**Аутентификация**

Обзор аутентификации gRPC, включая встроенные механизмы аутентификации, а также способы подключения собственных систем аутентификации.

Обзор

gRPC разработан для работы с различными механизмами аутентификации, что позволяет безопасно использовать gRPC для взаимодействия с другими системами. Вы можете использовать наши поддерживаемые механизмы — SSL/TLS с аутентификацией на основе токенов Google или без неё — или подключить собственную систему аутентификации, расширив предоставленный нами код.

gRPC также предоставляет простой API аутентификации, который позволяет предоставлять всю необходимую информацию для аутентификации в виде Credentials при создании канала или выполнении вызова.

Поддерживаемые механизмы аутентификации

В gRPC встроены следующие механизмы аутентификации:

* **SSL/TLS**: gRPC поддерживает интеграцию с SSL/TLS и рекомендует использовать SSL/TLS для аутентификации сервера и шифрования всех данных, которыми обмениваются клиент и сервер. Для взаимной аутентификации клиентам доступны дополнительные механизмы.
* **ALTS**: gRPC поддерживает ALTS в качестве механизма транспортной безопасности, если приложение работает на Compute Engine или Google Kubernetes Engine (GKE). Подробнее см. на одной из следующих страниц для конкретного языка: ALTS на C++, ALTS на Go, ALTS на Java, ALTS на Python.
* **Аутентификация на основе токенов в Google**: gRPC предоставляет универсальный механизм (описанный ниже) для прикрепления учётных данных на основе метаданных к запросам и ответам. Дополнительная поддержка получения токенов доступа (обычно токенов OAuth2) при доступе к API Google через gRPC предоставляется для определённых потоков аутентификации: вы можете увидеть, как это работает, в наших примерах кода ниже. В целом этот механизм необходимо использовать *вместе с* SSL/TLS на канале — Google не разрешает подключения без SSL/TLS, а большинство реализаций gRPC не позволяют отправлять учётные данные по незашифрованному каналу.

Предупреждение

Учетные данные Google следует использовать только для подключения к сервисам Google. Отправка выданного Google токена OAuth2 в сервис, не принадлежащий Google, может привести к краже этого токена и его использованию для выдачи себя за клиента сервисов Google.

API аутентификации

gRPC предоставляет простой API аутентификации, основанный на единой концепции объектов Credentials, которые можно использовать при создании всего канала gRPC или отдельного вызова.

Типы учетных данных

Учетные данные могут быть двух типов:

* **Учетные данные канала**, которые прикреплены к Channel, например, учетные данные SSL.
* **Параметры вызова**, которые прикрепляются к вызову (или ClientContext в C++).

Вы также можете объединить их в CompositeChannelCredentials и указать, например, сведения о SSL для канала, а также учётные данные для каждого вызова, совершаемого по каналу. CompositeChannelCredentials объединяет ChannelCredentials и CallCredentials для создания нового ChannelCredentials. В результате при каждом вызове, совершаемом по каналу, будут отправляться данные аутентификации, связанные с составным CallCredentials.

Например, вы можете создать ChannelCredentials из SslCredentials и AccessTokenCredentials. В результате при применении к Channel будет отправляться соответствующий токен доступа для каждого вызова по этому каналу.

Отдельные CallCredentials также могут быть составлены с помощью CompositeCallCredentials. Полученный CallCredentials при использовании в вызове инициирует отправку данных аутентификации, связанных с двумя CallCredentials.

Использование клиентского протокола SSL / TLS

Теперь давайте посмотрим, как Credentials работает с одним из наших поддерживаемых механизмов аутентификации. Это самый простой сценарий аутентификации, при котором клиент просто хочет пройти аутентификацию на сервере и зашифровать все данные. Пример написан на C++, но API одинаков для всех языков: вы можете посмотреть, как включить SSL/TLS на других языках, в разделе «Примеры» ниже.

*// Create a default SSL ChannelCredentials object.*

**auto** channel\_creds = grpc::SslCredentials(grpc::SslCredentialsOptions());

*// Create a channel using the credentials created in the previous step.*

**auto** channel = grpc::CreateChannel(server\_name, channel\_creds);

*// Create a stub on the channel.*

std::unique\_ptr<Greeter::Stub> stub(Greeter::NewStub(channel));

*// Make actual RPC calls on the stub.*

grpc::Status s = stub->sayHello(&context, \*request, response);

Для более сложных случаев использования, таких как изменение корневого центра сертификации или использование клиентских сертификатов, соответствующие параметры можно задать в параметре SslCredentialsOptions переданном в метод factory.

Примечание

В системах, не поддерживающих POSIX (например, в Windows), необходимо указать корневые сертификаты в SslCredentialsOptions, поскольку настройки по умолчанию применяются только к файловым системам POSIX.

Использование аутентификации на основе токена OAuth

Протокол OAuth 2.0 — это отраслевой стандарт авторизации. Он позволяет веб-сайтам или приложениям получать ограниченный доступ к учётным записям пользователей с помощью токенов OAuth.

gRPC предлагает набор простых API для интеграции OAuth 2.0 в приложения, упрощая аутентификацию.

At a high level, using OAuth token-based authentication includes 3 steps:

1. Получите или сгенерируйте токен OAuth на стороне клиента.
   * Вы можете сгенерировать токены, относящиеся к Google, следуя приведенным ниже инструкциям.
2. Создайте учетные данные с помощью токена OAuth.
   * Токен OAuth всегда является частью учётных данных для каждого вызова. Вы также можете прикрепить учётные данные для каждого вызова к учётным данным для некоторых каналов.
   * Токен будет отправлен на сервер, как правило, в составе заголовка HTTP Authorization.
3. Серверная сторона проверяет токен.
   * В большинстве реализаций проверка выполняется с помощью перехватчика на стороне сервера.

Подробные инструкции по использованию токенов OAuth на разных языках приведены в наших примерах ниже.

Использование аутентификации на основе токенов Google

Приложения gRPC могут использовать простой API для создания учётных данных, которые подходят для аутентификации в Google в различных сценариях развёртывания. Опять же, наш пример написан на C++, но вы можете найти примеры на других языках в разделе «Примеры».

**auto** creds = grpc::GoogleDefaultCredentials();

*// Create a channel, stub and make RPC calls (same as in the previous example)*

**auto** channel = grpc::CreateChannel(server\_name, creds);

std::unique\_ptr<Greeter::Stub> stub(Greeter::NewStub(channel));

grpc::Status s = stub->sayHello(&context, \*request, response);

Этот объект учётных данных канала работает как для приложений, использующих учётные записи служб, так и для приложений, работающих в Google Compute Engine (GCE). В первом случае закрытые ключи учётной записи службы загружаются из файла, указанного в переменной среды GOOGLE\_APPLICATION\_CREDENTIALS. Ключи используются для создания токенов-носителей, которые прикрепляются к каждому исходящему RPC в соответствующем канале.

Для приложений, работающих в GCE, во время настройки виртуальной машины можно настроить учетную запись службы по умолчанию и соответствующие области OAuth2. Во время выполнения эти учетные данные используются для взаимодействия с системами аутентификации для получения токенов доступа OAuth2 и прикрепляются к каждому исходящему RPC-запросу по соответствующему каналу.

Расширение gRPC для поддержки других механизмов аутентификации

API-интерфейс плагина Credentials позволяет разработчикам подключать собственные типы учётных данных. Он состоит из:

* Абстрактный класс MetadataCredentialsPlugin содержит чистый виртуальный метод GetMetadata , который должен быть реализован в подклассе, созданном разработчиком.
* MetadataCredentialsFromPluginФункция, которая создает CallCredentials из MetadataCredentialsPlugin.

Вот пример простого плагина для аутентификации, который устанавливает билет аутентификации в пользовательском заголовке.

**class** **MyCustomAuthenticator** : **public** grpc::MetadataCredentialsPlugin {

**public**:

MyCustomAuthenticator(**const** grpc::string& ticket) : ticket\_(ticket) {}

grpc::Status GetMetadata(

grpc::string\_ref service\_url, grpc::string\_ref method\_name,

**const** grpc::AuthContext& channel\_auth\_context,

std::multimap<grpc::string, grpc::string>\* metadata) **override** {

metadata->insert(std::make\_pair("x-custom-auth-ticket", ticket\_));

**return** grpc::Status::OK;

}

**private**:

grpc::string ticket\_;

};

**auto** call\_creds = grpc::MetadataCredentialsFromPlugin(

std::unique\_ptr<grpc::MetadataCredentialsPlugin>(

**new** MyCustomAuthenticator("super-secret-ticket")));

Более глубокая интеграция может быть достигнута путём подключения учётных данных gRPC на базовом уровне. Внутренняя структура gRPC также позволяет заменить SSL/TLS другими механизмами шифрования.

Языковые руководства и примеры

Эти механизмы аутентификации будут доступны на всех поддерживаемых gRPC языках. В следующей таблице приведены ссылки на примеры, демонстрирующие аутентификацию и авторизацию на разных языках.

ЯзыкПримерДокументацияC++N/AN/AВпередПерейти к ПримеруПерейти к документацииJavaПример Java TLS (Пример Java ATLS)Документация JavaПитонПример PythonДокументация по Python

Языковые руководства и примеры аутентификации на основе токенов OAuth

В следующей таблице приведены ссылки на примеры, демонстрирующие аутентификацию и авторизацию на основе токенов OAuth на разных языках.

ЯзыкПримерДокументацияC++N/AN/AВпередПример перехода к OAuthПерейти к документации OAuthJavaПример Java OAuthДокументация Java OAuthПитонПример Python OAuthДокументация Python OAuth

Дополнительные примеры

В следующих разделах показано, как функции аутентификации и авторизации, описанные выше, выглядят на других языках, не перечисленных выше.

Рубин

Базовый вариант - без шифрования или аутентификации

stub = Helloworld::Greeter::Stub.new('localhost:50051', :this\_channel\_is\_insecure)

...

С аутентификацией сервера SSL / TLS

creds = GRPC::Core::ChannelCredentials.new(load\_certs) *# load\_certs typically loads a CA roots file*

stub = Helloworld::Greeter::Stub.new('myservice.example.com', creds)

Аутентификация с помощью Google

require 'googleauth' *# from http://www.rubydoc.info/gems/googleauth/0.1.0*

...

ssl\_creds = GRPC::Core::ChannelCredentials.new(load\_certs) *# load\_certs typically loads a CA roots file*

authentication = Google::Auth.get\_application\_default()

call\_creds = GRPC::Core::CallCredentials.new(authentication.updater\_proc)

combined\_creds = ssl\_creds.compose(call\_creds)

stub = Helloworld::Greeter::Stub.new('greeter.googleapis.com', combined\_creds)

Node.js

Базовый вариант - Отсутствие шифрования /аутентификации

**var** stub = **new** helloworld.Greeter('localhost:50051', grpc.credentials.createInsecure());

С аутентификацией сервера SSL / TLS

**const** root\_cert = fs.readFileSync('path/to/root-cert');

**const** ssl\_creds = grpc.credentials.createSsl(root\_cert);

**const** stub = **new** helloworld.Greeter('myservice.example.com', ssl\_creds);

Аутентификация с помощью Google

*// Authenticating with Google*

**var** GoogleAuth = require('google-auth-library'); *// from https://www.npmjs.com/package/google-auth-library*

...

**var** ssl\_creds = grpc.credentials.createSsl(root\_certs);

(**new** GoogleAuth()).getApplicationDefault(**function**(err, auth) {

**var** call\_creds = grpc.credentials.createFromGoogleCredential(auth);

**var** combined\_creds = grpc.credentials.combineChannelCredentials(ssl\_creds, call\_creds);

**var** stub = **new** helloworld.Greeter('greeter.googleapis.com', combined\_credentials);

});

Аутентификация в Google с использованием токена OAuth2 (устаревший подход)

**var** GoogleAuth = require('google-auth-library'); *// from https://www.npmjs.com/package/google-auth-library*

...

**var** ssl\_creds = grpc.Credentials.createSsl(root\_certs); *// load\_certs typically loads a CA roots file*

**var** scope = 'https://www.googleapis.com/auth/grpc-testing';

(**new** GoogleAuth()).getApplicationDefault(**function**(err, auth) {

**if** (auth.createScopeRequired()) {

auth = auth.createScoped(scope);

}

**var** call\_creds = grpc.credentials.createFromGoogleCredential(auth);

**var** combined\_creds = grpc.credentials.combineChannelCredentials(ssl\_creds, call\_creds);

**var** stub = **new** helloworld.Greeter('greeter.googleapis.com', combined\_credentials);

});

С аутентификацией сервера SSL / TLS и пользовательским заголовком с токеном

**const** rootCert = fs.readFileSync('path/to/root-cert');

**const** channelCreds = grpc.credentials.createSsl(rootCert);

**const** metaCallback = (\_params, callback) => {

**const** meta = **new** grpc.Metadata();

meta.add('custom-auth-header', 'token');

callback(**null**, meta);

}

**const** callCreds = grpc.credentials.createFromMetadataGenerator(metaCallback);

**const** combCreds = grpc.credentials.combineChannelCredentials(channelCreds, callCreds);

**const** stub = **new** helloworld.Greeter('myservice.example.com', combCreds);

PHP

Базовый вариант - Отсутствие шифрования /авторизации

$client = **new** helloworld\GreeterClient('localhost:50051', [

'credentials' => Grpc\ChannelCredentials::createInsecure(),

]);

С аутентификацией сервера SSL / TLS

$client = **new** helloworld\GreeterClient('myservice.example.com', [

'credentials' => Grpc\ChannelCredentials::createSsl(file\_get\_contents('roots.pem')),

]);

Аутентификация с помощью Google

**function** updateAuthMetadataCallback($context)

{

$auth\_credentials = ApplicationDefaultCredentials::getCredentials();

**return** $auth\_credentials->updateMetadata($metadata = [], $context->service\_url);

}

$channel\_credentials = Grpc\ChannelCredentials::createComposite(

Grpc\ChannelCredentials::createSsl(file\_get\_contents('roots.pem')),

Grpc\CallCredentials::createFromPlugin('updateAuthMetadataCallback')

);

$opts = [

'credentials' => $channel\_credentials

];

$client = **new** helloworld\GreeterClient('greeter.googleapis.com', $opts);

Аутентификация в Google с использованием токена OAuth2 (устаревший подход)

*// the environment variable "GOOGLE\_APPLICATION\_CREDENTIALS" needs to be set*

$scope = "https://www.googleapis.com/auth/grpc-testing";

$auth = Google\Auth\ApplicationDefaultCredentials::getCredentials($scope);

$opts = [

'credentials' => Grpc\Credentials::createSsl(file\_get\_contents('roots.pem'));

'update\_metadata' => $auth->getUpdateMetadataFunc(),

];

$client = **new** helloworld\GreeterClient('greeter.googleapis.com', $opts);

Дротик

Базовый вариант - без шифрования или аутентификации

**final** channel = **new** ClientChannel('localhost',

port: 50051,

options: **const** ChannelOptions(

credentials: **const** ChannelCredentials.insecure()));

**final** stub = **new** GreeterClient(channel);

С аутентификацией сервера SSL / TLS

*// Load a custom roots file.*

**final** trustedRoot = **new** File('roots.pem').readAsBytesSync();

**final** channelCredentials =

**new** ChannelCredentials.secure(certificates: trustedRoot);

**final** channelOptions = **new** ChannelOptions(credentials: channelCredentials);

**final** channel = **new** ClientChannel('myservice.example.com',

options: channelOptions);

**final** client = **new** GreeterClient(channel);

Аутентификация с помощью Google

*// Uses publicly trusted roots by default.*

**final** channel = **new** ClientChannel('greeter.googleapis.com');

**final** serviceAccountJson =

**new** File('service-account.json').readAsStringSync();

**final** credentials = **new** JwtServiceAccountAuthenticator(serviceAccountJson);

**final** client =

**new** GreeterClient(channel, options: credentials.toCallOptions);

Проверка подлинности одного вызова RPC

*// Uses publicly trusted roots by default.*

**final** channel = **new** ClientChannel('greeter.googleapis.com');

**final** client = **new** GreeterClient(channel);

...

**final** serviceAccountJson =

**new** File('service-account.json').readAsStringSync();

**final** credentials = **new** JwtServiceAccountAuthenticator(serviceAccountJson);

**final** response =

**await** client.sayHello(request, options: credentials.toCallOptions);

Последнее изменение: 12 января 2024 г.: добавлена аутентификация на основе токенов OAuth (#1245) (d90b5f9)