Thunder Security

Thunder Security provides a way to control access to various RDK Services on the STBs for web and native apps. This requires apps to provide a security token as part of each request to a RDK Service.

This token is generated automatically by the WPE runtime for web apps. The security token contains details on the application context like app URL. Based on the access permissions configured, Thunder will check the security token as part of each incoming request and allow/deny access to the RDK service. More details are explained below.

Figure 1: Component diagram.

- Application - A Lightning/web app (WPE runtime) or Lightning/Spark app (Spark runtime) loaded from the web.
- Thunder client - javascript client used by application
- WPE/Spark - Application Runtime Environments
- Thunder - Web Platform for Embedded Framework for services
- Security Agent - A thunder plugin. Accessible only by COM/RPC only by application runtimes.
- Plugin1 - Represents any plugin used as a service. Accessible by JSON/RPC
- thunder_permissions.conf - configuration for permissions of applications to access Thunder services.

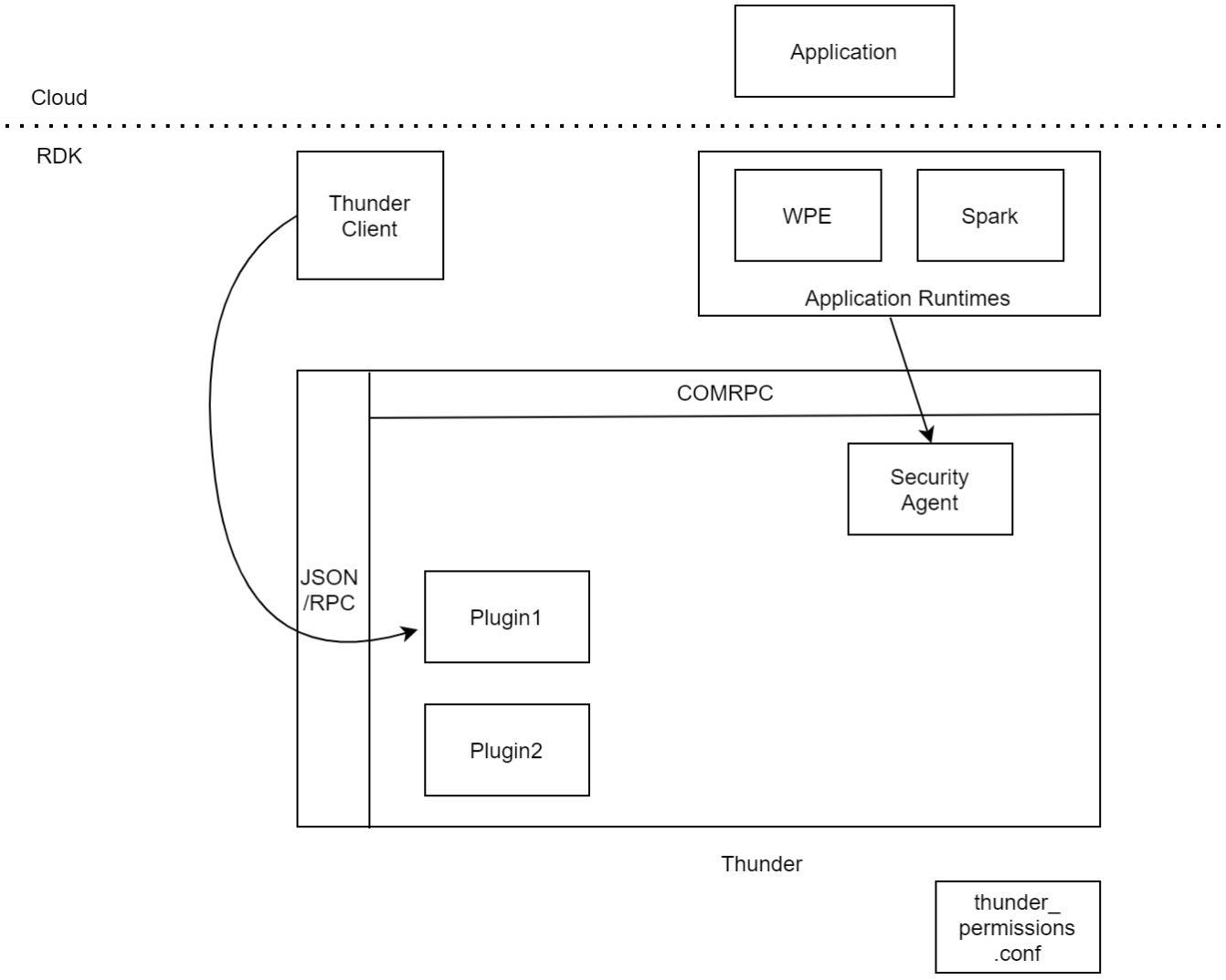
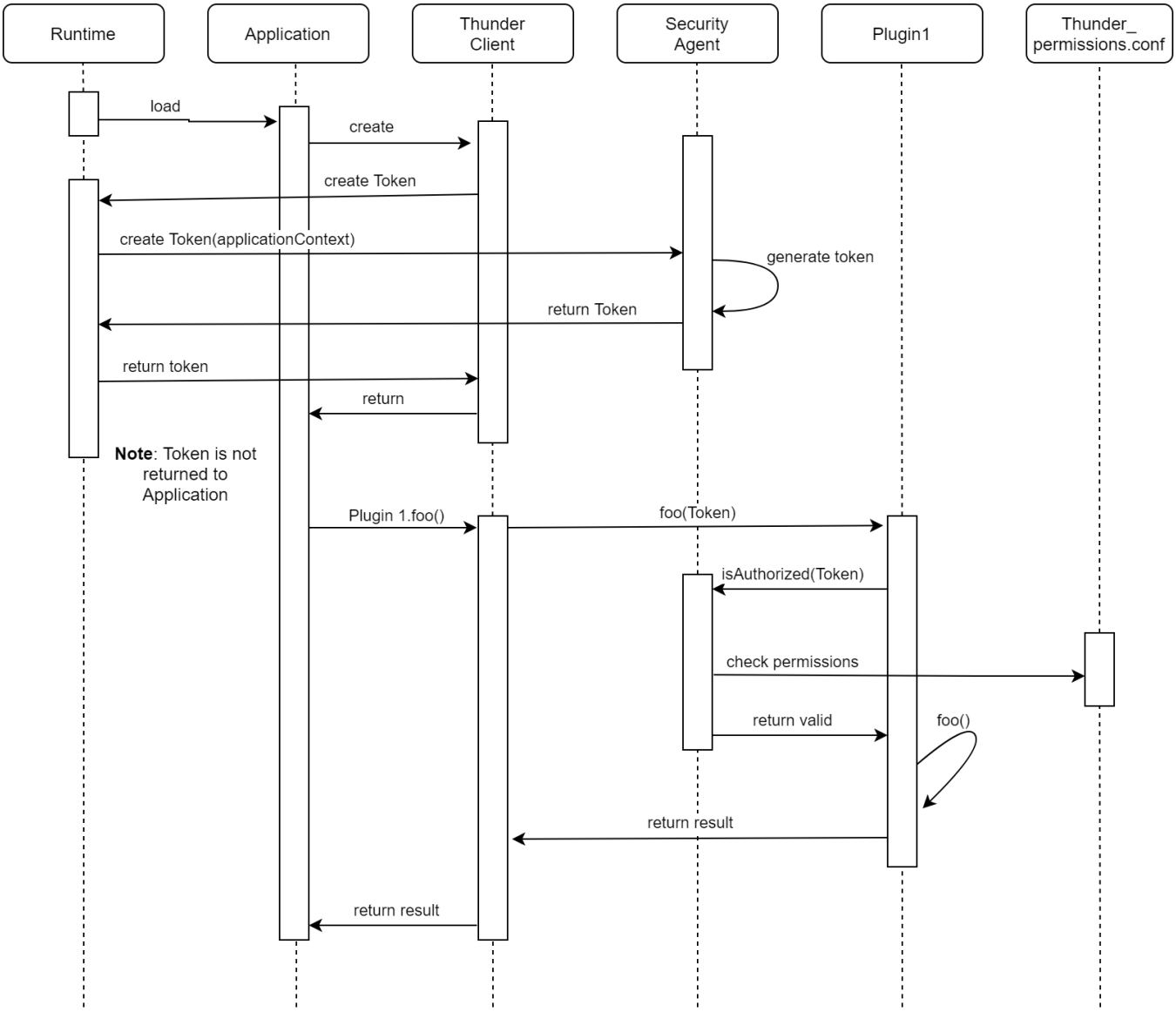


Figure 2. Sequence Diagram

Notes: The Spark or WPE runtime loads the application. The application that wants to use Thunder services creates the ThunderClient. When the client is created, it requests a security token from the runtime. The runtime then requests the token directly from the security agent through COM/RPC using the application context (the application's URL). The security agent then creates and returns the token and returns it to the client. As noted, the token is not returned back to the application that creates the Thunder client. Once the application has created the Thunder Client, it then can invoke a service on a plugin... in this case Plugin1.foo(). The Thunder Client creates the request to Thunder by including the security token. The Thunder Framework (not the actual plugin) checks with the security plugin to determine if the application can access this plugin based on the permissions file, and if so, allows the plugin to perform the request with the result being returned to the application.



Sending Security Token in JSON-RPC Requests

Web Apps

Web Apps that use ThunderJS to access plugins don't have to do anything to pass the security token. This is handled by the WPE runtime and ThunderJS. WPE runtime will get details of the web app (URL) and send these details to WPEFramework process to get a security token. ThunderJS client will read this token and include it as part of each request.

Native Apps

a) Native apps can use the GetSecurityToken() API from libWPEFrameworkSecurityUtil.so to get the security token.

Send Security Token in Native App

```
#include <securityagent/SecurityTokenUtil.h>
#define MAX_LENGTH 1024


unsigned char buffer[MAX_LENGTH] = {0};
int ret = GetSecurityToken(MAX_LENGTH,buffer);
if(ret > 0)
{
    string sToken = (char*)buffer;
    string query = "token=" + sToken;
    auto thunderClient = WPEFramework::JSONRPC::LinkType<WPEFramework::Core::JSON::IElement>("org.rdk.
DisplaySettings", "", false, query);
}
```

b) Native apps can use the WPEFrameworkSecurityUtility application to get a security token for localhost. This will give a security token as part of a json response as shown below.

WPEFrameworkSecurityUtility

```
cd /opt/logs# /usr/bin/WPEFrameworkSecurityUtility
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1cmwiOiJodHRwOi8vbG9jYWxob3N0In0.u1b-
drnYqxLlxAHqTLEFu8PmJ2iKNvrnvmyO0ofvrOI","success":true}
```

Native apps will have to send this token as a query string to JSONRPC::LinkType object as shown below.

Send Security Token in Native App

```
// assuming sToken contains the token retreived from WPEFrameworkSecurityUtility
string sToken;

string query = "token=" + sToken;
auto thunderClient = WPEFramework::JSONRPC::LinkType<WPEFramework::Core::JSON::IElement>("org.rdk.
DisplaySettings", "", false, query);
```

Sending Security Token in HTTP Requests

Security Token can be passed in the header as part of REST API calls to thunder to access RDK Services. These HTTP calls are used during development, triage and testing.

Similar to native apps, we can run the WPEFrameworkSecurityUtility application to get a security token for localhost. This will give a security token as part of the json response which should be passed as part of Authorization header as shown below.

HTTP Requests

```
cd /opt/logs# /usr/bin/WPEFrameworkSecurityUtility
{"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1cmwiOiJodHRwOi8vbG9jYWxob3N0In0.u1b-
drnYqxLlxAHqTLEFu8PmJ2iKNvrnvmyO0ofvrOI","success":true}


cd /opt/logs# curl -H "Content-Type: application/json"  -H "Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1cmwiOiJodHRwOi8vbG9jYWxob3N0In0.u1b-
drnYqxLlxAHqTLEFu8PmJ2iKNvrnvmyO0ofvrOI" -X POST   -d '{"jsonrpc":"2.0","id":1,"method":"org.rdk.
DisplaySettings.1.getCurrentResolution"}' http://127.0.0.1:9998/jsonrpc

{"jsonrpc":"2.0","id":1,"result":{"resolution":"1080p60","success":true}}
```

Additional Info on Security Token

The security token generated on a device is only valid for that device. It cannot be used for another device. Also, the token needs to be regenerated if the device is rebooted or services are restarted.

Error Handling

If no security token is passed in the request or invalid security token is passed, then WPEFramework::Core::ERROR_PRIVILIGED_REQUEST (24) error code is returned.

Native Apps

Any calls to WPEFramework::JSONRPC::LinkType APIs will return WPEFramework::Core::ERROR_PRIVILIGED_REQUEST if correct security token is not passed.

Error code in Native App

```
// Error code when no security token is passed.

auto thunderClient = WPEFramework::JSONRPC::LinkType<WPEFramework::Core::JSON::IElement>("org.rdk.
DisplaySettings", "");
JsonObject joParams;
JsonObject joResult;
uint32_t status = thunderClient.Invoke(2000, "getCurrentResolution", joParams, joResult);


if(status == WPEFramework::Core::ERROR_PRIVILIGED_REQUEST)
{
    cout << "Access denied. Pass valid security token" << endl;
}
```

Http Requests

Error code for HTTP Requests

```
cd /opt/logs# curl -H "Content-Type: application/json"   -X POST   -d '{"jsonrpc":"2.0","id":1,"method":"
Controller.1.status@org.rdk.DisplaySettings"}' http://127.0.0.1:9998/jsonrpc

{"jsonrpc":"2.0","id":1,"error":{"code":24,"message":"Request needs authorization. Missing or invalid token."}}
```

RFC Support

Thunder Security can be enabled/disabled using RFC Device.DeviceInfo.X_RDKCENTRAL-COM_RFC.Feature.ThunderSecurity.Enable. When Device.DeviceInfo.X_RDKCENTRAL-COM_RFC.Feature.ThunderSecurity.Enable is set to true, then Thunder Security is enabled and disabled when set to false.

The RFC can be set using XCONF or locally on the STB using tr181 commands. STB needs to be rebooted after changing the RFC value.

Details on Security Permissions

The Thunder security permissions or ACL are defined under /etc/thunder_acl.json. Sample permissions are shown below as an example. "assign" contains a list of urls for which different roles are assigned.

"roles" define each of the roles which specify the ACL as an allow and block list. For development and testing purposes /etc/thunder_acl.json file can be copied to /opt/thunder_acl.json and edited. This is allowed only on VBN builds.

Sample ACL

```
{
    "assign": [
        {
            "url": "*://localhost",
            "role": "local"
        },
        {
            "url": "*://testurl1.com/*",
            "role": "restricted1"
        },
        {
            "url": "*://testurl2.com/*",
            "role": "restricted2"
        },
        {
            "url": "*",
            "role": "default"
        }
    ]
"roles": {
        "default": {
            "thunder": {
                "block": [
                    "*"
                ]
            }
        },
        "local": {
            "thunder": {
                "allow": [
                    "*"
                ]
            }
        },
        "restricted1": {
            "thunder": {
                "allow": [
                    "org.rdk.DisplaySettings",
                    "org.rdk.Timer"
                ]
            }
        },
         "restricted2": {
            "thunder": {
            "allow": [
                    "*"
                ],
                "block": [
                    "org.rdk.SystemServices",
                    "org.rdk.StorageManager"
                ]
            }
        }
    }
}
```