ACLs

C1 Core uses ACLs to check access from untrusted sources. Untrusted sources are:

- Packets originating from C1 Proxy and sent by Homegear Cloudconnect.
- Packets originating from a user's device or browser.

Trusted sources include all Sensaru Cloud modules except C1 Proxy. ACLs are not checked for packets originating from trusted sources.

There are two types of ACLs:

1. **Edge client ACLs**
2. **User ACLs**

Homegear Cloudconnect instances have **edge client ACLs** assigned. Users have **user ACLs** assigned.

ACL structure and storage location

There are two different types of ACLs:

1. **User ACLs**
2. and **edge client ACLs**.

ACLs can only be assigned to groups. They cannot be assigned directly to users or edge instances. A user or edge instance can have multiple groups assigned.

User ACL

An **user ACL** looks like this:

```
{
  "version": 1,
  "moduleAccess": {
    "c1-device-management": {
      "global": {
        "read": true,
        "write":true,
        "event":true,
        "isAdmin":true
      },
      "rpcMethods": []
    }
  },
```

```
  "restAccess": {
    "/user": ["GET", "POST", "PUT"],
    "/sessions": ["GET", "POST", "PUT"],
    "/test/*": ["GET", "POST", "PUT"],
    "/test/no-access": {
      "GET": false,
      "POST": false,
      "PUT": false
    },
    "/sessions/session": ["GET", "POST", "PUT"]
  },
  "assetAccess": ["6582", "5912.*", "7291.4.2", "51:*", "52:9893.3.2"],
  "roleAccess": [200384, 709839]
}
```

Key	Description
`moduleAccess`	An object with ACLs that are applied when the user tries to access a specific module.
`moduleAccess\<module>\global`	Permissions that are applied globally for the module.
`moduleAccess\<module>\global\read`	Access is allowed to RPC methods that have the `read` flag assigned.
`moduleAccess\<module>\global\write`	Access is allowed to RPC methods that have the `write` flag assigned.
`moduleAccess\<module>\global\event`	Access is allowed to RPC methods that have the `event` flag assigned.
`moduleAccess\<module>\global\isAdmin`	Access is allowed to RPC methods that have the `admin` flag assigned.
`moduleAccess\<module>\rpcMethods`	A list of `rpcMethods` access is allowed to. This is only relevant when access is not granted by one of the `global` flags.
`restAccess`	An object containing the REST paths and HTTP methods that access is allowed to. "/api/v1" is ommited. Wildcards are allowed at any path position. The methods can be an array or an object. In the latter case, permissions can be denied explicitly. This is relevant for example if a path is given access to by a wildcard and access to a sub path should be denied.

Key	Description
`assetAccess`	A list of asset IDs this user has access to. Every entry gives access to exactly the specified asset. `*` can be specified as an wildcard at the last level to give access to all assets of lower levels. E. g. to access all properties of the economic unit "1234", specify `["1234.*"]` (`"*.123"` is not allowed - the wildcard must be the last character). To also include access to the economic unit itself, "1234" must be added as well: `["1234", "1234.*"]`. Please note that `*` (a single asterisk) grants access to all economic units. To grant access to all assets when portfolios are used, add the entry `*:` (asterisk followed by a colon). This grants access to all assets of the business partner (this can be used when no portfolios are used as well).
`roleAccess`	A list of role IDs this user has access to. When empty, the user has access to all roles. Every entry gives access to exactly the specified role.

Differentiating administrative and non-administrative module REST access

Module administrative REST calls should be placed in the subpath `/admin`, e. g. `/api/v1/modules/my-module/admin`. Like this the ACL can differentiate easily between administrative and non-administrative REST calls.

Edge client ACL

An **Edge client ACL** looks like this:

```
{
  "version": 1,
  "moduleAccess": {
    "c1-device-management": {
      "global": {
        "read": true,
        "write":true,
        "event":true,
        "isAdmin":true
      },
      "rpcMethods": []
    }
  }
}
```

Key	Description
`moduleAccess`	An object with ACLs that are applied when the edge client tries to access a specific module. Use `*` for `<module>` to apply the permissions to all modules.
`moduleAccess\<module>\global`	Permissions that are applied globally for the module.
`moduleAccess\<module>\global\read`	Access is allowed to RPC methods that have the `read` flag assigned.
`moduleAccess\<module>\global\write`	Access is allowed to RPC methods that have the `write` flag assigned.
`moduleAccess\<module>\global\event`	Access is allowed to RPC methods that have the `event` flag assigned.
`moduleAccess\<module>\global\isAdmin`	Access is allowed to RPC methods that have the `admin` flag assigned.
`moduleAccess\<module>\rpcMethods`	A list of `rpcMethods` access is allowed to. This is only relevant when access is not granted by one of the `global` flags.

With the flags there are three options ordered by precedence:

1. To set the flag to `false.`
2. To set the flag to `true` .
3. To not specify the flag at all.

The precedence is relevant when a user has multiple groups assigned. When a `flag` is set to `false` , then access is always denied. The `false` can't be overridden by a `true` of another group. When there are no flags at all, access is denied as well.

The global flags are optional. For more granular permissions you can use `rpcMethods` . `rpcMethods` from different groups are `or` 'd.

In the future it might be possible to add `true/false` to `rpcMethods` as well. Additionally it might be possible that modules can specify custom flags.

Associate flags with RPC methods

C1 Core doesn't know the RPC methods of a module. This means the module needs to send its RPC methods and the associated flags to C1 Core. This is done by the `aclInfo` parameter of `registerModule()`. When using C1 Module Proxy, it is provided by the setting `moduleAclInfo`. When using libc1-module you can set it in the client info `struct`. The ACL info structure looks like this:

```
{
  "rpcMethods": {
    "myMethod1": "admin",
    "myMethod2": "admin",
    "myMethod3": "write"
  },
  "version":1
}
```

Key	Description
`rpcMethods`	The object containing the RPC method flags.
`rpcMethods\<method>\admin`	Method requires the `isAdmin` ACL flag to be accessed.
`rpcMethods\<method>\read`	Method requires the `read` ACL flag to be accessed.
`rpcMethods\<method>\write`	Method requires the `write` ACL flag to be accessed.
`rpcMethods\<method>\event`	Method requires the `event` ACL flag to be accessed.

C1 Core methods relevant for ACL checks

We got two possible entry-points of untrusted method calls: Calls coming from users, i. e. all REST calls, and information coming from devices not secured in our data center.

In addition there are semi-trusted sources, like devices within secure areas of system distributors and business partners.

Calls originating from users

There are two types of calls:

1. REST calls with C1 Core as endpoint.
2. REST calls that are translated into RPC requests with modules as endpoints.

For REST calls the ACL specifies paths and methods that access is allowed to. These ACL is checked within `RestServer::packetReceived()`. REST calls that are translated into module RPC requests are routed through `RpcServer::userPreinvoke()` and `RpcServer::userInvoke()`. Here the access to the called module method is checked.

Calls originating from untrusted devices

All calls from untrusted devices are coming from C1 Proxy and routed through `RpcServer::moduleMethod()`, i. e. ACL checks are performed there.

Calls originating from semi-trusted devices

Semi-trusted devices must be assigned to a specific business partner. This makes sure, these devices only have access to data of this one principal.

RpcServer::registerModule

Modules pass information about their RPC methods - including tunneled REST requests - and the required flags to `registerModule()`. The information is stored in database and held within memory.

RpcServer::moduleMethod

1. Checks if parameter 5 (`homeClientId`) is set and the source of the packet is C1 Proxy. When this is the case, the edge ID is read (this information is trustable) and the associated groups and associated users are retrieved from database. The group ACL information for the users and the edge client is then read from database and module access is checked. The checked fields are destination module ID and called method name. Additionally parameter 6 (`homeClientUsers`) of the moduleMethod arguments is replaced with the users associated to the edge ID. Parameter 7 (`userHomeClients`) is overridden with an empty array.
2. When parameter 4 (`userId`) is set, the edge clients associated to this user are loaded from database. Parameter 7 (`userHomeClients`) is overridden with this data. When parameter 5 (`homeClientId`) is empty, parameter 6 is overridden with an empty array. When the destination of the packet is C1 Proxy, access is only granted, when the destination edge client ID is in the list of edge clients associated to the user.

RpcServer::userInvoke

This method translates REST requests to RPC requests and therefore is the central point for proxying user requests to modules. User ACL checks need to take place here.

Associating users and edge client instances

Edge instances or modules storing data associated with an edge instance need to be able to access user data or modules of the user's "owning" that edge instance. This means the edge client ACLs need information about the users these edge instances are allowed to access and the user ACL needs information about the edge instances the user has access to. The association between user and edge instance is done by C1 Core. Whenever a user is associated or unassociated with an edge instance, the user and edge client ACLs are updated.

ACLs, principals and principal users

Only end-users can be associated to an edge instance. End-users always have a business partner assigned. Edge instances also have a business partner assigned always. On association these business partners must match. An end-user can't be assigned to an edge instance of a different business partner.

Example user ACL with full access

```
{
  "version": 1,
  "assignableModules": [
    "c1-device-management"
  ],
  "moduleAccess":
  {
    "*": {
      "global": {
        "read": true,
        "write": true,
        "event": true,
        "isAdmin": true
      },
      "rpcMethods": []
    }
  },
  "restAccess": {
    "/*": [
      "GET",
      "POST",
      "PATCH",
      "PUT",
      "DELETE"
    ]
  },
  "assetAccess": [],
  "roleAccess": []
}
```