Прокси для API

Функциональность API-прокси StarVault Proxy обеспечивает возможность использовать его в качестве промежуточного звена (прокси) для передачи запросов к API StarVault.

1. Функциональность

Блок конфигурации `listener` для StarVault Proxy настраивает слушателя для StarVault Proxy. Если его `role` не установлена на `metrics_only`, он будет действовать как прокси для сервера StarVault, который был настроен в блоке конфигурации `vault` Proxy Agent. Это позволяет обращаться к API StarVault через API Proxy и поддерживает настройку поведения аутентификации: использовать токен Auto-Auth автоматически или разрешать его использование при необходимости — в зависимости от конфигурации, описанной ниже.

Если слушатель был настроен вместе с блоком конфигурации `cache`, API Proxy сначала попытается использовать подсистему кэша для соответствующих запросов, прежде чем перенаправить запрос в StarVault. Дополнительные сведения о кэшировании см. в документации по кэшированию.

2. Использование токена автоматической аутентификации

StarVault Proxy позволяет легко аутентифицироваться в StarVault в самых разных средах с помощью Auto-Auth. При установке конфигурации `use_auto_auth_token` (см. ниже) клиентам не нужно будет предоставлять токен StarVault для запросов, созданных с помощью Agent. При активированной конфигурации, если входящий запрос не содержит токена, для его перенаправления на сервер StarVault будет использован токен автоматической аутентификации. Если в запросе уже присутствует токен, он будет иметь приоритет и будет использован вместо Auto-Auth токена.

3. Принудительное использование токена автоматической аутентификации

StarVault Proxy можно настроить на принудительное использование токена автоматической аутентификации, используя значение `force` для параметра `use_auto_auth_token`. Эта конфигурация переопределяет поведение по умолчанию, описанное выше в разделе Использование токена автоматической аутентификацией, и вместо этого игнорирует любой

существующий токен StarVault в запросе и вместо этого использует токен автоматической аутентификации.

4. Конфигурация (api_proxy)

Блок `api_proxy` верхнего уровня имеет следующие записи конфигурации:

- `use_auto_auth_token` `(bool/string: false)` — если установлено, запросы к Agent без токена StarVault будут пересылаться на сервер StarVault с прикрепленным токеном автоматической аутентификации. Если запросы уже содержат токен, эта конфигурация будет переопределена, и токен в запросе будет использоваться для пересылки запроса на сервер StarVault. Если установлено значение `«force»`, Agent будет использовать токен автоматической аутентификации, перезаписывая прикрепленный токен StarVault, если он установлен.

5. Пример конфигурации

Ниже приведен пример конфигурации `listener` вместе с конфигурацией `api_proxy` для принудительного использования токена `auto_auth` и обеспечения согласованности.

```
# Другие блоки конфигурации прокси-сервера StarVault
# ...

api_proxy {
  use_auto_auth_token = "force"
}

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

Методы автоматической аутентификации. AppRole

Метод `approle` считывает идентификатор роли и секретный идентификатор из файлов и отправляет значения в метод AppRole Auth.

Метод кэширует значения, и можно смело удалять файлы с идентификатором роли/ секретным идентификатором после их считывания. На самом деле, по умолчанию после считывания секретного идентификатора агент удаляет файл. Новые файлы или значения, записанные в ожидаемых местах, будут использоваться при следующей аутентификации, а новые значения будут кэшироваться.

1. Конфигурация

- `role_id_file_path (string: required)` - путь к файлу с идентификатором роли.
- `secret_id_file_path (string: optional)` - путь к файлу с секретным идентификатором. Если не задано, будет использоваться только `role-id`. В этом случае для AppRole должно быть установлено значение `bind_secret_id` равное `false`, иначе StarVault Agent не сможет войти в систему.
- `remove_secret_id_file_after_reading (bool: optional, defaults to true)` - это значение может быть установлено в ``falsem, чтобы отключить стандартное поведение удаления файла секретного идентификатора после его чтения.
- `secret_id_response_wrapping_path (string: optional)` - если установлено, значение в `secret_id_file_path` будет ожидаться как **Response-Wrapping Token**, содержащий вывод конечной точки получения секретного идентификатора для роли `(например, auth/approle/role/webservers/secret-id)`, а путь создания response-wrapping token должен соответствовать значению, установленному здесь.

2. Пример конфигурации

Ниже приведен пример конфигурации с использованием `approle` для включения автоматической проверки подлинности и создания как текстового приемника токенов, так и файла приемника токенов в оболочке ответа:

```
JSON
pid_file = "./pidfile"

vault {
```

```
  address = "https://127.0.0.1:8200"
}

auto_auth {
  method {
    type      = "approle"

    config = {
      role_id_file_path = "roleid"
      secret_id_file_path = "secretid"
      remove_secret_id_file_after_reading = false
    }
  }

  sink {
    type = "file"
    wrap_ttl = "30m"
    config = {
      path = "sink_file_wrapped_1.txt"
    }
  }

  sink {
    type = "file"
    config = {
      path = "sink_file_unwrapped_2.txt"
    }
  }
}

api_proxy {
  use_auto_auth_token = true
}

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

Автоматическая аутентификация

Функции автоматической аутентификации в StarVault Agent и StarVault Proxy позволяют легко проходить аутентификацию в различных средах.

1. Функциональные возможности

Автоматическая аутентификация состоит из двух частей: метода, который представляет собой способ аутентификации, используемый в текущей среде, а также любого количества стоков, которые представляют собой места, куда агент должен записывать токен каждый раз, когда текущее значение токена меняется.

Когда StarVault Agent или StarVault Proxy запускается с включенной функцией Auto-Auth, он попытается получить токен StarVault, используя настроенный метод. В случае неудачи он экспоненциально откатится назад, а затем повторит попытку. В случае успеха, если метод аутентификации не настроен на обертку токенов, он будет обновлять полученный токен до тех пор, пока обновление не будет запрещено или не произойдет сбой, и тогда он попытается пройти повторную аутентификацию.

Каждый раз, когда аутентификация проходит успешно, токен записывается в настроенные источники, в зависимости от их конфигурации.

2. Расширенный функционал

Источники поддерживают некоторые дополнительные функции, в том числе возможность шифрования записываемых значений или оболочки ответа.

Оба механизма могут использоваться одновременно; в этом случае значение будет зашифровано, а затем получено ответной оболочкой.

3. Токены для упаковки ответа

Токены могут быть упакованы в ответ двумя способами:

1. По методу авторизации. Это позволяет конечному клиенту узнать `creation_path` токена, что помогает предотвратить атаки Man-In-The-Middle (MITM). Однако, поскольку автоматический аутентификатор не может развернуть токен и повторно развернуть его без изменения `creation_path`, мы не можем обновить токен; обновление токена остается за конечным клиентом. Агент и прокси остаются демонизированными в этом

режиме, поскольку некоторые методы аутентификации позволяют повторную аутентификацию по определенным событиям.

2. Любой из источников токенов. Поскольку может быть настроено более одного источника, токен должен быть упакован после его получения, а не упакован системой StarVault при его возврате. В результате `creation_path` всегда будет `sys/wrapping/wrap`, и проверка этого поля не может использоваться в качестве защиты от MITM-атак. Тем не менее, этот режим позволяет системе автоматической аутентификации обновлять токен для конечного клиента и автоматически проходить повторную аутентификацию по истечении срока его действия.

4. Шифрование токенов

Поддержка зашифрованных токенов является экспериментальной; если форматы ввода/вывода изменятся, мы приложим все усилия, чтобы обеспечить обратную совместимость.

Токены можно зашифровать, используя метод Диффи-Хеллмана для генерации эфимерного ключа. В этом механизме клиент, получающий токен, записывает сгенерированный открытый ключ в файл. Стоковый сервер, ответственный за запись токена для этого клиента, ищет этот открытый ключ и использует его для вычисления общего секретного ключа, который затем используется для шифрования токена с помощью AES-GCM. Одноразовый номер, зашифрованная полезная нагрузка и открытый ключ приемника затем записываются в выходной файл, где клиент может вычислить общий секрет и расшифровать значение токена.

Шифрование токенов не является защитой от MITM-атак! Эта функция предназначена для обеспечения секретности и защиты от сохранения голых значений токенов. MITM, способный записывать выходные файлы поглотителя и/или входные файлы клиента с открытым ключом, может атаковать этот обмен. Настоятельно рекомендуется использовать TLS для защиты передачи токенов.

Для защиты от атак MITM Агенту и Прокси могут быть предоставлены дополнительные аутентифицированные данные (AAD). Эти данные записываются как часть тега AES-GCM и должны совпадать как на Агенте и Прокси, так и на клиенте. Это, подразумевает, что защита этих AAD становится важной, но это создает еще один уровень, который придется преодолеть злоумышленнику. Например, если у злоумышленника есть доступ к файловой системе, в которую записывается токен, но нет доступа к чтению конфигурации или переменных окружения, этот AAD может быть сгенерирован и передан Агенту или Прокси и клиенту таким образом, что злоумышленнику будет сложно его найти.

При использовании AAD всегда лучше, чтобы он был как можно более свежим; сгенерируйте значение и передайте его клиенту и агенту или прокси при запуске. Кроме того, агент и прокси используют модель Trust On First Use; после того как он найдет сгенерированный

открытый ключ, он будет использовать его повторно, а не искать новые значения, которые были записаны.

Если вы пишете клиента, использующего эту функцию, вам, вероятно, будет полезно посмотреть на библиотеку dhutil. Она показывает ожидаемый формат ввода открытого ключа и формат вывода конверта.

5. Конфигурация

Блок верхнего уровня `auto_auth` имеет две конфигурационные записи:

- `method (объект: обязательно)` - конфигурация для метода
- `sinks (массив объектов: необязательно)` - конфигурация для sinks

6. Конфигурация (метод)

Автоматическая авторизация не поддерживает использование токенов с ограниченным числом использований. Автоматическая авторизация не отслеживает количество оставшихся использований и может позволить токену исчерпать себя, прежде чем попытаться его обновить. Например, при использовании AppRole auto-auth в качестве значения для `token_num_uses` необходимо использовать 0 (то есть неограниченное количество).

Это общие значения конфигурации, которые находятся в `блоке метода`:

- `type (string: required)` - тип используемого метода.

Примечание:

При использовании HCL это может использоваться в качестве ключа для блока.

- `mount_path (string: необязательно)` - путь подключения метода. Если не указан, по умолчанию используется значение `auth/<method type>`.
- `namespace (string: необязательно)` - пространство имен, в котором находится метод. Порядок приоритета следующий: данная настройка самая младшая, за ней следует переменная окружения `VAULT_NAMESPACE`, а затем опция командной строки `-namespace` с наивысшим приоритетом. Если ни один из этих параметров не указан, по умолчанию используется корневое пространство имен. Обратите внимание, что поскольку обертка и шаблонизация ответов sink также основаны на клиенте, созданном с помощью auto-auth, они используют одно и то же пространство имен.
- `wrap_ttl (string или integer: необязательно)` - если указано, то записанный токен будет упакован в ответ с помощью автоматического аутентификатора. Это более

безопасно, чем упаковка с помощью стоков, но не позволяет автоматическому аутентификатору обновлять токен или автоматически проходить повторную аутентификацию по истечении срока его действия. Вместо простой строки, записываемое значение будет представлять собой структуру SecretWrapInfo, закодированную в JSON.

- `min_backoff (string или integer: «1s»)` - минимальное время ожидания автоматического аутентификатора перед повторной попыткой после неудачной попытки аутентификации. Откат будет начинаться с заданного значения и удваиваться (с некоторой случайностью) после следующих неудач, ограничиваясь значением `max_backoff`. Если используется шаблонизация агента, это значение также используется в качестве минимального времени отката для сервера шаблонизации.
- `max_backoff (string или integer: «5m»)` - максимальное время, которое Агент будет откладывать перед повторной попыткой авторизации после неудачной попытки. Откат будет начинаться с `min_backoff` и удваиваться (с некоторой случайностью) после последовательных неудач, достигая `max_backoff`. Если используется шаблонизация Агента, это значение также используется в качестве максимального времени отката для сервера шаблонизации. `max_backoff` - это продолжительность между повторными попытками, а не продолжительность, в течение которой будут выполняться повторные попытки перед сдачей.

7. Конфигурация (Sinks)

- `type (string: required)` - тип используемого метода, например, `file`.

При использовании HCL это может быть использовано в качестве ключа для блока, например `sink «file» {…}`.

- `wrap_ttl` (string или integer: необязательно) - если указано, то записанный токен будет упакован в ответ стоком. Это менее безопасно, чем обертывание методом, но позволяет auto-auth сохранять токен обновленным и автоматически проходить повторную аутентификацию по истечении срока его действия. Вместо простой строки записанное значение будет представлять собой структуру SecretWrapInfo, закодированную в JSON.
- `dh_type (string: optional)` - если указано, тип обмена Диффи-Хеллмана, который будет выполняться, то есть какие шифры и/или кривые. В настоящее время поддерживается только `curve25519`.
- `dh_path (string: требуется, если задан dh_type)` - путь, с которого автоаутентификация должна считывать начальные параметры клиента (например, открытый ключ curve25519).
- `derive_key (bool: false)` - если указано, окончательный ключ шифрования вычисляется с помощью HKDF-SHA256 для получения ключа из вычисленного общего

секрета и двух открытых ключей для повышения безопасности. Это рекомендуется, если обратная совместимость не является проблемой.

- `aad (string: необязательно)` - если указано, дополнительные аутентифицированные данные для использования при AES-GCM-шифровании токена. Может быть любой строкой, включая сериализованные данные.
- `aad_env_var (string: необязательно)` - если указано, то AAD будет считываться из данной переменной окружения, а не из значения в конфигурационном файле.
- `config (object: required)` - Конфигурация самого блока. Информацию о каждом блоке смотри на боковой панели.

8. Примеры автоматической авторизации

Объекты конфигурации автоматического аутентификатора принимают две разные формы, когда задаются в HCL и JSON. Следующие примеры призваны прояснить различия между этими двумя форматами:

Источники (формат HCL)

Формат HCL может определять любое количество объектов источника с необязательным объектом `sinks {…}`.

В соответствующем формате JSON должен быть указан массив `«sinks»: […]`, в котором содержатся все `источники` JSON-объектов.

JSON

```
// Other StarVault Agent or StarVault Proxy configuration blocks
// ...

auto_auth {
  method {
    type = "approle"

    config = {
      role_id_file_path = "/etc/vault/roleid"
      secret_id_file_path = "/etc/vault/secretid"
    }
  }

  sinks {
    sink {
      type = "file"

      config = {
        path = "/tmp/file-foo"
      }
    }
```

```
  }
}
```

В следующем допустимом HCL объект `sinks` опускается при указании нескольких источников.

JSON

```
// Other StarVault Agent or StarVault Proxy configuration blocks
// ...

auto_auth {
  method {
    type = "approle"

    config = {
      role_id_file_path = "/etc/vault/roleid"
      secret_id_file_path = "/etc/vault/secretid"
    }
  }

  sink {
    type = "file"

    config = {
      path = "/tmp/file-foo"
    }
  }

  sink {
    type = "file"

    config = {
      path = "/tmp/file-bar"
    }
  }
}
```

Источники (формат JSON)

Следующая конфигурация JSON иллюстрирует необходимость использования массива `sinks: […]` массив, содержащий любое количество объектов `источников` :

JSON

```
{
  "auto_auth" : {
    "method" : [
      {
        type = "approle"

        config = {
          role_id_file_path = "/etc/vault/roleid"
```

```
          secret_id_file_path = "/etc/vault/secretid"
        }
      }
    ],
    "sinks" : [
      {
        "sink" : {
          type = "file"

          config = {
            path = "/tmp/file-foo"
          }
        }
      }
    ]
  }
}
```

Множественные источники определяются путем добавления нескольких объектов источников в массив источников:

JSON

```
{
  "auto_auth" : {
    "method" : [
      {
        type = "approle"

        config = {
          role_id_file_path = "/etc/vault/roleid"
          secret_id_file_path = "/etc/vault/secretid"
        }
      }
    ],
    "sinks" : [
      {
        "sink" : {
          type = "file"

          config = {
            path = "/tmp/file-foo"
          }
        }
      },
      {
        "sink" : {
          type = "file"

          config = {
            path = "/tmp/file-bar"
          }
```

```
        }
      }
    ]
  }
}
```

9. Методы автоматической аутентификации в хранилище

Автоматическая аутентификация - это механизм, используемый StarVault Agent и StarVault Proxy для автоматической аутентификации в StarVault при наличии набора параметров, позволяющих выполнить аутентификацию. Доступные методы и их использование и настройка приведены ниже:

- AppRole
- Сертификат
- JWT
- Kerberos
- Создание файла токена