SEL-5056 v0.4 Getting Started

Whats New

- Updates to this document colored green
- OpenSSL required - https://slproweb.com/products/Win32OpenSSL.html - Win32 OpenSSL v1.0.2d Light
- .NET 4.6 required - Microsoft .NET Framework 4.6 (Offline Installer) https://www.microsoft.com/en-us/download/details.aspx?id=48137
- GUI interface (web based) is now enabled
- Python bindings for the REST interface

Known Bugs

- The user interface should allow you to arrange nodes in the order you want, but when you do that it will re-arrange them back to the way it originally displayed them.
- Spurious disconnected "Ghost links" are showing up in the topology due to an interaction between the "auto adoption agent" (which is being provided as a convenience for UIUC) and the host discovery.

Getting Started

Copy the SEL-5056v0.4.2 folder to the computer you want it to run on. I will refer to that computer as "sdncontroller.example.com".

Configuration

Listening for external REST connections...

By default, SEL-5056 is configured to listen only on localhost:1234. To change this, edit file SEL-5056.exe.config and change the "value" of "appSettings / webURI" to the name of the interface to listen on. For example, if SEL-5056 is running on computer sdncontroller.example.com...

```
<add key="webUri" value="http://sdncontroller.example.com:1234" />
```

See Note below about running the controller when the webUri has been changed.

If you wish to use the GUI you must edit the hostnames in `static/src/config/properties.js` which refer to localhost (Properties.SDN_CONTROLLER_URI and Properties.SDN_IDENTITY_SERVER_URI) to refer to the computer's name.

Logging...

The logging level can be configured in file logging.xml. Logging is initially set to level "Info". The logging.xml file shows how to change the logging level to "Trace" (all messages). This is probably not useful unless you encounter problems.

Running the controller

Execute SEL-5056.exe from the distribution folder.

If the computer name is changed from "localhost" (see "Listening for external REST connections" above), then the controller must be run as an administrator by right-clicking on the SEL-5056.exe executable and selecting "Run as administrator".

A "cmd" window will be opened if you run the controller by clicking on the icon. You could also open up a cmd window in the directory (possibly running cmd as administrator) and run SEL-5056 from the command line.

Use <ctrl-c> to stop the controller.

Authentication

All REST requests to the system outside the identity services must have an "Authorization" header in the format "Bearer ACCESS_TOKEN". Access tokens must be retrieved from the controller using one of the OAuth endpoints. Per the OAuth standard, information about the supported OAuth endpoints can be retrieved from the controller at the endpoint:

> http://sdncontroller.example.com:1234/identity/.well-known/openid-configuration

(To assist UIUC with this, SEL will provide an example Python script that will retrieve a bearer token for the "validator" user, using the "password" grant type, but other options are possible.)

> Bearer tokens expire in an hour. (There may be OAuth libraries for Python that will transparently handle renewing the token?)

For this release, the system has three users and three roles hard-coded: Superuser (with role Admin), hobbs (with role Engineer) and validator (with role Readonly).

- The Admin role has access to create and modify users
- The Engineer role has access to create and modify flows and topology objects
- The Readonly role is provided to model the expected connection capabilities that will be provided for flow validation in the future

The password for all of the users is "Asdf123$".

Exploring the REST interface

I'll assume you're using a REST interface tool (e.g. DHC, Postman)

Get a bearer token. (The Python program "get_bearer_token.py" can be used.)

In your REST interface tool add a Header with the key "Authorization" and the value "Bearer <bearer token>".

"Self Documenting" REST interface

The REST endpoint uses the Open Data Protocol (OData) o be "self documenting". Of particular interest is the $metadata endpoint at

- http://sdncontroller.example.com:1234/api/default/config/$metadata

which provides an XML document which lists schemas for all the the data types needed to POST data to the config tree. There is also metadata provided for the operational tree (as you might expect) at:

- http://sdncontroller.example.com:1234/api/default/operational/$metadata

Data type required to POST to an endpoint

If you look in the XML at edmx:Edmx / edmx:DataServices / Schema Namespace="Sel" / EntityContainer Name="ConfigTree" you can see a list of endpoints in the config tree (nodes, ports, links, etc.) and the EntityType specifies the name of the Type of the data needed to POST a value to that endpoint. For example, to post a flow to the "flows" endpoint, you need to submit data of type Sel.Sel5056.OpenFlowPlugin.DataTreeObjects .Flow.

Structure of a data type

To see the structure of the data type Sel.Sel5056.OpenFlowPlugin.DataTreeObjects.Flow, you can look in the XML file under edmx:Edmx / edmx:DataServices / Schema Namespace="Sel.Sel5056.OpenFlowPlugin.DataTreeObjects" / EntityType Name="Flow"... you'll see that you need to provide a node, cookie, tableId, bufferId, outGroup, outPort, match, instructions, and id, along with the types of each of those (some of which are themselves complex types that you will need to look up.)

> We understand that use of the OData protocol does not really make the REST interface completely self documenting and will not be surprised if you have questions!

Operational Tree

Do a GET on http://sdncontroller.example.com:1234/api/default/operational

> The response indicates the additional REST endpoints that you can hit:

- http://sdncontroller.example.com:1234/api/default/operational/nodes
- http://sdncontroller.example.com:1234/api/default/operational/ports
- ...
- http://sdncontroller.example.com:1234/api/default/operational/transactions

To get a list of the OpenFlow capable nodes on the network, do a GET on:

> http://sdncontroller.example.com:1234/api/default/operational/nodes

The values have an @odata.type of #Sel.Sel5056.TopologyManager.OpenFlowNode - information about that type is available from the endpoint http://localhost:1234/api/default/operational/$metadata.

To get a list of the flows in the switches do a GET on:

- http://sdncontroller.example.com:1234/api/default/operational/flowStats

The flowStats endpoint returns values organized by switch (indicated by the dataPathId.) The operational/$metadata endpoint can be used to determine the name and structure of the data type of the returned data (Sel.Sel5056.OpenFlowPlugin.DataTreeObjects.FlowStats).

Config Tree

Do a GET on http://sdncontroller.example.com:1234/api/default/config

As before, the response indicates the available REST endpoints in the config tree:

- http://sdncontroller.example.com:1234/api/default/config/nodes
- http://sdncontroller.example.com:1234/api/default/config/ports
- http://sdncontroller.example.com:1234/api/default/config/links
- ...
- http://sdncontroller.example.com:1234/api/default/config/flowSet
- http://sdncontroller.example.com:1234/api/default/config/transactions

Programming Flows

Flows get programmed on the config tree, then show up on the operational tree when the switch reports that the flow is in its flow table.

First, find the id of the node you want to program the flow onto on the config tree (these ids are GUIDs that have had their dashes removed and an 'a' prepended to them - e.g. a8b6035a7926e4329b2ce4138a6fb1baa)

- http://sdncontroller.example.com:1234/api/default/config/nodes

Now, to program a flow you need to create JSON for an Sel.Sel5056.OpenFlowPlugin.DataTreeObjects.Flow object. (Earlier, I described how you know this from the information provided by the config/$metadata REST endpoint, as well as how to find the structure that you need to build.)

Here is an example JSON structure that can be submitted (after replacing the text "<node id here>" with the id of the node you want the flow programmed for...

```
{
    "instructions": [
        {
            "@odata.type": "#Sel.Sel5056.OpenFlowPlugin.DataTreeObjects.WriteActions",
            "instructionType": "WriteActions",
            "actions": [
                {
                    "maxLength": 65535,
                    "@odata.type":
"#Sel.Sel5056.OpenFlowPlugin.DataTreeObjects.OutputAction",
                    "outPort": 1,
                    "setOrder": 0,
                    "actionType": "Output"
                }
            ]
        }
    ],
    "outPort": 0,
    "bufferId": 0,
    "node": "<node id here>",
    "match": {
        "ethDst": null,
        "udpSrc": null,
        "ipv4Dst": null,
        "tcpDst": null,
        "udpDst": null,
        "ipv4Src": "192.168.1.1",
        "vlanVid": null,
        "vlanPcp": null,
        "inPort": "3",
        "tcpSrc": null,
        "ethSrc": null,
        "ethType": "2048"
    },
    "cookie": 0,
    "tableId": 0,
    "outGroup": 0
}
```

You should then check that the flow got created on the operational tree (under flowStats). At this time we do not have a feedback mechanism to give you additional information if a flow fails to program. Also note that every POST to the config tree will result in an entry in the GET from the same endpoint (in the config tree), so if you POST the same flow multiple times, it will show up in the GET from the config tree multiple times, but it will only show up once in a GET from the operational tree because an OpenFlow switch throws away duplicate flows.

Removing programmed flows

Removing flows seems to be broken - you can use a DELETE action to remove the entries from the config table, but they do not get removed from the operational table. To delete a flow with id "a08415ea1ac9a465aba2cf82f1dee6e1d" you would perform a DELETE action on endpoint http://sdncontroller.example.com:1234/api/default/config/flows('a08415ea1ac9a465aba2cf82f1dee6e1d')

Topology

Topology information is available in the operational tree. Nodes are under the "nodes" endpoint and links are under the "links" endpoint.

GUI

The GUI should now be available on http://sdncontroller.example.com:1234/static/index.html - It is far from complete and bug-free, but may be helpful.