Метод аутентификации. JWT/OIDC

Данный механизм может использовать внешние сертификаты X.509 в качестве части TLS или проверки подписи. Проверка подписей по сертификатам X.509, использующим SHA-1, устарела и по умолчанию отключена.

Метод **jwt auth** можно использовать для аутентификации в StarVault с помощью OIDC или путем предоставления JWT.

Метод OIDC позволяет выполнять аутентификацию через настроенный провайдер OIDC с помощью веб-браузера пользователя. Этот метод может быть инициирован из пользовательского интерфейса StarVault или командной строки. Кроме того, JWT может быть предоставлен напрямую. JWT криптографически проверяется с помощью локально предоставленных ключей, или, если настроено, для получения соответствующих ключей может использоваться служба OIDC Discovery. Выбор метода настраивается для каждой роли. Оба метода позволяют дополнительно обрабатывать данные утверждений в JWT. Сначала будут рассмотрены некоторые концепции, общие для обоих методов, а затем конкретные примеры использования OIDC и JWT.

Оба метода позволяют дополнительно обрабатывать данные утверждений в JWT. Сначала будут рассмотрены некоторые концепции, общие для обоих методов, а затем конкретные примеры использования OIDC и JWT.

1. Аутентификация OIDC

В данном разделе рассматривается настройка и использование ролей OIDC. Если необходимо напрямую предоставить JWT, обратитесь к разделу «Аутентификация JWT» ниже. Предполагается базовое знакомство с концепцией OIDC. Поток кода авторизации использует расширение Proof Key for Code Exchange (PKCE).

StarVault включает два встроенных потока авторизации OIDC: пользовательский интерфейс StarVault и CLI с использованием логина `starvault`.

1.1. Перенаправляющие URI

Важной частью конфигурации роли OIDC является правильная настройка URI перенаправления. Это должно быть сделано как в StarVault, так и в провайдере OIDC, и эти конфигурации должны совпадать. URI перенаправления задаются для роли с помощью параметра `allowed_redirect_uris`. Существуют разные URI перенаправления для

настройки потоков StarVault UI и CLI, поэтому в зависимости от установки необходимо настроить один или оба.

1.2. CLI

Если вы планируете поддерживать аутентификацию через `starvault login -method=oidc`, необходимо задать URI перенаправления на localhost. Обычно это может быть: `http://localhost:8250/oidc/callback`. При необходимости при входе в систему через CLI можно указать другой хост и/или порт прослушивания, и URI с этим хостом/портом должен совпадать с одним из настроенных перенаправляемых URI. Эти же URI «localhost» должны быть добавлены и в провайдер.

1.3. Пользовательский интерфейс StarVault

Для входа в систему через StarVault UI требуется URI перенаправления вида: `https://{host:port}/ui/vault/auth/{path}/oidc/callback`.

Параметры `host:port` должны соответствовать серверу StarVault, а `path` - пути, по которому подключен бэкенд JWT (например, `oidc` или `jwt`).

Если для параметра `oidc_response_mode` установлено значение `form_post`, то для входа в систему через пользовательский интерфейс StarVault требуется URI перенаправления вида: `https://{host:port}/v1/auth/{path}/oidc/callback`.

2. Вход в систему OIDC

Выполнить вход в систему возможно двумя способами:

▸ Через пользовательский интрефейс StarVault

▸ Через CLI

3. Конфигурация провайдера OIDC

Поток аутентификации OIDC был успешно протестирован с рядом провайдеров. Полное руководство по настройке приложений OAuth/OIDC выходит за рамки документации StarVault.

4. Устранение неполадок в конфигурации OIDC

Объем настроек, необходимых для OIDC, сравнительно невелик, но отладка причин неработоспособности может оказаться непростой задачей. Несколько советов по настройке OIDC:

- Если параметр роли (например, `bound_claims`) требует значения карты, его нельзя установить отдельно с помощью StarVault CLI. В таких случаях лучше всего записать всю конфигурацию в виде одного JSON-объекта:

```JSON
starvault write auth/oidc/role/demo -<<EOF
{
  "user_claim": "sub",
  "bound_audiences": "abc123",
  "role_type": "oidc",
  "policies": "demo",
  "ttl": "1h",
  "bound_claims": { "groups": ["mygroup/mysubgroup"] }
}
EOF
```

- Следите за выходом журнала StarVault. Будет выводиться важная информация о сбоях проверки OIDC.
- Убедитесь, что URI перенаправления корректны в StarVault и на провайдере. Они должны точно совпадать. Проверьте: `http/https`, 1`27.0.0.1/localhost`, номера портов, наличие слэшей в конце.
- Начните с простого. Единственная конфигурация утверждения, которая требуется роли, - это `user_claim`. После того как аутентификация будет работать, вы можете добавить дополнительные привязки утверждений и копирование метаданных.
- `bound_audiences` является необязательным для ролей OIDC и обычно не требуется. Провайдеры OIDC будут использовать `client_id` в качестве аудитории, и валидация OIDC ожидает этого.
- Уточните у своего провайдера, какие диапазоны необходимы для получения всей необходимой информации. Часто требуется запрашивать диапазоны **профиль** и **группы**, которые можно добавить, установив для роли параметр `oidc_scopes=«profile,groups`.
- Если вы видите в журналах ошибки, связанные с претензиями, внимательно изучите документацию поставщика, чтобы понять, как он называет и структурирует свои претензии. В зависимости от провайдера, вы можете сконструировать простой запрос `curl implicit grant` для получения JWT, который вы можете проверить. Пример декодирования JWT (в данном случае расположенного в поле `access_token` JSON-ответа):

- `cat jwt.json | jq -r .access_token | cut -d. -f2 | base64 -D`
- Также доступна ролевая опция `verbose_oidc_logging`, которая будет записывать полученный токен OIDC в журналы сервера, если включено ведение журнала на уровне отладки. Это может быть полезно при отладке настройки провайдера и проверке того, что полученные претензии соответствуют вашим ожиданиям. Поскольку данные утверждений записываются дословно и могут содержать конфиденциальную информацию, эту опцию не следует использовать в производстве.

5. Аутентификация JWT

Процесс аутентификации для ролей типа `jwt` проще, чем в OIDC, поскольку StarVault нужно только подтвердить предоставленный JWT.

5.1. Проверка JWT

Подписи JWT будут проверяться по открытым ключам эмитента. Этот процесс может осуществляться тремя различными способами, хотя для одного бэкенда может быть настроен только один метод:

- **Статические ключи**. Набор открытых ключей хранится непосредственно в конфигурации бэкенда.
- **JWKS**. Настраивается URL-адрес JSON Web Key Set (JWKS) (и дополнительная цепочка сертификатов). Ключи будут извлекаться из этой конечной точки во время аутентификации.
- **IDC Discovery** (и необязательная цепочка сертификатов). Ключи будут извлекаться из этого URL во время аутентификации. При использовании OIDC Discovery будут применяться критерии проверки OIDC (например, iss, aud и т. д.).

Если необходимо использовать несколько методов, можно установить и настроить другой экземпляр бэкенда по другому пути одним из способов:

▸ Через CLI

▸ Через API

6. Конфигурация

Методы аутентификации должны быть настроены заранее, прежде чем пользователи или машины смогут пройти аутентификацию. Эти шаги обычно выполняются оператором или

инструментом управления конфигурацией.

1. Включите метод аутентификации JWT. Можно использовать имя `jwt` или `oidc`. Бэкэнд будет смонтирован по выбранному имени.

```bash
$ starvault auth enable jwt
```

или

```bash
$ starvault auth enable oidc
```

2. Используйте конечную точку `/config` для настройки StarVault. Для поддержки ролей JWT необходимо наличие либо локальных ключей, либо URL JWKS, либо URL OIDC Discovery. Для ролей OIDC требуется URL-адрес OIDC Discovery, идентификатор клиента OIDC и секрет клиента OIDC. Список доступных параметров конфигурации см. в документации по API.

```bash
$ starvault write auth/jwt/config \
    oidc_discovery_url="https://myco.auth0.com/" \
    oidc_client_id="m5i8bj3iofytj" \
    oidc_client_secret="f4ubv72nfiu23hnsj" \
    default_role="demo"
```

Если вам нужно выполнить проверку JWT с помощью валидации JWT-токена, оставьте `oidc_client_id` и `oidc_client_secret` пустыми.

```bash
$ starvault write auth/jwt/config \
   oidc_discovery_url="https://MYDOMAIN.eu.auth0.com/" \
   oidc_client_id="" \
   oidc_client_secret="" \
```

3. Создайте именованную роль:

```bash
starvault write auth/jwt/role/demo \
    allowed_redirect_uris="http://localhost:8250/oidc/callback" \
    bound_subject="r3qX9DljwFIWhsiqwFiu38209F10atW6@clients" \
    bound_audiences="https://starvault.plugin.auth.jwt.test" \
    user_claim="https://starvault/user" \
    groups_claim="https://starvault/groups" \
    policies=webapps \
    ttl=1h
```

Эта роль авторизует JWT с заданными утверждениями `subject` и `audience`, назначает политику webapps и использует заданные утверждения `user`/`groups` для настройки псевдонимов Identity.

Полный список параметров конфигурации приведен в документации по API.

6.1. Связанные требования

После того как JWT был подтвержден как правильно подписанный и не истекший, поток авторизации проверит соответствие всех настроенных «связанных» параметров. В некоторых случаях существуют специальные параметры, например `bound_subject`, которые должны совпадать с параметром `sub` в JWT. Роль также может быть настроена на проверку произвольных утверждений с помощью карты `bound_claims`. Карта содержит набор утверждений и их необходимые значения. Например, предположим, что для `bound_claims` установлено значение:

```json
{
  "division": "Europe",
  "department": "Engineering"
}
```

Только JWT, содержащие утверждения `division` и `department` и соответствующие значения `Europe` и `Engineering`, будут авторизованы. Если ожидаемое значение представляет собой список, утверждение должно соответствовать одному из элементов в списке. Чтобы ограничить авторизацию набором адресов электронной почты:

```json
{
  "email": ["fred@example.com", "julie@example.com"]
}
```

Связанные утверждения могут быть дополнительно сконфигурированы. Более подробную информацию см. в документации по API.

6.2. Утверждения как метаданные

Данные из утверждений можно скопировать в результирующие метаданные токена аутентификации и псевдонима, настроив `claim_mappings`. Этот параметр роли представляет собой карту элементов для копирования. Элементы карты имеют вид: `«<JWT claim>«:»<metadata key>»`. Предположим, что параметр `claim_mappings` имеет значение:

```json
{
    "division": "organization",
    "department": "department"
}
```

Это указывает, что значение утверждения JWT `department` должно быть скопировано в ключ метаданных `department`. Значение утверждения JWT `department` также будет скопировано в метаданные, но сохранит имя ключа. Если утверждение настроено в `claim_mappings`, оно должно существовать в JWT, иначе аутентификация не пройдет.

Имя ключа метаданных `role` зарезервировано и не может быть использовано для сопоставления утверждений.

6.3. Спецификации требований и указатель JSON

Некоторые параметры (например, `bound_claims`, `groups_claim`, `claim_mappings`, `user_claim`) используются для указания на данные внутри JWT. Если нужный ключ находится на верхнем уровне JWT, его имя может быть указано напрямую. Если он вложен на более низком уровне, можно использовать JSON-указатель.

Предположим, что необходимо сослаться на следующие данные JSON:

JSON

```
{
  "division": "North America",
  "groups": {
    "primary": "Engineering",
    "secondary": "Software"
  }
}
```

Параметр `division` будет ссылаться на `North America`, поскольку это ключ верхнего уровня. Параметр `/groups/primary` использует синтаксис JSON Pointer для ссылки на `Engineering` на более низком уровне. В качестве селектора можно использовать любой правильный указатель JSON. Полное описание синтаксиса см. в JSON Pointer RFC.

Методы аутентификации. Kerberos

Данный механизм может использовать внешние сертификаты X.509 как часть проверки TLS или подписи. Проверка подписей по сертификатам X.509, использующим SHA-1, является устаревшей и больше не используется без обходного пути. Дополнительную информацию см. в разделе FAQ об устаревании.

Метод аутентификации `kerberos` — это автоматизированный механизм для получения токена хранилища для субъектов Kerberos.

Kerberos — это протокол сетевой аутентификации, изобретенный в Массачусетском технологическом институте в 1980-х годах. Его название вдохновлено Цербером, трехголовым псом Аида из греческой мифологии. Три головы относятся к трем сущностям Kerberos: серверу аутентификации, серверу выдачи билетов и базе данных принципалов. Kerberos лежит в основе аутентификации в Active Directory, и его цель — распределить нагрузку по аутентификации в сети.

1. Предпосылки

Kerberos — это очень практичный метод авторизации. Другие методы аутентификации, такие как, например, LDAP, требуют лишь беглых знаний для настройки и использования. С другой стороны, Kerberos лучше всего использовать людям, уже знакомым с ним. Рекомендуем использовать более простые методы аутентификации, если задача может быть решена с их помощью. В противном случае, прежде чем приступать к использованию Kerberos, рекомендуем ознакомиться с его основами.

Независимо от способа получения знаний, перед использованием этого метода аутентификации убедитесь, что хорошо знакомы с высокоуровневой архитектурой Kerberos, и выполните следующие действия:

- Создайте достоверный файл `krb5.conf`
- Создайте достоверный файл `keytab`
- Аутентификация на сервере домена с файлом `keytab` с помощью `kinit`

Этих знаний и уже протестированной и подтвержденной рабочей средой достаточно для подготовки использования Kerberos в StarVault.

2. Конфигурация

- Включите аутентификацию Kerberos в StarVault:

```bash
starvault auth enable \
    -passthrough-request-headers=Authorization \
    -allowed-response-headers=www-authenticate \
    kerberos
```

- Создайте keytab для плагина Kerberos (этот keytab используется самим сервером StarVault, для входа в систему следует создать другой keytab):

```bash
$ ktutil
ktutil:  addent -password -p your_service_account@REALM.COM -e aes256-cts -k
1
Password for your_service_account@REALM.COM:
ktutil:  list -e
slot KVNO Principal
---- ---- ---------------------------------------------------------------------
---
  1    1            your_service_account@REALM.COM (aes256-cts-hmac-sha1-96)
ktutil:  wkt starvault.keytab
```

KVNO (`-k 1`) должен совпадать с KVNO учетной записи службы. Если это неверно, в журналах StarVault появится ошибка.

В `keytab` также можно добавить различные типы шифрования, например `-e rc4-hmac` с дополнительными командами `addent` .

Затем выполните base64-кодирование:

```bash
base64 starvault.keytab > starvault.keytab.base64
```

- Настройте метод аутентификации Kerberos с keytab и именем записи, которая будет использоваться для проверки входящих запросов на вход:

```bash
starvault write auth/kerberos/config \
    keytab=@starvault.keytab.base64 \
    service_account="starvault_svc"
```

- Настройте метод аутентификации Kerberos для связи с LDAP с помощью учетной записи службы, настроенной выше. Это пример конфигурации LDAP. Ваша конфигурация будет отличаться. Убедитесь, что сначала проверили свою конфигурацию с сервера StarVault с помощью такого инструмента, как `ldapsearch` .

```bash
starvault write auth/kerberos/config/ldap \
    binddn=starvault_svc@MATRIX.LAN \
    bindpass=$STARVAULT_SVC_PASSWORD \
    groupattr=sAMAccountName \
```

```
    groupdn="DC=MATRIX,DC=LAN" \
    groupfilter="(&(objectClass=group)(member:1.2.840.113556.1.4.1941:=
{{.UserDN}}))" \
    userdn="CN=Users,DC=MATRIX,DC=LAN" \
    userattr=sAMAccountName \
    upndomain=MATRIX.LAN \
    url=ldaps://somewhere.foo
```

Вышеуказанный метод LDAP использует тот же код, что и метод аутентификации LDAP . Дополнительные сведения о доступных параметрах см. в документации к нему.

- Настройте политики StarVault, которые должны быть предоставлены тем, кто успешно прошел аутентификацию, на основе их членства в группах LDAP. Поскольку этот метод идентичен методу аутентификации LDAP, дальнейшее обсуждение смотрите в разделе Разрешение членства в группах и сопоставление групп LDAP → политики.

BASH

```
starvault write auth/kerberos/groups/engineering-team \
    policies=engineers
```

Вышеуказанная группа предоставляет политику "engineers" тем, кто проходит аутентификацию через Kerberos и является членом LDAP-группы "engineering-team".

3. Аутентификация

На клиентской машине с действительными файлами krb5.conf и keytab выполните следующую команду:

BASH

```
starvault login -method=kerberos \
    username=grace \
    service=HTTP/my-service \
    realm=MATRIX.LAN \
    keytab_path=/etc/krb5/krb5.keytab  \
    krb5conf_path=/etc/krb5.conf \
    disable_fast_negotiation=false
```

- `krb5conf_path` — путь к действительному файлу `krb5.conf`, описывающему взаимодействие со средой Kerberos.
- `keytab_path` — путь к `keytab`, в которой находится запись для субъекта, аутентифицирующегося в StarVault. Файлы `keytab` должны быть защищены от других пользователей на общем сервере с помощью соответствующих разрешений на файлы.
- `username` — имя пользователя для записи в `keytab`, которое будет использоваться для входа в Kerberos. Это имя пользователя должно совпадать с учетной записью службы в LDAP.

- `service` — главное имя службы, которое следует использовать для получения билета службы для получения токена SPNEGO. Этот сервис должен существовать в LDAP.
- `realm` — имя пространства Kerberos. Это пространство должно совпадать с UPNDomain, настроенным на LDAP-соединении. Эта проверка чувствительна к регистру.
- `disable_fast_negotiation` — отключение метода аутентификации Kerberos, по умолчанию использующего согласование FAST. FAST — это механизм предварительной аутентификации для Kerberos. Включает в себя механизм туннелирования обменов перед аутентификацией с помощью защищенных сообщений KDC. FAST обеспечивает повышенную устойчивость к пассивным атакам на угадывание пароля. Некоторые распространенные реализации Kerberos не поддерживают согласование FAST.
- `remove_instance_name` — удаляет любые имена экземпляров из имени принципала сервиса Kerberos при разборе файла keytab. Например, если для keytab установлено значение true, то при разборе файла с именем принципала сервиса `foo/localhost@example.com`, CLI удалит из него имя экземпляра и останется только `foo@example.com`.

4. Устранение неполадок

4.1. Определите неисправную деталь

Определив неисправный участок пути, сможете направить усилия по отладке в наиболее полезное русло.

1. Используйте `ldapsearch`, войдите в систему на машине, на которой размещен StarVault, чтобы убедиться в работоспособности конфигурации LDAP.
2. Пройдите аутентификацию на сервере домена с помощью `kinit`, `keytab` и `krb5.conf`. Сделайте это как с `keytab` StarVault, так и с любой клиентской `keytab`, используемой для входа в систему. Это гарантирует, что ваша сеть Kerberos работает.
3. Войдя в клиентскую машину, убедитесь, что можете получить доступ к StarVault с помощью следующей команды: `curl $STARVAULT_ADDR/v1/sys/health`.

4.2. Составьте четкие шаги для воспроизведения проблемы

Если возможно, сделайте так, чтобы проблему мог воспроизвести кто-то вне вашей компании. Например, если ожидаете, что сможете войти в систему, используя такую команду, как:

BASH

```
starvault login -method=kerberos \
    username=my-name \
```

```
    service=HTTP/my-service \
    realm=EXAMPLE.COM \
    keytab_path=/etc/krb5/krb5.keytab  \
    krb5conf_path=/etc/krb5.conf
```

Затем убедитесь, что готовы поделиться результатами ошибки этой команды, содержимым файла `krb5.conf` и записями, перечисленными в файле `keytab`.

4.3. Дополнительные ресурсы для устранения неполадок

Библиотека StarVault Kerberos имеет рабочую интеграционную тестовую среду, которую можно использовать в качестве примера полноценной среды Kerberos и LDAP. Она работает через Docker и может быть запущена с помощью одной из следующих команд:

BASH

```
make integration
make dev-env
```

Эти команды запускают вариации сценария, который запускает полное окружение, добавляет пользователей и выполняет вход в систему от клиента.

5. API

Метод аутентификации Kerberos имеет полный HTTP API. Более подробную информацию можно найти в разделе API метода аутентификации Kerberos.

Методы аутентификации. Kubernetes

Данный механизм может использовать внешние сертификаты X.509 как часть проверки TLS или подписи. Проверка подписей по сертификатам X.509, использующим SHA-1, является устаревшей и больше не используется без обходного пути. Дополнительную информацию см. в разделе FAQ об устаревании.

Метод аутентификации kubernetes можно использовать для аутентификации в StarVault с помощью токена учетной записи сервиса Kubernetes. Этот метод аутентификации позволяет легко внедрить токен StarVault в под Kubernetes.

Также можете использовать токен учетной записи сервиса Kubernetes для входа в систему через JWT-аутентификацию. В разделе "Как работать с недолговечными токенами Kubernetes" кратко описана работа с недолговечными токенами Kubernetes, и как сравнить JWT-аутентификацию с аутентификацией Kubernetes.

Если вы обновляетесь до Kubernetes v1.21+, убедитесь, что параметр конфигурации `disable_iss_validation` установлен в **true**. Предполагая путь монтирования по умолчанию, вы можете проверить это с помощью команды `starvault read -field disable_iss_validation auth/kubernetes/config`. Дополнительные сведения см. в разделе Kubernetes 1.21 ниже.

1. Аутентификация

1.1. С помощью CLI

По умолчанию используется путь `/kubernetes`. Если этот метод авторизации был включен по другому пути, укажите в CLI `-path=/my-path`, где `my-path` — путь, по которому выключен метод аутентификации.

BASH

```
starvault write auth/kubernetes/login role=demo jwt=...
```

1.2. С помощью API

По умолчанию, конечная точка — `auth/kubernetes/login`. Если этот метод авторизации был включен по другому пути, используйте это значение вместо `kubernetes`.

BASH

```
curl \
    --request POST \
```

```
    --data '{"jwt": "<your service account jwt>", "role": "demo"}' \
    http://127.0.0.1:8200/v1/auth/kubernetes/login
```

В ответе будет содержаться токен `auth.client_token`:

```
{
  "auth": {
    "client_token": "38fe9691-e623-7238-f618-c94d4e7bc674",
    "accessor": "78e87a38-84ed-2692-538f-ca8b9f400ab3",
    "policies": ["default"],
    "metadata": {
      "role": "demo",
      "service_account_name": "myapp",
      "service_account_namespace": "default",
      "service_account_secret_name": "myapp-token-pd21c",
      "service_account_uid": "aa9aa8ff-98d0-11e7-9bb7-0800276d99bf"
    },
    "lease_duration": 2764800,
    "renewable": true
  }
}
```

2. Конфигурация

Методы аутентификации должны быть настроены заранее, прежде чем пользователи или машины смогут пройти аутентификацию. Эти шаги обычно выполняются оператором или инструментом управления конфигурацией.

1. Включите метод аутентификации Kubernetes:

   BASH
   ```
   starvault auth enable kubernetes
   ```

2. Используйте конечную точку `/config`, чтобы настроить StarVault на взаимодействие с Kubernetes. Используйте `kubectl cluster-info` для проверки адреса хоста и TCP-порта Kubernetes.

   BASH
   ```
   starvault write auth/kubernetes/config \
       token_reviewer_jwt="<your reviewer service account JWT>" \
       kubernetes_host=https://192.168.99.100:<your TCP port or blank for 443>
   \
       kubernetes_ca_cert=@ca.crt
   ```

Шаблон, который StarVault использует для аутентификации подов, зависит от обмена JWT-токеном по сети. Учитывая модель безопасности StarVault, это допустимо, поскольку StarVault является частью базы доверенных вычислений. В целом, приложения Kubernetes не должны передавать этот JWT другим приложениям, поскольку позволяет выполнять вызовы API от имени пода и может привести к непреднамеренному предоставлению доступа третьим лицам.

3. Создайте именованную роль:

```
starvault write auth/kubernetes/role/demo \
    bound_service_account_names=myapp \
    bound_service_account_namespaces=default \
    policies=default \
    ttl=1h
```

BASH

Эта роль авторизует учетную запись службы *myapp* в пространстве имен по умолчанию и наделяет ее политикой по умолчанию.

3. Kubernetes 1.21

Начиная с версии 1.21, функция Kubernetes BoundServiceAccountTokenVolume по умолчанию включена. Это изменяет JWT-токен, устанавливаемый в контейнеры по умолчанию, двумя способами, важными для аутентификации Kubernetes :

- Имеет срок действия и привязан к сроку жизни пода и учетной записи сервиса.
- Значение параметра "iss" в JWT зависит от конфигурации кластера.

Изменения в сроке жизни токена важны при настройке опции `token_reviewer_jwt`. Если используется токен с коротким сроком действия, Kubernetes отзовет его, как только под или учетная запись сервиса будут удалены, или если истечет срок действия, и StarVault больше не сможет использовать API TokenReview. Подробнее о том, как работать с недолговечными токенами Kubernetes, читайте ниже.

В ответ на изменения эмитента в StarVault был обновлен метод аутентификации Kubernetes, который по умолчанию не проверяет эмитент. API Kubernetes выполняет ту же проверку при просмотре токенов, поэтому включение проверки эмитента на стороне StarVault — дублирование работы. Без отключения проверки эмитента StarVault невозможно, чтобы одна конфигурация аутентификации Kubernetes работала для установленных по умолчанию токенов подов в Kubernetes 1.20 и в 1.21. Если хотите включить проверку эмитента в StarVault, смотрите раздел Обнаружение `issuer` учетной записи службы ниже.

3.1. Как работать с недолговечными токенами kubernetes

Существует несколько способов настройки аутентификации для подов Kubernetes, когда установленные по умолчанию токены подов недолговечны, каждый из которых имеет свои преимущества. В этой таблице представлены все варианты, каждый из которых более подробно описан ниже.

Опция	**Все токены недолговечны**	**Можно досрочно отозвать токены**	**Другие соображения**
Используйте локальный токен в качестве JWT-ревьюера	Да	Да	Требуется развертывание StarVault на кластере Kubernetes
Используйте JWT клиента в качестве JWT-ревьюера	Да	Да	Операционные накладные расходы
Использование долгоживущего токена в качестве JWT-ревьюера	Нет	Да	
Вместо этого используйте JWT-авторизацию	Да	Нет	

По умолчанию Kubernetes продлевает срок действия внедренных токенов учетных записей сервисов до года, чтобы сгладить переход к недолговечным токенам. Если хотите отключить эту функцию, задайте `--service-account-extend-token-expiration=false` для `kube-apiserver` или укажите свой собственный монтируемый том `serviceAccountToken`. Пример смотрите здесь.

3.1.1. Используйте токен учетной записи локальной службы в качестве JWT-ревьюера

При запуске StarVault в поде Kubernetes рекомендуется использовать токен локальной учетной записи службы пода. StarVault будет периодически перечитывать файл, чтобы поддерживать недолговечные токены. Чтобы использовать локальный токен и сертификат центра сертификации, опустите `token_reviewer_jwt` и `kubernetes_ca_cert` при настройке метода аутентификации. StarVault попытается загрузить их из файлов `token` и `ca.crt` соответственно в папке монтирования по умолчанию `/var/run/secrets/kubernetes.io/serviceaccount/`.

BASH

```
starvault write auth/kubernetes/config \
    kubernetes_host=https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT
```

Когда срок действия токена учетной записи сервиса истечет или он будет отозван, StarVault больше не сможет использовать API `TokenReview`, и аутентификация клиента завершится неудачей.

3.1.2. Используйте JWT клиента StarVault в качестве JWT-ревьюера.

При настройке Kubernetes auth можно опустить параметр `token_reviewer_jwt`, и StarVault будет использовать JWT клиента StarVault в качестве собственного auth-токена при взаимодействии с Kubernetes `TokenReview` API. Если StarVault работает в Kubernetes, также нужно установить `disable_local_ca_jwt`=true.

Это означает, что StarVault не хранит никаких JWT и позволяет использовать недолговечные токены повсеместно, но добавляет некоторые операционные накладные расходы на поддержание привязки ролей кластера к набору учетных записей служб, которые хотите аутентифицировать в StarVault. Каждому клиенту StarVault потребуется кластерная роль (`ClusterRole`) `system:auth-delegator`:

BASH

```
kubectl create clusterrolebinding starvault-client-auth-delegator \
  --clusterrole=system:auth-delegator \
  --group=group1 \
  --serviceaccount=default:svcaccount1 \
  ...
```

3.1.3. Продолжайте использовать долгоживущие токены

Можете создать долгоживущий секрет, используя инструкции здесь, и использовать его в качестве `token_reviewer_jwt`. В этом примере учетной записи службы хранилища потребуется роль (`ClusterRole`) `system:auth-delegator`:

BASH

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: starvault-k8s-auth-secret
  annotations:
    kubernetes.io/service-account.name: starvault
type: kubernetes.io/service-account-token
EOF
```

Использование этого способа позволяет сохранить прежние рабочие процессы, но не получить преимущества от повышения уровня безопасности недолговечных токенов.

3.1.4. Используйте JWT-авторизацию

Аутентификация Kubernetes специализируется на использовании API TokenReview от Kubernetes. Однако JWT-токены, которые генерирует Kubernetes, также можно проверять, используя Kubernetes в качестве OIDC-провайдера. В документации по методу JWT-аутентификации есть инструкции по настройке JWT-аутентификацию с Kubernetes в качестве OIDC-провайдера.

Это решение позволяет использовать недолговечные токены для всех клиентов и устраняет необходимость в ревьюере JWT. Однако клиентские токены не могут быть отозваны до истечения их TTL, поэтому рекомендуется держать TTL коротким с учетом этого ограничения.

3.2. Обнаружение `issuer` учетной записи обслуживания

Следующий раздел применим только в том случае, если установлено `disable_iss_validation=false`, но `disable_iss_validation=true` - это рекомендуемое значение для всех версий StarVault.

Для кластеров Kubernetes 1.21+ может потребоваться установить эмитента (`issuer`) учетной записи сервиса в то же значение, что и флаг `kube-apiserver` `--service-account-issuer`. Это связано с тем, что JWT-файлы учетных записей сервисов для этих кластеров могут иметь эмитента, специфичного для самого кластера, вместо старого значения по умолчанию `kubernetes/serviceaccount`. Если нет возможности проверить это значение напрямую, то можете выполнить следующую процедуру и найти нужное значение в поле `"iss"`:

BASH

```
echo '{"apiVersion": "authentication.k8s.io/v1", "kind": "TokenRequest"}' \
  | kubectl create -f- --raw
/api/v1/namespaces/default/serviceaccounts/default/token \
  | jq -r '.status.token' \
  | cut -d . -f2 \
  | base64 -d
```

В большинстве кластеров эта информация также доступна в конечной точке `.well-known/openid-configuration`:

BASH

```
kubectl get --raw /.well-known/openid-configuration | jq -r .issuer
```

Это значение затем используется при настройке аутентификации Kubernetes, например:

BASH

```
starvault write auth/kubernetes/config \
  kubernetes_host="https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT" \
  issuer="\"test-aks-cluster-dns-d6cbb78e.hcp.uksouth.azmk8s.io\""
```

4. Конфигурация kubernetes

Этот метод авторизации обращается к API Kubernetes TokenReview для проверки того, что предоставленный JWT все еще действителен. Kubernetes должен быть запущен с параметром `--service-account-lookup`. Начиная с Kubernetes 1.7 по умолчанию установлено значение `true`. В противном случае удаленные токены в Kubernetes не будут должным образом отозваны и смогут аутентифицироваться в этом методе авторизации.

Учетные записи служб, используемые в этом методе аутентификации, должны иметь доступ к API TokenReview. Если Kubernetes настроена на использование ролей RBAC, учетной записи службы должны быть предоставлены разрешения на доступ к этому API. Следующий пример `ClusterRoleBinding` может быть использован для предоставления этих разрешений:

YAML

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: role-tokenreview-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
  - kind: ServiceAccount
    name: starvault-auth
    namespace: default
```

5. API

Плагин Kubernetes Auth имеет полноценный HTTP API. Более подробную информацию можно найти в документации по API.

6. Пример кода

Следующий пример демонстрирует метод Kubernetes для аутентификации в StarVault.

GO

```go
package main

import (
    "fmt"
    "os"

    vault "github.com/hashicorp/vault/api"
```

```
    auth "github.com/hashicorp/vault/api/auth/kubernetes"
)

// Fetches a key-value secret (kv-v2) after authenticating to Vault with a
Kubernetes service account.
// For a more in-depth setup explanation, please see the relevant readme in the
hashicorp/vault-examples repo.
func getSecretWithKubernetesAuth() (string, error) {
    // If set, the VAULT_ADDR environment variable will be the address that
    // your pod uses to communicate with Vault.
    config := vault.DefaultConfig() // modify for more granular configuration

    client, err := vault.NewClient(config)
    if err != nil {
        return "", fmt.Errorf("unable to initialize Vault client: %w", err)
    }

    // The service-account token will be read from the path where the token's
    // Kubernetes Secret is mounted. By default, Kubernetes will mount it to
    // /var/run/secrets/kubernetes.io/serviceaccount/token, but an administrator
    // may have configured it to be mounted elsewhere.
    // In that case, we'll use the option WithServiceAccountTokenPath to look
    // for the token there.
    k8sAuth, err := auth.NewKubernetesAuth(
        "dev-role-k8s",
        auth.WithServiceAccountTokenPath("path/to/service-account-token"),
    )
    if err != nil {
        return "", fmt.Errorf("unable to initialize Kubernetes auth method: %w",
err)
    }

    authInfo, err := client.Auth().Login(context.TODO(), k8sAuth)
    if err != nil {
        return "", fmt.Errorf("unable to log in with Kubernetes auth: %w", err)
    }
    if authInfo == nil {
        return "", fmt.Errorf("no auth info was returned after login")
    }

    // get secret from Vault, from the default mount path for KV v2 in dev mode,
"secret"
    secret, err := client.KVv2("secret").Get(context.Background(), "creds")
    if err != nil {
        return "", fmt.Errorf("unable to read secret: %w", err)
    }

    // data map can contain more than one key-value pair,
    // in this case we're just grabbing one of them
    value, ok := secret.Data["password"].(string)
    if !ok {
```

```
        return "", fmt.Errorf("value type assertion failed: %T %#v",
secret.Data["password"], secret.Data["password"])
    }

    return value, nil
}
```

C# |

```
using System;
using System.IO;
using VaultSharp;
using VaultSharp.V1.AuthMethods;
using VaultSharp.V1.AuthMethods.Kubernetes;
using VaultSharp.V1.Commons;

namespace Examples
{
    public class KubernetesAuthExample
    {
        const string DefaultTokenPath = "path/to/service-account-token";

        // Fetches a key-value secret (kv-v2) after authenticating to Vault with
a Kubernetes service account.
        // For a more in-depth setup explanation, please see the relevant readme
in the hashicorp/vault-examples repo.
        public string GetSecretWithK8s()
        {
            var vaultAddr = Environment.GetEnvironmentVariable("VAULT_ADDR");
            if(String.IsNullOrEmpty(vaultAddr))
            {
                throw new System.ArgumentNullException("Vault Address");
            }

            var roleName = Environment.GetEnvironmentVariable("VAULT_ROLE");
            if(String.IsNullOrEmpty(roleName))
            {
                throw new System.ArgumentNullException("Vault Role Name");
            }

            // Get the path to service account token or fall back on default
path
            string pathToToken =
String.IsNullOrEmpty(Environment.GetEnvironmentVariable("SA_TOKEN_PATH")) ?
DefaultTokenPath : Environment.GetEnvironmentVariable("SA_TOKEN_PATH");
            string jwt = File.ReadAllText(pathToToken);

            IAuthMethodInfo authMethod = new KubernetesAuthMethodInfo(roleName,
jwt);
            var vaultClientSettings = new VaultClientSettings(vaultAddr,
authMethod);
```

```
            IVaultClient vaultClient = new VaultClient(vaultClientSettings);

            // We can retrieve the secret after creating our VaultClient object
            Secret<SecretData> kv2Secret = null;
            kv2Secret = vaultClient.V1.Secrets.KeyValue.V2.ReadSecretAsync(path:
"/creds").Result;

            var password = kv2Secret.Data.Data["password"];

            return password.ToString();
        }
    }
}
```
