Агент. Кэширование

Кэширование StarVault Agent позволяет кэшировать на стороне клиента ответы, содержащие вновь созданные токены, и ответы, содержащие арендованные секреты, сгенерированные на основе этих вновь созданных токенов. Агент также управляет обновлением кэшированных токенов и арендой.

1. Кэширование и обновления

Кэширование ответов и обновления управляются агентом только в этих конкретных сценариях:

1. Запросы на создание токена отправляются через агент. Это означает, что любые операции входа в систему, выполняемые с использованием различных методов аутентификации, и вызов конечных точек создания токена метода аутентификации токена через агент приведут к кэшированию ответа агентом. Ответы, содержащие новые токены, будут кэшироваться агентом только в том случае, если родительский токен уже управляется агентом или если новый токен является незарегистрированным токеном.
2. Запросы на создание выделенных секретных данных отправляются через агента с использованием токенов, которые уже находятся под управлением агента. Это означает, что любые динамические учетные данные, которые выдаются с использованием токенов, управляемых агентом, будут кэшироваться, а их обновление будет обеспечено.

2. Постоянный кэш

StarVault Agent может восстанавливать токены и договоры аренды из файла постоянного кэша, созданного предыдущим процессом StarVault Agent.

Для получения дополнительной информации об этой функции обратитесь к странице постоянного кэширования StarVault Agent.

3. Удаление кэша

Удаление записей кэша, относящихся к секретам, произойдет, когда агент больше не сможет их обновлять. Это может произойти, когда срок действия секретов достигнет максимального значения или если обновление приведет к ошибкам.

Агент выполняет некоторые операции по удалению кэша, используя определенные типы запросов и коды ответов. Например, если запрос на аннулирование токена отправляется через агента и если переадресованный запрос на сервер хранилища выполняется успешно, агент удаляет все записи кэша, связанные с отозванным токеном. Аналогичным образом, любая операция по аннулированию аренды также будет перехвачена агентом, и соответствующие записи в кэше будут удалены.

Обратите внимание, что, хотя агент удаляет записи из кэша по истечении секретного срока действия и при перехвате запросов на отзыв, агент все равно может не знать об отзывах, которые происходят в результате прямого взаимодействия клиента с сервером хранилища. Это потенциально может привести к устареванию записей в кэше. Для управления устаревшими записями в кэше доступна функция endpoint `/agent/v1/cache-clear` (см. ниже), позволяющая вручную удалять записи в кэше на основе некоторых критериев запроса, используемых для индексации записей в кэше.

4. Уникальность запроса

Чтобы обнаруживать повторяющиеся запросы и возвращать кэшированные ответы, агенту потребуется способ уникальной идентификации запросов. В сегодняшнем виде это вычисление использует упрощенный подход (который может измениться в будущем), заключающийся в сериализации и хэшировании HTTP-запроса вместе со всеми заголовками и телом запроса. Это значение хэша затем используется в качестве индекса в кэше для проверки доступности ответа. Следствием такого подхода является то, что значение хэша для любого запроса будет отличаться, если какие-либо данные в запросе будут изменены. Это имеет побочный эффект, приводящий к ложным срабатываниям, если, скажем, изменен порядок следования параметров запроса. Пока запросы поступают без каких-либо изменений, поведение кэширования должно быть согласованным. Идентичные запросы с по-разному упорядоченными значениями запросов приведут к дублированию записей в кэше. Эвристическое предположение о том, что клиенты будут использовать согласованные механизмы для выполнения запросов, что приведет к получению согласованных значений хэша для каждого запроса, является идеей, на которой построена функциональность кэширования.

5. Управление обновлением

Токены и договоры аренды продлеваются агентом с помощью секретного средства продления, доступного через Go API сервера хранилища. Агент выполняет все операции в памяти и ничего не сохраняет в хранилище. Это означает, что при отключении агента все операции по обновлению немедленно прекращаются, и агент не может возобновить обновления постфактум. Обратите внимание, что завершение работы агента не означает

отзыва секретов, вместо этого это означает только то, что агент хранилища больше не несет ответственности за обновление всех действительных неотозванных секретов.

5.1. Агент CLI

Адрес прослушивателя агента будет получен интерфейсом командной строки через переменную среды `STARVAULT_AGENT_ADDR`. Это должен быть полный URL-адрес, например `"http://127.0.0.1:8200"`.

6. API

6.1. Очистка кэша

Эта конечная точка очищает кэш на основе заданных критериев. Чтобы использовать этот API, необходимо заранее знать некоторую информацию о том, как агент кэширует значения. Каждый ответ, который кэшируется в агенте, будет проиндексирован по некоторым параметрам в зависимости от типа запроса. Этими факторами могут быть `token`, принадлежащий кэшированному ответу, `token_accessor` токена, принадлежащего кэшированному ответу, `request_path`, который привел к кэшированному ответу, `аренда`, которая привязана к кэшированному ответу, пространство имен, к которому принадлежит кэшированный ответ, и еще несколько факторов. Этот API предоставляет некоторые факторы, с помощью которых извлекаются и удаляются связанные записи кэша. Для слушателей без включенного кэширования этот API по-прежнему будет доступен, но ничего не будет делать (кэш не нужно очищать) и вернет `200` ответов.

Метод	Пусть	Генерирует
`POST`	`/agent/v1/cache-clear`	`200 application/json`

6.2. Параметры

- `type (strings: required)` - тип записей кэша, которые необходимо удалить. Допустимыми значениями являются `request_path`, `lease`, `token`, `token_accessory` и `all`. Если для параметра `type` задано значение `all`, весь кэш очищается.
- `value (strings: required)` - точное значение или префикс значения для выбранного `type`. Этот параметр необязателен, если для `type` задано значение `all`.
- `namespace (string: optional)` - это применимо только в том случае, если для `type` задано значение `request_path`. Пространство имен, из которого должны быть удалены записи кэша для данного пути запроса.

6.3. Пример полезной нагрузки

```C
{
  "type": "token",
  "value": "hvs.rlNjegSKykWcplOkwsjd8bP9"
}
```

6.4. Пример запроса

```BASH
$ curl \
    --request POST \
    --data @payload.json \
    http://127.0.0.1:1234/agent/v1/cache-clear
```

7. Конфигурация (кэш)

Наличие блока `кэширования` верхнего уровня любым способом (включая пустой блок `кэширования`) активирует `кэширование`. Блок `кэширования` верхнего уровня содержит следующую конфигурационную запись:

- `persist (object: optioanl)` - конфигурация для постоянного кэша.

Блок `кэширования` также поддерживает значения конфигурации `use_auto_auth_token`, `enforce_consistency` и `when_inconsistent` блока `api_proxy`, описанные в документации API Proxy, только для поддержания обратной совместимости. Эта конфигурация не может быть указана вместе с эквивалентами `api_proxy`, не должна быть предпочтительнее настройки этих значений в блоке `api_proxy`, и `api_proxy` должен быть предпочтительным местом для настройки этих значений.

Когда определен блок `кэша`, в конфигурации также должен быть определен по крайней мере один шаблон или слушатель, в противном случае нет возможности использовать кэш.

8. Конфигурация (Сохраняющаяся)

Это общие значения конфигурации, которые хранятся в блоке `сохранения`:

- `type (string: required)` - тип используемого постоянного кэша, например, `kubernetes`.

при использовании HCL это может быть использовано в качестве ключа для блока, например, `сохранить "kubernetes" {…}`. В настоящее время поддерживается только kubernetes.

- `path (string: required)` - путь на диске, по которому должен быть создан или восстановлен файл постоянного кэша.
- `keep_after_import (bool: optional)` - Если установлено значение `true`, восстановленный файл кэша не удаляется. По умолчанию установлено значение `false`.
- `exit_on_err (bool: optional)` - если установлено значение true, при возникновении каких-либо ошибок во время восстановления сохраняемого кэша StarVault Agent завершит работу с ошибкой. По умолчанию установлено значение `true`.
- `service_account_token_file (string: optional)` - Если для параметра `type` задано значение `kubernetes`, это настраивает путь на диске, по которому можно найти токен учетной записи службы Kubernetes. По умолчанию используется `/var/run/secrets/kubernetes.io/serviceaccount/token`.

9. Конфигурация (слушатель)

- `listener (array of objects: required)` - конфигурация для прослушивателей.

На верхнем уровне может быть один или несколько блоков `прослушивателя`. Добавление прослушивателя активирует прокси-сервер API и позволяет [прокси-серверу API использовать кэш, если он настроен. Эти значения конфигурации являются общими для блоков прослушивателя `tcp` и `unix`. Блоки типа `tcp` поддерживают стандартные параметры слушателя `tcp`. Кроме того, параметр `role string` доступен как часть верхнего уровня блока `прослушивателя`, который можно настроить на значение `metrics_only` для обслуживания только показателей, или на роль по умолчанию, `default`, которая обслуживает все (включая показатели).

- `type (string: required)` - тип используемого прослушивателя. Допустимые значения - `tcp` и `unix`.

при использовании HCL это значение может использоваться в качестве ключа для блока, например, `прослушиватель "tcp" {…}`.

- `address (string: required)` - адрес, который прослушиватель должен прослушивать. Это может быть либо URL-адрес при использовании `tcp`, либо путь к файлу при использовании `unix`. Например, `127.0.0.1:8200`, либо `/path/to/socket`. Значение по умолчанию - `127.0.0.1:8200`.
- `tls_disable (bool: false)` - указывает, будет ли TLS отключен.
- `tls_key_file (string: optional)` - указывает путь к закрытому ключу для сертификата.
- `tls_cert_file (string: optional)` - указывает путь к сертификату для TLS.

10. Пример конфигурации

Вот пример конфигурации кэша с необязательным блоком `сохранения`, наряду с обычным прослушивателем и прослушивателем, который обслуживает только показатели.

```
# другие блоки конфигурации агента
# ...

cache {
    persist = {
        type = "kubernetes"
        path = "/vault/agent-cache/"
        keep_after_import = true
        exit_on_err = true
        service_account_token_file = "/tmp/serviceaccount/token"
    }
}

listener "tcp" {
    address = "127.0.0.1:8100"
    tls_disable = true
}

listener "tcp" {
    address = "127.0.0.1:3000"
    tls_disable = true
    role = "metrics_only"
}
```

Методы автоматической аутентификации. Сертификат

Метод `cert` (сертификат) использует настроенные сертификаты TLS из строфы `vault` конфигурации агента и принимает опциональный параметр `name`. Возможность использовать сертификаты, отличные от тех, что используются в `блоке конфигурации хранилища`, отсутствует.

Настоятельно рекомендуется указывать параметры TLS в строке конфигурации в методе auth, чтобы избежать использования кэш-памяти агента, если она также включена, тех же параметров TLS при проксировании запросов. Если настройки TLS отсутствуют в строфе конфигурации, агент и прокси вернутся к использованию настроек TLS из соответствующих блоков конфигураций хранилища.

1. Конфигурация

- `name ( string: optional)` - роль доверенного сертификата, которая должна использоваться при аутентификации с помощью TLS. Если поле `name` не указано, метод `auth` будет пытаться аутентифицироваться по всем доверенным сертификатам.
- `ca_cert ( string: optional)` - путь на локальном диске к одному сертификату ЦС в PEM-кодировке для проверки SSL-сертификата сервера StarVault.
- `client_cert ( string: optional)` - путь на локальном диске к одному сертификату клиента в PEM-кодировке для использования при аутентификации методом `cert auth`.
- `client_key ( string: optional)` - путь на локальном диске к одному закрытому ключу в PEM-кодировке, соответствующему клиентскому сертификату из `client_cert`.
- `reload (bool: optional, default: false)` - Если `true`, то при каждой попытке аутентификации локальная пара ключей x509 будет перезагружаться с диска. Это полезно в ситуациях, когда клиентские сертификаты недолговечны и автоматически обновляются.

Агент. Генерация конфигураций

Генерирует простой файл конфигурации StarVault Agent из заданных параметров. В настоящее время единственным поддерживаемым типом конфигурации является `env-template`, который помогает генерировать файл конфигурации с шаблонами переменных среды для запуска StarVault Agent в режиме супервизора процесса.

Для каждого указанного секретного `-пути` команда попытается сгенерировать одну или несколько записей `env_template` на основе ключей `JSON`, хранящихся в указанном секрете. Если секретный `-путь` заканчивается на `/*`, команда попытается выполнить рекурсию по дереву секретов, корнем которого является указанный путь, генерируя записи `env_template` для каждого обнаруженного секрета. В настоящее время поддерживаются только пути kv-v1 и kv-v2.

Команда, указанная в параметре `-exec`, будет использоваться для создания записи `exec`, которая сообщит StarVault Agent, какой дочерний процесс следует запустить.

В дополнение к записям `env_template` команда генерирует раздел `auto_auth` с методом аутентификации `token_file`. Хотя этот метод очень удобен для локального тестирования, его **НЕ** следует использовать в производственной среде. В производственной среде вместо этого используйте любой другой метод Auto-Auth.

1. Пример

Перед генерацией файла конфигурации давайте вставим секретный файл `foo`:

BASH

```
$ starvault kv put -mount=secret foo user="admin" password="s3cr3t"
```

Создайте файл конфигурации агента, который будет ссылаться на `secret/foo`:

BASH

```
$ starvault agent generate-config \
        -type="env-template" \
        -exec="./my-app arg1 arg2" \
        -namespace="my/ns/" \
        -path="secret/foo" \
        my-config.hcl
```

Ожидаемый результат:

BASH

```
Успешно сгенерирован файл конфигурации "my-config.hcl"!
Внимание: сгенерированный файл использует метод аутентификации "token_file",
```

```
который не подходит для производственных сред.
```

Это создаст файл `my-config.hcl` в текущем каталоге с содержимым, аналогичным следующему:

```
auto_auth {

  method {
    type = "token_file"

    config {
      token_file_path = "/Users/avean/.vault-token"
    }
  }
}

template_config {
  static_secret_render_interval = "5m"
  exit_on_retry_failure         = true
}

vault {
  address = "http://localhost:8200"
}

env_template "FOO_PASSWORD" {
  contents             = "{{ with secret \"secret/data/foo\" }}{{
.Data.data.password }}{{ end }}"
  error_on_missing_key = true
}
env_template "FOO_USER" {
  contents             = "{{ with secret \"secret/data/foo\" }}{{
.Data.data.user }}{{ end }}"
  error_on_missing_key = true
}

exec {
  command                   = ["./my-app", "arg1", "arg2"]
  restart_on_secret_changes = "always"
  restart_stop_signal       = "SIGTERM"
}
```

2. Использование

В дополнение к стандартному набору флагов, включенных во все команды, доступны следующие флаги.

- `type (string: <required>)` — тип файла конфигурации для генерации; в настоящее время поддерживается только `env-template`.
- `path (string: "")` — путь к секрету kv-v1 или kv-v2 (например, `secret/data/foo`, `kv-v2/my-app/`**); разрешены несколько секретов и подстановочные знаки** .
- `-exec (string: "env")` — команда для выполнения в режиме супервизора процесса агента.
