан, равен 72 часам по умолчанию, а максимальное время жизни не должно превышать значение, которое будет выше 10000 часов. Исходя из этого попробуем создать новый токен с временем жизни в 2000 часов:

> vault token create -policy=default -period=2000h

```
Key                 Value
---                 -----
token               hvs.CAESIE5bHFZ75fx5J9osMf-fH68PyhCrNaNRQTqQKekPqbuzGh4KHGh2cy5taThjblpJZFJi
token_accessor      e5It0ZzDDr1AG6alhi8RhAaW
token_duration      2000h
token_renewable     true
token_policies      ["default"]
identity_policies   []
policies            ["default"]
```

Вообще, говоря о сроках жизни, каждый не корневой токен связан со значением TTL, который представляет собой текущий период действия с момента создания токена или времени последнего обновления, в зависимости от того, что наступит позже. После того, как значение TTL истекает, токен перестает функционировать, как и все его дочерние токены, если только они не были созданы с помощью ключа -orphan (об этом ниже). Если токен является возобновляемым, можно продлить срок действия токена, используя обновление токена или соответствующую конечную точку для продления. Когда для токена не был установлен ни период, ни явное максимальное значение TTL, время жизни токена с момента его создания будет сравниваться со значением TTL по умолчанию. В основе времени жизни токена лежит совокупность факторов, некоторые из которых были упомянуты ранее:

> Максимальный срок жизни токена по умолчанию составляет 32 дня, но его можно изменить в файле конфигурации Vault.

> Максимальный TTL, установленный с помощью конфигурации. Это значение может переопределять максимальное значение TTL системы – оно может быть длиннее или короче, и если оно было установлено, это значение будет учитываться.

> Значение, предложенное методом аутентификации, выдавшим токен. Это можно настроить для каждой роли, группы или пользователя. Такое значение может быть меньше максимального значения TTL, заданного в настройках (или, если оно не установлено, максимального значения TTL системы), но не может быть больше.

К слову, есть и похожий на period параметр. Команда будет выглядеть так:

> vault token create -policy=default -ttl=2000h

```
Key                 Value
---                 -----
token               hvs.CAESIK_gjn69bbG3dOFzZLkqMRMbiyZVqP81sgBc0DGLuNHZGh4KHGh2cy5FOU9xUlBnSmlE
token_accessor      W6pcAIWyS9rcvUNnhqFLRIxo
```

```
token_duration      2000h
token_renewable     true
token_policies      ["default"]
identity_policies   []
policies            ["default"]
```

Не вникая в детали результат кажется одни и тем же, но суть в том, что пользователь с правами root имеет возможность создавать периодические токены. На момент выпуска TTL периодического токена будет равен настроенному периоду. При каждом продлении TTL будет сбрасываться обратно на этот настроенный период, и пока токен успешно продлевается в течение каждого из этих периодов времени, срок его действия никогда не истечет. За исключением корневых токенов, в настоящее время это единственный способ иметь неограниченный срок действия токена в Vault. Это полезно для долго работающих служб, которые не могут обрабатывать повторное создание токена. Итак, вот мы создали токен с временем жизни в 2000 часов. Попробуем зайти, указав его:

> vault login

```
Success! You are now authenticated. The token information displayed below is already stored in t

Key                  Value
---                  -----
token                hvs.CAESIE5bHFZ75fx5J9osMf-fH68PyhCrNaNRQTqQKekPqbuzGh4KHGh2cy5taThjblpJZFJi
token_accessor       e5It0ZzDDr1AG6alhi8RhAaW
token_duration       1995h26m29s
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

Как вы могли заметить, мы успешно вошли в Vault с помощью данного токена, только теперь к нему относится политика по умолчанию, а его общее время жизни составляет чуть меньше, чем 2000 часов. Далее, если попытаться вывести список всех текущих секретов, мы получим сообщением об ошибке, поскольку у нас нет на это прав (так как в политике Default не прописаны такие опции, а мы взяли данную политику за основу):

```
Error listing secrets engines: Error making API request.

URL: GET https://vault.kitezh.online:443/v1/sys/mounts
Code: 403. Errors:

* 1 error occurred:
       * permission denied
```

Зато в политику Default прописано взаимодействие с cubbyhole (подсистема секретов KV по умолчанию), что позволяет нам создать новый секрет, что мы, собственно, и сделаем:

> vault kv put cubbyhole/test host=localhost
> vault kv get cubbyhole/test

```
==== Data ====
Key    Value
---    -----
host   localhost
```

Таким образом, создав новый токен, мы явно ограничили его в правах, что повысит общую безопасность системы, и при этом владелец токена сможет взаимодействовать с той подсистемой секретов и теми данными от секретов, которые ему необходимы. И давайте заодно рассмотрим бесхозные токены, которые упоминались ранее. Предположим, что мы хотим создать токен от текущего, но при этом чтобы он потерял свою наследственность в иерархии. В таком случае нам понадобится следующая команда:

> vault token create -policy=default -period=1000h -orphan

```
Key                  Value
---                  -----
token                hvs.CAESIA-Gau19WwmUX9y8KMxWEqgRk4bfe_KcU6AX7vmJsV9MGh4KHGh2cy5CdWQxaTJ2SFVE
token_accessor       i7gngmkBlVuWEI4P7sxuYgcD
token_duration       1000h
token_renewable      true
token_policies       ["default"]
identity_policies    []
policies             ["default"]
```

На самом деле, если вы вошли не из под корневого токена, то сейчас у вас ничего не выйдет с выполнением этой командой, поскольку политике Default не хватает соответствующих прав на создание токена, что правильно и в целом данный момент сильно переплетается с ними (политиками), что выходит за рамки текущего повествования, однако в следующей статье речь пойдет как раз о них и там этот пример будет фигурировать в более развернутой форме. Пока что предлагаю войти в Vault из под корневого токена и выполнить команду, представленную выше, для наглядной демонстрации возможностей бесхозного токена. После чего можно выполнить ещё команду, которая выдаст более подробную информацию уже о самом токене:

> vault token lookup hvs.CAESIA-Gau19WwmUX9y8KMxWEqgRk4bfe_KcU6AX7vmJsV9MGh4KHGh2cy5CdWQxaTJ2SFVBazNpZTNYUmlwYWpKMXo

```
Key                  Value
---                  -----
accessor             i7gngmkBlVuWEI4P7sxuYgcD
creation_time        1694103352
```

```
creation_ttl       1000h
display_name       token
entity_id          n/a
expire_time        2023-10-19T08:15:52.393554517Z
explicit_max_ttl   0s
id                 hvs.CAESIA-Gau19WwmUX9y8KMxWEqgRk4bfe_KcU6AX7vmJsV9MGh4KHGh2cy5CdWQxaTJ2SFVBa
issue_time         2023-09-07T16:15:52.393560728Z
meta               <nil>
num_uses           0
orphan             true
path               auth/token/create
period             1000h
policies           [default]
renewable          true
ttl                999h58m27s
type               service
```

Среди прочих атрибутов можно найти тот же orphan, который равен значению true, что говорит о том, что данный токен является бесхозным. Здесь также указано время жизни токена, его тип, дата создания, идентификатор и т.д. Иногда очень полезно получить такую информацию. Теперь давайте вернемся к токену, который был создан с временем жизни в 2000 часов. Сейчас его время жизни сократилось, о чем свидетельствует вывод ниже:

> vault token lookup hvs.CAESIE5bHFZ75fx5J9osMf-fH68PyhCrNaNRQTqQKekPqbuzGh4KHGh2cy5taThjblpJZFJiZzBMczV4V3ZRWExjMWc

```
Key                Value
---                -----
accessor           e5It0ZzDDr1AG6alhi8RhAaW
creation_time      1694085020
creation_ttl       2000h
display_name       token
entity_id          n/a
expire_time        2023-11-29T19:10:20.139700548Z
explicit_max_ttl   0s
id                 hvs.CAESIE5bHFZ75fx5J9osMf-fH68PyhCrNaNRQTqQKekPqbuzGh4KHGh2cy5taThjblpJZFJiZ
issue_time         2023-09-07T11:10:20.139707874Z
meta               <nil>
num_uses           0
orphan             false
path               auth/token/create
period             2000h
policies           [default]
renewable          true
ttl                1994h45m32s
type               service
```

Но мы можем обновить это время жизни, выполнив следующую команду:

> vault token renew hvs.CAESIE5bHFZ75fx5J9osMf-fH68PyhCrNaNRQTqQKekPqbuzGh4KHGh2cy5taThjblpJZFJiZzBMczV4V3ZRWExjMWc

```
Key                 Value
---                 -----
token               hvs.CAESIE5bHFZ75fx5J9osMf-fH68PyhCrNaNRQTqQKekPqbuzGh4KHGh2cy5taThjblpJZFJi
token_accessor      e5It0ZzDDr1AG6alhi8RhAaW
token_duration      2000h
token_renewable     true
token_policies      ["default"]
identity_policies   []
policies            ["default"]
```

И ещё один важный момент по поводу токенов. При создании токенов также создается и возвращается средство доступа к токену. Этот метод доступа представляет собой значение, которое действует как ссылка на токен и может использоваться только для выполнения ограниченного ряда действий, среди которых обновление токена, его отзыв, а также просмотр доступности токена. Существует множество полезных рабочих процессов, связанных с методами доступа к токенам. Например, служба, которая создает токены от имени другой службы (планировщик Nomad), может хранить метод доступа, связанный с определенным идентификатором задания. Когда задание завершается, метод доступа можно использовать для мгновенного отзыва токена, предоставленного заданию, и всех арендованных учетных данных, что ограничивает вероятность того, что потенциальный злоумышленник обнаружит и использует их. И, наконец, единственный способ составить список токенов – это использовать следующую команду:

> vault list auth/token/accessors

```
Keys
----
1tI7ndiGjFcx9Mz42vZqfeNM
3utxC8j0hJIz66PKBftLul21
9SWBDgdgwdHzPgjfxp6OaXqA
9muBUCGWEaDlwMqGWlvbcTxp
CpCigFH5vMJnvRMw877btufs
CtaAsJYsq4pRzYOQ5hJ9fliL
EKuI5cF0VgQ0NDexNErrzcuu
EYcE79Z5vWSuOAbZJrPWKeSN
Fm8YdWEiRttuaDIGWHQmaXVy
GlSFG0ribgBl8DBpVKuvGpQ7
JtXlDuoBdCYASeoOmeCXD4Sp
N74X1yCHWslseEYBFhP7Q7ly
Rl0NmTmxXBfRS2wANmG8ZuJg
W6pcAIWyS9rcvUNnhqFLRIxo
ZCCpODQxQVeMonNh7ICumX1u
bJkn55AAZ6CYbCLEe57DRYge
```

У меня этих акссесоров накопилось много, потому что я каждый раз создавал новый токен, демонстрируя тот или иной пример, и при этом не отзывал их, поскольку это мой тестовый стенд. К слову, есть и более ухищренная команда, которая в формате JSON выведет все акссесоры, относящиеся к корневому токену, например:

> vault list -format json auth/token/accessors | jq -r .[] | xargs -I '{}' vault token lookup -format json -accessor '{}' | jq -r 'select(.data.policies | any(. == "root"))'

На этом рассмотрение как аутентификации с помощью токенов, так и работу с ними в целом можно завершить. Далее перейдем к ознакомлению с ещё одним методом аутентификации в Vault.

Аутентификация с помощью метода Userpass: Метод аутентификации Userpass позволяет пользователям проходить аутентификацию в Vault, используя для этого комбинацию из имени пользователя и пароля. Комбинации, состоящие из имени пользователя и пароля настраиваются непосредственно для метода аутентификации с использованием userpass/. Данный метод не может считывать имена пользователей и пароли из внешнего источника. Этот метод также преобразует все отправленные имена пользователей в нижний регистр. Например, Opengrad и opengrad – это одна и та же запись. Метод аутентификации должен быть настроен заранее, прежде чем пользователь или сервис сможет пройти аутентификацию. Чтобы начать работу с методом Userpass, его необходимо включить. Сделать это можно с помощью следующей команды:

> vault auth enable userpass

```
Success! Enabled userpass auth method at: userpass/
```

Далее необходимо создать пользователя, который воспользуется данным методом аутентификации. Сделать это можно с помощью данной команды:

> vault write auth/userpass/users/admin password=test policies=default

```
Success! Data written to: auth/userpass/users/admin
```

Теперь мы можем попробовать войти в Vault с помощью представленного метода аутентификации. Сделать это можно, используя следующую команду:

> vault login -method userpass username=admin

```
Password (will be hidden):
Success! You are now authenticated. The token information displayed below is already stored in t

Key                    Value
```

```
---                   -----
token                 hvs.CAESIGlMqHkNT7VfdRFOpZpIjPtYDu7w61975AK4_p9QIwZaGh4KHGh2cy5CRTdMNnZWUD
token_accessor        PJEetJzUCpeZuffrD3aS46IR
token_duration        24h
token_renewable       true
token_policies        ["default"]
identity_policies     []
policies              ["default"]
token_meta_username   admin
```

Таким же образом мы можем войти и в веб-панель Vault:

65f170466cf7039134b38.png

К слову, необязательно использовать название Userpass при активации метода аутентификации. Можно указать собственное имя, задав его в виде пути. Выглядеть это будет вот так:

> vault auth enable -path=opengrad userpass

`Success! Enabled userpass auth method at: opengrad/`

> vault auth list

```
Path        Type       Accessor                Description
----        ----       --------                -----------
opengrad/   userpass   auth_userpass_ad238c9f   n/a
token/      token      auth_token_ed139e53      token based credentials
userpass/   userpass   auth_userpass_033fce13   n/a
```

Создадим ещё одного пользователя уже в подсистеме opengrad/:

> vault write auth/opengrad/users/opengrad-user password=test policies=default

`Success! Data written to: auth/opengrad/users/opengrad-user`

Только теперь для того, чтобы войти в Vault нужно немного видоизменить предыдущую команду:

> vault login -method userpass -path=opengrad username=opengrad-user

```
Password (will be hidden):
Success! You are now authenticated. The token information displayed below is already stored in t

Key                  Value
```

```
---                   -----
token                 hvs.CAESIA32R9pXp0BXn206ASG8sk0UGaConph3XoMoYwLAo2gIGh4KHGh2cy5JaVFPRnJWUT
token_accessor        4oyfplsj4RraBrPkqp4C40He
token_duration        24h
token_renewable       true
token_policies        ["default"]
identity_policies     []
policies              ["default"]
token_meta_username   opengrad-user
```

Это же касается и входа через веб-панель в Vault:

c7dd0b9a90d63c731dc54.png

Чтобы удалить того или иного пользователя, воспользуйтесь такой командой:

> vault delete auth/opengrad/users/opengrad-user

```
Success! Data deleted at: auth/opengrad/users/opengrad-user
```

Аутентификация с помощью метода AppRole: Метод аутентификации AppRole позволяет узлам или приложениям проходить аутентификацию с использованием ролей, определенных в Vault. Открытый дизайн AppRole позволяет использовать разнообразный набор процессов и конфигураций для обработки большого количества приложений. Такой метод аутентификации ориентирован на автоматизированные рабочие процессы (узлы и службы) и менее полезен для людей. AppRole представляет собой набор политик и ограничений для входа, которые необходимо выполнить для получения токена с соответствующими правами. Область применения может быть как узкой, так и широкой по желанию. AppRole может быть создана для конкретного узла или даже для конкретного пользователя на этом узле, или для службы, распределенной по узлам. Учетные данные, необходимые для успешного входа в систему, зависят от ограничений, установленных для AppRole. Путь по умолчанию – /approle. Если этот метод аутентификации был включен по другому пути, укажите вместо него auth/<your_path>/login. Чтобы начать работу с методом AppRole, его необходимо включить. Сделать это можно с помощью следующей команды:

> vault auth enable approle

```
Success! Enabled approle auth method at: approle/
```

Чтобы создать роль, используйте данную команду (параметры опциональны):

> vault write auth/approle/role/my-role secret_id_ttl=10m token_num_uses=10 token_ttl=20m token_max_ttl=30m secret_id_num_uses=40

Если для токена, выпущенного вашим приложением, требуется возможность создания дочерних токенов, вам необходимо установить для параметра token_num_uses значение 0. Чтобы узнать идентификатор созданной ранее роли, воспользуйтесь следующей командой:

> vault read auth/approle/role/my-role/role-id

```
Key       Value
---       -----
role_id   7a50fcd5-2f36-d80c-7c11-d81ab552f2a6
```

А чтобы получить идентификатор секрета для выданного идентификатора роли, используйте эту команду:

> vault write -f auth/approle/role/my-role/secret-id

```
Key                   Value
---                   -----
secret_id             f4d43c52-e30d-a73b-0554-42de1f3ea4a6
secret_id_accessor    faaf7dca-c029-7de3-179b-04215ecf03e0
secret_id_num_uses    40
secret_id_ttl         10m
```

Теперь рассмотрим такие понятия, как Secret ID и Role ID более подробно:

> Role ID: Это идентификатор, который выбирает AppRole, по которому оцениваются другие учетные данные. При использовании данного метода аутентификации Role ID всегда является обязательным аргументом. По умолчанию Role ID представляет собой уникальный UUID, который позволяет ему служить вторичным секретом для другой учетной информации. Однако ему могут быть присвоены определенные значения, чтобы соответствовать внутренней информации клиента (например, доменному имени клиента).

> Secret ID: Это учетные данные, которые по умолчанию требуются для любого входа в систему и такие данные всегда должны оставаться секретными. Secret ID можно создавать для AppRole либо путем генерации 128-битного чисто случайного UUID самой ролью (режим Pull), либо с помощью определенных настраиваемых значений (режим Push). Подобно токенам, Secret ID имеют такие свойства, как лимит использования, TTL и срок действия.

Если Secret ID, используемый для входа в систему, извлекается из AppRole, он работает в режиме запроса. Если клиент устанавливает пользовательский Secret ID для AppRole, это называется режимом Push. Режим Push имитирует поведение устаревшего метода аутентификации App ID, однако в большинстве случаев режим Pull является лучшим подходом. Причина в том, что режим Push требует, чтобы какая-то другая система имела информацию о полном наборе учетных данных клиента (Role ID и Secret ID) для создания записи, даже если затем они распространятся по разным путям. Однако в режиме Pull, даже несмотря на то, что Role ID должен быть известен клиенту для распространения, Secret ID может оставаться конфиденциальным для всех сторон, за исключением конечного клиента, проходящего проверку подлинности, с помощью переноса ответов. Режим Push доступен для обеспечения совместимости рабочего процесса App ID, что в некоторых конкретных случаях предпочтительнее, но в большинстве случаев режим Pull более безопасен, поэтому предпочтение отдают именно ему. Более подробно об этом можно узнать в документации Vault.

?????? ????????????? ???????????

- Методы аутентификации в HashiCorp Vault (Часть 3)
- AppRole auth method