Communication, Authentication and Authorization in CalAmp Telematics Cloud platform

Version 1.5

Contents

Introduction

This document explains the authentication and authorization strategy used in Connect Telematics Cloud ("CTC") platform. It defines the relationships between the Account, Application, User, Role and Permission resources in CTC and how those work together to provide role-based access control (RBAC) in a multi-tenant, multi-application platform.

Conventions

This section lists out the written conventions followed in the document.

`User` or `user` - is a resource type.
`User::` or `User::*` - represents any instance of `User`
`User::Joe` - is a specific instance of `User` with username Joe.
`User::11` - is a specific instance of `User` with id:11
`Users` - represents a collection of `User::`
`User::Joe.Roles` - represents the collection of `Role::` assigned to `User::Joe`
`User::Joe.Roles::23` - represents `Role::23` assigned to `User::Joe`

Communication

All communication with the CTC APIs is via HTTP over TLSv1.2 (`https` protocol). Plain `http` is not supported.

Only authenticated users are allowed to make API calls in CTC.

`JSON` is used as the data exchange format by CTC APIs.

The HTTP header `Content-type: application/json;charset=UTF-8` is required for all requests that send data to CTC APIs, except `login` (See the Login section below)

Authentication

CTC uses persistent token based authentication. To receive the persistent token, first a login call needs to be made.

Passwords are stored as a `bcrypt` hash. As such, only way to recover password is using the "Forgot Password" feature/api. For this feature, a valid email address is required.

Login

URL: `https://<host:port>/connect/services/login?useAuthToken=true`
HTTP method: `POST`
HTTP headers:
`Content-Type: application/x-www-form-urlencoded`
`calamp-services-app`: <application key supplied>
Request body: `username`=<given username>&`password`=<password>

Example:
HTTP:

```
POST /connect/services/login?useAuthToken=true HTTP/1.1
Host: dev.connect.calamp.com
```

```
Content-Type: application/x-www-form-urlencoded
calamp-services-app: xxxxxxxx-yyyy-NNNN-bbbb-aaaaaaaaaaaa
Body: password=<password>&username=<username>
```

cURL:

```
curl -X POST \
'https://<host:port>/connect/services/login?useAuthToken=true' \
-H 'calamp-services-app: xxxxxxxx-yyyy-NNNN-bbbb-aaaaaaaaaaaa' \
-H 'content-type: application/x-www-form-urlencoded;charset=UTF-8' \
--data 'username=<username>&password=<password>'
```

On successful login, the service will return the `authToken` in HTTP `Set-Cookie` header along with other relevant headers.

Example response headers:

```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 30 Oct 2018 13:30:30 GMT
Content-Length: 0
Connection: keep-alive
Location: https://connect.calamp.com/connect/services/users/64
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET,PUT,POST,DELETE,HEAD,OPTIONS
Access-Control-Allow-Origin: https://connect.calamp.com
Access-Control-Max-Age: 600
Set-Cookie: authToken=<value>;
      Expires=Tue, 30-Oct-2018 14:30:30 GMT; Path=/; Secure; HttpOnly
Set-Cookie: authToken=<value>;
      Expires=Tue, 30-Oct-2018 14:30:30 GMT; Domain=.connect.calamp.com;
      Path=/; Secure; HttpOnly
Via: 1.1 connect.calamp.com
Strict-Transport-Security: max-age=2592000; includeSubDomains
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
```

A user can have multiple authenticated sessions active at the same time, each with their own `authToken`. Each session is independent of each other.

OIDC tokens.

With introduction of OAuth2.0 flows and OIDC support in CTC, the `authToken` is changing from a session token to an access token. The `authToken` is a JWT with custom claims encoded in it that identify the authorizations for the user. As such, the token will be carrying lot more information that can help decentralize the authorization checks and thus the size of the token will increase.

With OIDC access token, `authToken`

1. size can be upto 2048 bytes,
2. will be valid for 24 hours, and
3. will not auto-refresh.

After 24 hours, the token will expire and the user MUST make a fresh `login` call to obtain a new token.

Logout

URL: https://<host:port>/connect/services/logout
HTTP method: POST
HTTP headers:

Cookie: authToken=<value received in response to login call>
calamp-services-app: <application key supplied>

Example

HTTP:

```
POST /connect/services/logout HTTP/1.1
Host: connect.calamp.com
calamp-services-app: xxxxxxxx-yyyy-NNNN-bbbb-aaaaaaaaaaaa
```

cURL:

```
curl -X POST 'https://connect.calamp.com/connect/services/logout' \
-H 'calamp-services-app: xxxxxxxx-yyyy-NNNN-bbbb-aaaaaaaaaaaa'
```

Sample response:

```
access-control-allow-credentials: true
access-control-allow-methods: GET,PUT,POST,DELETE,HEAD,OPTIONS
access-control-allow-origin: https://connect.calamp.com
access-control-max-age: 600
content-length: 0
date: Wed, 03 Apr 2019 21:41:48 GMT
location: https:
server: Apache-Coyote/1.1
set-cookie: authToken=""; Domain=.connect.calamp.com; Expires=Thu, 01-
     Jan-1970 00:00:10 GMT; Path=/; Secure; HttpOnly
set-cookie: authToken=""; Expires=Thu, 01-Jan-1970 00:00:10 GMT;
     Path=/; Secure; HttpOnly
status: 302
strict-transport-security: max-age=2592000; includeSubDomains
via: 1.1 dev.connect.calamp.com
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
x-xss-protection: 1; mode=block
```

When the `logout` endpoint is invoked, all tokens of the authenticated user are invalidated thereby ending all sessions for that user.

API Calls

Only authenticated users are allowed to make API calls in CTC. To establish the credentials of the user making the call, it MUST include the following two HTTP headers:

1. Cookie: authToken=<value received in response to login call>
2. calamp-services-app: <the application key provided when signing up to develop CTC client applications>

Authorization

CTC uses Role Based Access Control (RBAC) approach with a mixture of Flat RBAC and Symmetric RBAC layered on top of the principle of least privilege. With RBAC, users are assigned `Roles` that have `Permissions`. In addition, CTC allows for new `Role::` to be created as needed by the customer to support their authorization requirements. Thus, while some predefined roles are available, the customer is neither constrained to just using those, nor dependent on CTC development to create new ones.

Since CTC is a multi-application platform, a `Role::` is specific to an `Application::`.

Being a multi-tenant platform, CTC automatically restricts all actions by an authenticated `User::` to only resources in the hierarchy of `Accounts` the user has access to.

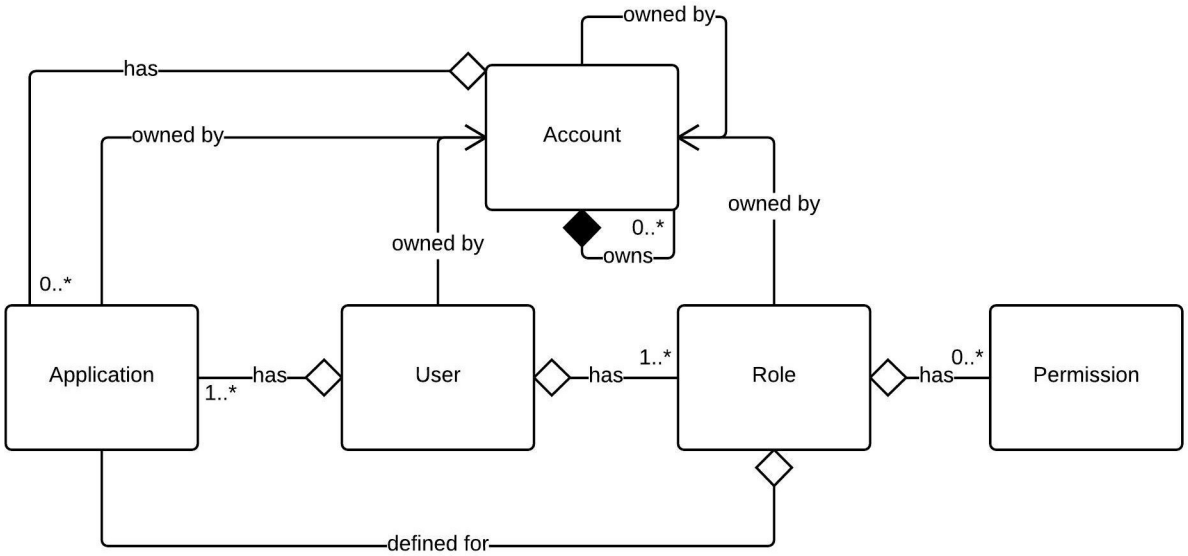
Each resource in CTC has 4 basic actions that can be performed - `create`, `read`, `update` and `delete`.

A `Permission` is then a combination of resource type and an action. Example: `account-read`, `account-update`, `device-delete`, and so on.

What actions, on which resources, the `User::` is authorized to undertake, is defined by

1. the `Permissions` assigned to the `User::` for given `Application::` and
2. the `User::`'s position in the `Account` hierarchy.

Design



Glossary

A volatile resource in CTC is one whose state can be changed by actors via the CTC APIs.

`Account` is the owner of all volatile resources in CTC. It is itself a volatile resource. An `Account::` has a parent account and zero or more child accounts. All volatile resources in CTC have one, and only one, `Account::` as direct owner.

`Application` is a volatile resource representing any client of the CTC services. It uniquely identifies the client. An application could be CalAmp internal, implemented and controlled by CalAmp developer teams, or a third-party application that CalAmp has no control over.

`User` is a volatile resource that represents any actor that can authenticate against CTC.

`Role` a volatile resource representing a collection of `Permission::` resources, for a given `Application::`.

`Permission` is a non-volatile resource that represents what actions an authenticated `User::` is authorized to take in CTC.

Additional Considerations

An `Application::` belongs to only one `Account::`, but can be assigned to multiple accounts.
Though an `Application::` is owned by only one `Account::`, it can be associated with multiple accounts, if each of those accounts is a child of the `Account::`. Thus, an `Account::` can have `Applications` directly as resources it controls and also be assigned `Applications` from its **`direct`** parent `Account::`.

A `User::` may have multiple roles. The `Permissions` for a `User::` are then a union of all the `Permissions` in all the `Roles` assigned to the `User::`, for the `Application::` being used to access CTC APIs.

A `User::` with no `Roles` assigned to it will be able to authenticate against CTC via any `Application::` but, will not be authorized to perform any actions.

Recommendations

authToken

Since the token is valid for a certain duration of time, there is no need to make repeated `login` calls and obtain a new `authToken` for every API call to CTC. Instead, it is more efficient to call `login` under the following conditions:

1. upon receiving a `401 Unauthorized` response from the API or,
2. if it is known beforehand that the client application will need a new token
    a. This could be either on restart of the client application that caches the `authToken` only in memory
    b. The client application has not communicated with CTC API for more than the validity period of the token.

Additionally, for multi-threaded clients, while there is no restriction on each thread obtaining and using its own `authToken`, and certainly there could be cases where this is warranted, it is more efficient for the process to obtain one token and have it be used by all its threads when communicating with CTC API.

When the client receives a `401 Unauthorized` response from CTC API, that token is invalid and any more CTC calls using that token will always result in the same `401 Unauthorized` response.
Clients MUST obtain a new `authToken` via a `login` call in this scenario.

Supplying credentials

User credentials SHOULD not be supplied in the URL. Instead, for the `login` call, the credentials must be in the request body.
Similarly, the `application key` should be presented to CTC API in the HTTP header – `calamp-services-app` and not in the URL. See the details in Login section

Document History

Version	Date	Author	Description
1.0	12/13/14	CTC Engineering	Initial release.
1.1	10/17/2017	CTC Engineering	Updated User-Role association constraint.
1.2	10/24/2018	CTC Engineering	Added Conventions, Authentication and Authorization sections.
1.3	04/03/2019	CTC Engineering	Added a Recommendations section
1.4	07/23/2021	CTC Engineering	Added OIDC access token details.
1.5	05/20/2025	CTC Engineering	Updated token expiry and changes to login API