Authentication

This document provides an overview of gRPC authentication, including our built-in supported auth mechanisms, how to plug in your own authentication systems, and examples of how to use gRPC auth in our supported languages.

Overview

gRPC is designed to work with a variety of authentication mechanisms, making it easy to safely use gRPC to talk to other systems. You can use our supported mechanisms - SSL/TLS with or without Google token-based authentication - or you can plug in your own authentication system by extending our provided code.

gRPC also provides a simple authentication API that lets you provide all the necessary authentication information as `Credentials` when creating a channel or making a call.

Supported auth mechanisms

The following authentication mechanisms are built-in to gRPC:

- **SSL/TLS**: gRPC has SSL/TLS integration and promotes the use of SSL/TLS to authenticate the server, and to encrypt all the data exchanged between the client and the server. Optional mechanisms are available for clients to provide certificates for mutual authentication.
- **Token-based authentication with Google**: gRPC provides a generic mechanism (described below) to attach metadata based credentials to requests and responses. Additional support for acquiring access tokens (typically OAuth2 tokens) while accessing Google APIs through gRPC is provided for certain auth flows: you can see how this works in our code examples below. In general this mechanism must be used *as well as* SSL/TLS on the channel - Google will not allow connections without SSL/TLS, and most gRPC language implementations will not let you send credentials on an unencrypted channel.

WARNING: Google credentials should only be used to connect to Google services. Sending a Google issued OAuth2 token to a non-Google service could result in this token being stolen and used to impersonate the client to Google services.

Authentication API

gRPC provides a simple authentication API based around the unified concept of Credentials objects, which can be used when creating an entire gRPC channel or an individual call.

Credential types

Credentials can be of two types:

- **Channel credentials**, which are attached to a `Channel`, such as SSL credentials.
- **Call credentials**, which are attached to a call (or `ClientContext` in C++).

You can also combine these in a `CompositeChannelCredentials`, allowing you to specify, for example, SSL details for the channel along with call credentials for each call made on the channel. A `CompositeChannelCredentials` associates a `ChannelCredentials` and a `CallCredentials` to create a new `ChannelCredentials`. The result will send the authentication data associated with the composed `CallCredentials` with every call made on the channel.

For example, you could create a `ChannelCredentials` from an `SslCredentials` and an `AccessTokenCredentials`. The result when applied to a `Channel` would send the appropriate access token for each call on this channel.

Individual `CallCredentials` can also be composed using `CompositeCallCredentials`. The resulting `CallCredentials` when used in a call will trigger the sending of the authentication data associated with the two `CallCredentials`.

Using client-side SSL/TLS

Now let's look at how `Credentials` work with one of our supported auth mechanisms. This is the simplest authentication scenario, where a client just wants to authenticate the server and encrypt all data. The example is in C++, but the API is similar for all languages: you can see how to enable SSL/TLS in more languages in our Examples section below.

```
// 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);
```

For advanced use cases such as modifying the root CA or using client certs, the corresponding options can be set in the `SslCredentialsOptions` parameter passed to the factory method.

Using Google token-based authentication

gRPC applications can use a simple API to create a credential that works for authentication with Google in various deployment scenarios. Again, our example is in C++ but you can find examples in other languages in our Examples section.

```
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);
```

This channel credentials object works for applications using Service Accounts as well as for applications running in Google Compute Engine (GCE). In the former case, the service account's private keys are loaded from the file named in the environment variable `GOOGLE_APPLICATION_CREDENTIALS`. The keys are used to generate bearer tokens that are attached to each outgoing RPC on the corresponding channel.

For applications running in GCE, a default service account and corresponding OAuth2 scopes can be configured during VM setup. At run-time, this credential handles communication with the authentication systems to obtain OAuth2 access tokens and attaches them to each outgoing RPC on the corresponding channel.

Extending gRPC to support other authentication mechanisms

The Credentials plugin API allows developers to plug in their own type of credentials. This consists of:

- The `MetadataCredentialsPlugin` abstract class, which contains the pure virtual `GetMetadata` method that needs to be implemented by a sub-class created by the developer.
- The `MetadataCredentialsFromPlugin` function, which creates a `CallCredentials` from the `MetadataCredentialsPlugin`.

Here is example of a simple credentials plugin which sets an authentication ticket in a custom header.

```
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")));
```

A deeper integration can be achieved by plugging in a gRPC credentials implementation at the core level. gRPC internals also allow switching out SSL/TLS with other encryption mechanisms.

Examples

These authentication mechanisms will be available in all gRPC's supported languages. The following sections demonstrate how authentication and authorization features described above appear in each language: more languages are coming soon.

Go

Base case - no encryption or authentication

Client:

```
conn, _ := grpc.Dial("localhost:50051",
grpc.WithInsecure())
// error handling omitted
client := pb.NewGreeterClient(conn)
// ...
```

Server:

```
s := grpc.NewServer()
lis, _ := net.Listen("tcp", "localhost:50051")
// error handling omitted
s.Serve(lis)
```

With server authentication SSL/TLS

Client:

```
creds, _ := credentials.NewClientTLSFromFile(certFile, "")
conn, _ := grpc.Dial("localhost:50051",
grpc.WithTransportCredentials(creds))
// error handling omitted
client := pb.NewGreeterClient(conn)
// ...
```

Server:

```
creds, _ := credentials.NewServerTLSFromFile(certFile,
keyFile)
s := grpc.NewServer(grpc.Creds(creds))
lis, _ := net.Listen("tcp", "localhost:50051")
// error handling omitted
s.Serve(lis)
```

Authenticate with Google

```
pool, _ := x509.SystemCertPool()
// error handling omitted
creds := credentials.NewClientTLSFromCert(pool, "")
perRPC, _ := oauth.NewServiceAccountFromFile("service-
account.json", scope)
conn, _ := grpc.Dial(
    "greeter.googleapis.com",
    grpc.WithTransportCredentials(creds),
    grpc.WithPerRPCCredentials(perRPC),
)
// error handling omitted
client := pb.NewGreeterClient(conn)
// ...
```

Ruby

Base case - no encryption or authentication

```
stub = Helloworld::Greeter::Stub.new('localhost:50051',
:this_channel_is_insecure)
...
```

With server authentication 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)
```

Authenticate with 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)
```

C++

Base case - no encryption or authentication

```
auto channel = grpc::CreateChannel("localhost:50051",
InsecureChannelCredentials());
std::unique_ptr<Greeter::Stub>
stub(Greeter::NewStub(channel));
...
```

With server authentication SSL/TLS

```
auto channel_creds =
grpc::SslCredentials(grpc::SslCredentialsOptions());
auto channel = grpc::CreateChannel("myservice.example.com",
channel_creds);
std::unique_ptr<Greeter::Stub>
stub(Greeter::NewStub(channel));
...
```

Authenticate with Google

```
auto creds = grpc::GoogleDefaultCredentials();
auto channel =
grpc::CreateChannel("greeter.googleapis.com", creds);
std::unique_ptr<Greeter::Stub>
stub(Greeter::NewStub(channel));
...
```

C#

Base case - no encryption or authentication

```
var channel = new Channel("localhost:50051",
ChannelCredentials.Insecure);
var client = new Greeter.GreeterClient(channel);
...
```

With server authentication SSL/TLS

```
var channelCredentials = new
SslCredentials(File.ReadAllText("roots.pem"));  // Load a
custom roots file.
var channel = new Channel("myservice.example.com",
channelCredentials);
var client = new Greeter.GreeterClient(channel);
```

Authenticate with Google

```
using Grpc.Auth;  // from Grpc.Auth NuGet package
...
// Loads Google Application Default Credentials with
publicly trusted roots.
var channelCredentials = await
GoogleGrpcCredentials.GetApplicationDefaultAsync();

var channel = new Channel("greeter.googleapis.com",
channelCredentials);
var client = new Greeter.GreeterClient(channel);
...
```

Authenticate a single RPC call

```
var channel = new Channel("greeter.googleapis.com", new
SslCredentials());  // Use publicly trusted roots.
var client = new Greeter.GreeterClient(channel);
...
var googleCredential = await
GoogleCredential.GetApplicationDefaultAsync();
var result = client.SayHello(request, new
CallOptions(credentials:
googleCredential.ToCallCredentials()));
...
```

Python

Base case - No encryption or authentication

```
import grpc
import helloworld_pb2

channel = grpc.insecure_channel('localhost:50051')
stub = helloworld_pb2.GreeterStub(channel)
```

With server authentication SSL/TLS

Client:

```
import grpc
import helloworld_pb2

with open('roots.pem', 'rb') as f:
    creds = grpc.ssl_channel_credentials(f.read())
channel = grpc.secure_channel('myservice.example.com:443',
creds)
stub = helloworld_pb2.GreeterStub(channel)
```

Server:

```
import grpc
import helloworld_pb2
from concurrent import futures

server =
grpc.server(futures.ThreadPoolExecutor(max_workers=10))
with open('key.pem', 'rb') as f:
    private_key = f.read()
with open('chain.pem', 'rb') as f:
    certificate_chain = f.read()
server_credentials = grpc.ssl_server_credentials( (
(private_key, certificate_chain), ) )
# Adding GreeterServicer to server omitted
server.add_secure_port('myservice.example.com:443',
server_credentials)
server.start()
# Server sleep omitted
```

Authenticate with Google using a JWT

```
import grpc
import helloworld_pb2

from google import auth as google_auth
from google.auth import jwt as google_auth_jwt
from google.auth.transport import grpc as
google_auth_transport_grpc

credentials, _ = google_auth.default()
jwt_creds =
google_auth_jwt.OnDemandCredentials.from_signing_credentials(

    credentials)
channel =
google_auth_transport_grpc.secure_authorized_channel(
    jwt_creds, None, 'greeter.googleapis.com:443')
stub = helloworld_pb2.GreeterStub(channel)
```

Authenticate with Google using an Oauth2 token

```
import grpc
import helloworld_pb2

from google import auth as google_auth
from google.auth.transport import grpc as
google_auth_transport_grpc
from google.auth.transport import requests as
google_auth_transport_requests

credentials, _ = google_auth.default(scopes=(scope,))
request = google_auth_transport_requests.Request()
channel =
google_auth_transport_grpc.secure_authorized_channel(
    credentials, request, 'greeter.googleapis.com:443')
stub = helloworld_pb2.GreeterStub(channel)
```

Java

Base case - no encryption or authentication

```
ManagedChannel channel =
ManagedChannelBuilder.forAddress("localhost", 50051)
    .usePlaintext(true)
    .build();
GreeterGrpc.GreeterStub stub =
GreeterGrpc.newStub(channel);
```

With server authentication SSL/TLS

In Java we recommend that you use OpenSSL when using gRPC over TLS. You can find details about installing and using OpenSSL and other required libraries for both Android and non-Android Java in the gRPC Java Security documentation.

To enable TLS on a server, a certificate chain and private key need to be specified in PEM format. Such private key should not be using a password. The order of certificates in the chain matters: more specifically, the certificate at the top has to be the host CA, while the one at the very bottom has to be the root CA. The standard TLS port is 443, but we use 8443 below to avoid needing extra permissions from the OS.

```
Server server = ServerBuilder.forPort(8443)
    // Enable TLS
    .useTransportSecurity(certChainFile, privateKeyFile)

.addService(TestServiceGrpc.bindService(serviceImplementation)

    .build();
server.start();
```

If the issuing certificate authority is not known to the client then a properly configured `SslContext` or `SSLSocketFactory` should be provided to the `NettyChannelBuilder` or `OkHttpChannelBuilder`, respectively.

On the client side, server authentication with SSL/TLS looks like this:

```
// With server authentication SSL/TLS
ManagedChannel channel =
ManagedChannelBuilder.forAddress("myservice.example.com",
443)
    .build();
GreeterGrpc.GreeterStub stub =
GreeterGrpc.newStub(channel);

// With server authentication SSL/TLS; custom CA root
certificates; not on Android
ManagedChannel channel =
NettyChannelBuilder.forAddress("myservice.example.com",
443)

.sslContext(GrpcSslContexts.forClient().trustManager(new
File("roots.pem")).build())
    .build();
GreeterGrpc.GreeterStub stub =
GreeterGrpc.newStub(channel);
```

Authenticate with Google

The following code snippet shows how you can call the Google Cloud PubSub API using gRPC with a service account. The credentials are loaded from a key stored in a well-known location or by detecting that the application is running in an environment that can provide one automatically, e.g. Google Compute Engine. While this example is specific to Google and its services, similar patterns can be followed for other service providers.

```
GoogleCredentials creds =
GoogleCredentials.getApplicationDefault();
ManagedChannel channel =
ManagedChannelBuilder.forTarget("greeter.googleapis.com")
    .build();
GreeterGrpc.GreeterStub stub = GreeterGrpc.newStub(channel)
    .withCallCredentials(MoreCallCredentials.from(creds));
```

Node.js

Base case - No encryption/authentication

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

With server authentication SSL/TLS

```
var ssl_creds = grpc.credentials.createSsl(root_certs);
var stub = new helloworld.Greeter('myservice.example.com',
ssl_creds);
```

Authenticate with 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);
});
```

Authenticate with Google using Oauth2 token (legacy approach)

```
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);
});
```

PHP

Base case - No encryption/authorization

```
$client = new helloworld\GreeterClient('localhost:50051', [
    'credentials' =>
Grpc\ChannelCredentials::createInsecure(),
]);
...
```

Authenticate with 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.pe



Grpc\CallCredentials::createFromPlugin('updateAuthMetadataCal

);
$opts = [
  'credentials' => $channel_credentials
];
$client = new
helloworld\GreeterClient('greeter.googleapis.com', $opts);
```

Authenticate with Google using Oauth2 token (legacy approach)

```
// the environment variable
"GOOGLE_APPLICATION_CREDENTIALS" needs to be set
$scope = "https://www.googleapis.com/auth/grpc-testing";
$auth =
Google\Auth\ApplicationDefaultCredentials::getCredentials($sc

$opts = [
  'credentials' =>
Grpc\Credentials::createSsl(file_get_contents('roots.pem'));

  'update_metadata' => $auth->getUpdateMetadataFunc(),
];
$client = new
helloworld\GreeterClient('greeter.googleapis.com', $opts);
```

Dart

Base case - no encryption or authentication

```
final channel = new ClientChannel('localhost',
      port: 50051,
      options: const ChannelOptions(
          credentials: const
ChannelCredentials.insecure()));
final stub = new GreeterClient(channel);
```

With server authentication 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);
```

Authenticate with 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);
```

Authenticate a single RPC call

```
// 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);
```