Managing Roles, Users, and Signature-Based Transactions

Managing APIC Roles and Users

User Access, Authorization, and Accounting

Application Policy Infrastructure Controller (APIC) policies manage the authentication, authorization, and accounting (AAA) functions of the Cisco Application Centric Infrastructure (ACI) fabric. The combination of user privileges, roles, and domains with access rights inheritance enables administrators to configure AAA functions at the managed object level in a granular fashion. These configurations can be implemented using the REST API, the CLI, or the GUI.

Note There is a known limitation where you cannot have more than 32 characters for the login domain name. In addition, the combined number of characters for the login domain name and the user name cannot exceed 64 characters.

Accounting

ACI fabric accounting is handled by these two managed objects (MO) that are processed by the same mechanism as faults and events:

- The `aaaSessionLR` MO tracks user account login and logout sessions on the APIC and switches, and token refresh. The ACI fabric session alert feature stores information such as the following:
  - Username
  - IP address initiating the session
  - Type (telnet, https, REST etc.)
  - Session time and length

- Token refresh – a user account login event generates a valid active token which is required in order for the user account to exercise its rights in the ACI fabric.

Note Token expiration is independent of login; a user could log out but the token expires according to the duration of the timer value it contains.

- The `aaaModLR` MO tracks the changes users make to objects and when the changes occurred.
- If the AAA server is not pingable, it is marked unavailable and a fault is seen.

Both the `aaaSessionLR` and `aaaModLR` event logs are stored in APIC shards. Once the data exceeds the pre-set storage allocation size, it overwrites records on a first-in first-out basis.

Note In the event of a destructive event such as a disk crash or a fire that destroys an APIC cluster node, the event logs are lost; event logs are not replicated across the cluster.

The `aaaModLR` and `aaaSessionLR` MOs can be queried by class or by distinguished name (DN). A class query provides all the log records for the whole fabric. All `aaaModLR` records for the whole fabric are available from the GUI at the **Fabric** > **Inventory** > **POD** > **History** > **Audit Log** section, The APIC GUI **History** > **Audit Log** options enable viewing event logs for a specific object identified in the GUI.

The standard syslog, callhome, REST query, and CLI export mechanisms are fully supported for `aaaModLR` and `aaaSessionLR` MO query data. There is no default policy to export this data.

There are no pre-configured queries in the APIC that report on aggregations of data across a set of objects or for the entire system. A fabric administrator can configure export policies that periodically export `aaaModLR` and `aaaSessionLR` query data to a syslog server. Exported data can be archived periodically and used to generate custom reports from portions of the system or across the entire set of system logs.

Multiple Tenant Support

A core Application Policy Infrastructure Controller (APIC) internal data access control system provides multitenant isolation and prevents information privacy from being compromised across tenants. Read/write restrictions prevent any tenant from seeing any other tenant's configuration, statistics, faults, or event data. Unless the administrator assigns permissions to do so, tenants are restricted from reading fabric configuration, policies, statistics, faults, or events.

User Access: Roles, Privileges, and Security Domains

The APIC provides access according to a user’s role through role-based access control (RBAC). An Cisco Application Centric Infrastructure (ACI) fabric user is associated with the following:

- A set of roles
- For each role, a privilege type: no access, read-only, or read-write
- One or more security domain tags that identify the portions of the management information tree (MIT) that a user can access

The ACI fabric manages access privileges at the managed object (MO) level. A privilege is an MO that enables or restricts access to a particular function within the system. For example, fabric-equipment is a privilege bit. This bit is set by the Application Policy Infrastructure Controller (APIC) on all objects that correspond to equipment in the physical fabric.

A role is a collection of privilege bits. For example, because an “admin” role is configured with privilege bits for “fabric-equipment” and “tenant-security,” the “admin” role has access to all objects that correspond to equipment of the fabric and tenant security.

A security domain is a tag associated with a certain subtree in the ACI MIT object hierarchy. For example, the default tenant “common” has a domain tag `common`. Similarly, the special domain tag `all` includes the entire MIT object tree. An administrator can assign custom domain tags to the MIT object hierarchy. For example, an administrator could assign the “solar” domain tag to the tenant named solar. Within the MIT, only certain objects can be tagged as security domains. For example, a tenant can be tagged as a security domain but objects within a tenant cannot.

Note Security Domain password strength parameters can be configured by creating **Custom Conditions** or by selecting **Any Three Conditions** that are provided.

Creating a user and assigning a role to that user does not enable access rights. It is necessary to also assign the user to one or more security domains. By default, the ACI fabric includes two special pre-created domains:

- `All`—allows access to the entire MIT
- `Infra`— allows access to fabric infrastructure objects/subtrees, such as fabric access policies

Note For read operations to the managed objects that a user's credentials do not allow, a "DN/Class Not Found" error is returned, not "DN/Class Unauthorized to read." For write operations to a managed object that a user's credentials do not allow, an HTTP 401 Unauthorized error is returned. In the GUI, actions that a user's credentials do not allow, either they are not presented, or they are grayed out.

A set of predefined managed object classes can be associated with domains. These classes should not have overlapping containment. Examples of classes that support domain association are as follows:

- Layer 2 and Layer 3 network managed objects
- Network profiles (such as physical, Layer 2, Layer 3, management)
- QoS policies

When an object that can be associated with a domain is created, the user must assign domain(s) to the object within the limits of the user's access rights. Domain assignment can be modified at any time.

If a virtual machine management (VMM) domain is tagged as a security domain, the users contained in the security domain can access the correspondingly tagged VMM domain. For example, if a tenant named solar is tagged with the security domain called sun and a VMM domain is also tagged with the security domain called sun, then users in the solar tenant can access the VMM domain according to their access rights.

Configuring a Custom Role Using the REST API

SUMMARY STEPS

1. To configure a custom role, send a POST request with XML as in the following example:

DETAILED STEPS

To configure a custom role, send a POST request with XML as in the following example:

Example:

```
<aaaRoleresetToFactory="no"

priv="aaa,access-connectivity-l1,access-connectivity-l2,access-connectivity-l3,access-connectivity-mgmt,


access-connectivity-util,access-equipment,access-protocol-l1,access-protocol-l2,access-protocol-l3,access-protocol-mgmt,

    access-protocol-ops,access-protocol-util,access-qos,fabric-connectivity-l1,fabric-connectivity-l2,

     fabric-connectivity-l3,fabric-connectivity-mgmt,fabric-connectivity-util,fabric-equipment,
    fabric-protocol-l1,fabric-protocol-l2,fabric-protocol-l3,fabric-protocol-mgmt,fabric-protocol-ops,

     fabric-protocol-util,nw-svc-device,nw-svc-devshare,nw-svc-policy,ops,tenant-connectivity-l1,
    tenant-connectivity-l2,tenant-connectivity-l3,tenant-connectivity-mgmt,tenant-connectivity-util,

     tenant-epg,tenant-ext-connectivity-l1,tenant-ext-connectivity-l2,tenant-ext-connectivity-l3,

tenant-ext-connectivity-mgmt,tenant-ext-connectivity-util,tenant-ext-protocol-l1,tenant-ext-protocol-l2,

    tenant-ext-protocol-l3,tenant-ext-protocol-mgmt,tenant-ext-protocol-util,tenant-network-profile,

    tenant-protocol-l1,tenant-protocol-l2,tenant-protocol-l3,tenant-protocol-mgmt,tenant-protocol-ops,


tenant-protocol-util,tenant-qos,tenant-security,vmm-connectivity,vmm-ep,vmm-policy,vmm-protocol-ops,

     vmm-security" ownerTag="" ownerKey="" name="tenant-admin" dn="uni/userext/role-tenant-admin"
descr=""/>
```

Configuring a Local User

In the initial configuration script, the admin account is configured and the admin is the only user when the system starts. The APIC supports a granular, role-based access control system where user accounts can be created with various roles including non-admin users with fewer privileges.

Configuring a Local User Using the REST API

SUMMARY STEPS

1. Create a local user.

DETAILED STEPS

Create a local user.

Example:

```
URL: https://apic-ip-address/api/node/mo/uni/userext.xml
POST CONTENT:
    <aaaUser name="operations" phone="" pwd="<strong_password>" >
        <aaaUserDomain childAction="" descr="" name="all" rn="userdomain-all" status="">
        <aaaUserRole childAction="" descr="" name="Ops" privType="writePriv"/>
        </aaaUserDomain>
    </aaaUser>
```

Configuring a Remote User

Instead of configuring local users, you can point the APIC at the centralized enterprise credential datacenter. The APIC supports Lightweight Directory Access Protocol (LDAP), active directory, RADIUS, and TACACS+.

Note When an APIC is in minority (disconnected from the cluster), remote logins can fail because the ACI is a distributed system and the user information is distributed across APICS. Local logins, however, continue to work because they are local to the APIC.

Starting with the 3.1(1) release, **Server Monitoring** can be configured through RADIUS, TACACS+, LDAP, and RSA to determine whether the respective AAA servers are alive or not. Server monitoring feature uses the respective protocol login to check for server aliveness. For example, a LDAP server will use ldap login and a Radius server will use radius login with server monitoring to determine server aliveness.

To configure a remote user authenticated through an external authentication provider, you must meet the following prerequisites:

- The DNS configuration should have already been resolved with the hostname of the RADIUS server.
- You must configure the management subnet.

Configuring a Remote User Using the REST API

SUMMARY STEPS

1. Create a RADIUS provider.
2. Create a login domain.

DETAILED STEPS

Step 1 Create a RADIUS provider.

Example:

```
 URL: https://apic-ip-address/api/policymgr/mo/uni/userext/radiusext.xml
POST Content:
<aaaRadiusProvider name="radius-auth-server.org.com" key="test123" />
```

Step 2 Create a login domain.

Example:

```
 URL: https://apic-ip-address/api/policymgr/mo/uni/userext.xml
POST Content:
<aaaLoginDomain name="rad"> <aaaDomainAuth realm="radius"/> </aaaLoginDomain>
```

APIC Signature-Based Transactions

About Signature-Based Transactions

The APIC controllers in a Cisco ACI fabric offer different methods to authenticate users.

The primary authentication method uses a username and password and the APIC REST API returns an authentication token that can be used for future access to the APIC. This may be considered insecure in a situation where HTTPS is not available or enabled.

Another form of authentication that is offered utilizes a signature that is calculated for every transaction. The calculation of that signature uses a private key that must be kept secret in a secure location. When the APIC receives a request with a signature rather than a token, the APIC utilizes an X.509 certificate to verify the signature. In signature-based authentication, every transaction to the APIC must have a newly calculated signature. This is not a task that a user should do manually for each transaction. Ideally this function should be utilized by a script or an application that communicates with the APIC. This method is the most secure as it requires an attacker to crack the RSA/DSA key to forge or impersonate the user credentials.

Note Additionally, you must use HTTPS to prevent replay attacks.

Before you can use X.509 certificate-based signatures for authentication, verify that the following pre-requisite tasks are completed:

1. Create an X.509 certificate and private key using OpenSSL or a similar tool.
2. Create a local user on the APIC. (If a local user is already available, this task is optional).
3. Add the X.509 certificate to the local user on the APIC.

Using a Private Key to Calculate a Signature

Before you begin

You must have the following information available:

- HTTP method - GET, POST, DELETE

- REST API URI being requested, including any query options
- For POST requests, the actual payload being sent to the APIC
- The private key used to generate the X.509 certificate for the user
- The distinguished name for the user X.509 certificate on the APIC

Step 1 Concatenate the HTTP method, REST API URI, and payload together in this order and save them to a file.

This concatenated data must be saved to a file for OpenSSL to calculate the signature. In this example, we use a filename of payload.txt. Remember that the private key is in a file called userabc.key.

Example:

GET example:

```
GET http://10.10.10.1/api/class/fvTenant.json?rsp-subtree=children
```

POST example:

```
POST http://10.10.10.1/api/mo/tn-test.json{"fvTenant": {"attributes": {"status": "deleted", "name":
 "test"}}}
```

Step 2 Verify that the payload.txt file contains the correct information.

For example, using the GET example shown in the previous step:

```
GET http://10.10.10.1/api/class/fvTenant.json?rsp-subtree=children
```

Your payload.txt file should contain only the following information:

```
GET/api/class/fvTenant.json?rsp-subtree=children
```

Step 3 Verify that you didn't inadvertently create a new line when you created the payload file.

Example:

```
# cat -e payload.txt
```

Determine if there is a **$** symbol at the end of the output, similar to the following:

```
GET/api/class/fvTenant.json?rsp=subtree=children$
```

If so, then that means that a new line was created when you created the payload file. To prevent creating a new line when generating the payload file, use a command similar to the following:

```
echo -n "GET/api/class/fvTenant.json?rsp-subtree=children" >payload.txt
```

Step 4 Calculate a signature using the private key and the payload file using OpenSSL.

Example:

```
openssl dgst -sha256 -sign userabc.key payload.txt > payload_sig.bin
```

The resulting file has the signature printed on multiple lines.

Step 5 Convert the signature to base64 format:

Example:

```
openssl base64 -A -in payload_sig.bin -out payload_sig.base64
```

Step 6 Strip the signature of the new lines using Bash.

Example:

```
$ tr -d '\n' < payload_sig.base64
P+OTqK0CeAZjl7+Gute2R1Ww8OGgtzE0wsLlx8fIXXl4V79Zl7
Ou8IdJH9CB4W6CEvdICXqkv3KaQszCIC0+Bn07o3qF//BsIplZmYChD6gCX3f7q
IcjGX+R6HAqGeK7k97cNhXlWEoobFPe/oajtPjOu3tdOjhf/9ujG6Jv6Ro=
```

Note This is the signature that will be sent to the APIC for this specific request. Other requests will require to have their own signatures calculated.

Step 7 Place the signature inside a string to enable the APIC to verify the signature against the payload.

This complete signature is sent to the APIC as a cookie in the header of the request.

Example:

```
APIC-Request-Signature=P+OTqK0CeAZjl7+Gute2R1Ww8OGgtzE0wsLlx8f
IXXl4V79Zl7Ou8IdJH9CB4W6CEvdICXqkv3KaQszCIC0+Bn07o3qF//BsIplZmYChD6gCX3f
7qIcjGX+R6HAqGeK7k97cNhXlWEoobFPe/oajtPjOu3tdOjhf/9ujG6Jv6Ro=;
APIC-Certificate-Algorithm=v1.0; APIC-Certificate-Fingerprint=fingerprint;
APIC-Certificate-DN=uni/userext/user-userabc/usercert-userabc.crt
```

Note The DN used here must match the DN of the user certified object containing the x509 certificate in the next step.

Step 8 Use the CertSession class in the Python SDK to communicate with an APIC using signatures.

The following script is an example of how to use the CertSession class in the ACI Python SDK to make requests to an APIC using signatures.

Example:

```
#!/usr/bin/env python
# It is assumed the user has the X.509 certificate already added to
# their local user configuration on the APIC
from cobra.mit.session import CertSession
from cobra.mit.access import MoDirectory

def readFile(fileName=None, mode="r"):
    if fileName is None:
        return ""
    fileData = ""
    with open(fileName, mode) as aFile:
        fileData = aFile.read()
    return fileData

pkey = readFile("/tmp/userabc.key")
csession = CertSession("https://ApicIPOrHostname/",
                       "uni/userext/user-userabc/usercert-userabc", pkey)

modir = MoDirectory(csession)
resp = modir.lookupByDn('uni/fabric')
pring resp.dn
# End of script
```

Note The DN used in the earlier step must match the DN of the user certified object containing the x509 certificate in this step.

Guidelines and Limitations

Follow these guidelines and limitations:

- Local users are supported. Remote AAA users are not supported.
- The APIC GUI does not support the certificate authentication method.
- WebSockets and eventchannels do not work for X.509 requests.
- Certificates signed by a third party are not supported. Use a self-signed certificate.

Creating a Local User and Adding a User Certificate Using the REST API

Create a local user and add a user certificate.

Example:

```
method: POST
        url: http://apic/api/node/mo/uni/userext/user-userabc.json
        payload:
        {
            "aaaUser": {
                "attributes": {
                    "name": "userabc",
                    "firstName": "Adam",
                    "lastName": "BC",
                    "phone": "408-525-4766",
                    "email": "userabc@cisco.com",
                },
                "children": [{
                    "aaaUserCert": {
                        "attributes": {
                            "name": "userabc.crt",
                           "data": "-----BEGIN CERTIFICATE-----\nMIICjjCCAfegAwIBAgIJAMQnbE <snipped
 content> ==\n-----END CERTIFICATE-----",
                        },
                        "children": []
                    },
                    "aaaUserDomain": {
                        "attributes": {
                            "name": "all",
                        },
                        "children": [{
                            "aaaUserRole": {
                                "attributes": {
                                    "name": "aaa",
                                    "privType": "writePriv",
                                },
                                "children": []
                            }
                        }, {
                            "aaaUserRole": {
                                "attributes": {
                                    "name": "access-admin",
                                    "privType": "writePriv",
                                },
                                "children": []
                            }
                        }, {
```

```
    "aaaUserRole": {
        "attributes": {
            "name": "admin",
            "privType": "writePriv",
        },
        "children": []
    }
}, {
    "aaaUserRole": {
        "attributes": {
            "name": "fabric-admin",
            "privType": "writePriv",
        },
        "children": []
    }
}, {
    "aaaUserRole": {
        "attributes": {
            "name": "nw-svc-admin",
            "privType": "writePriv",
        },
        "children": []
    }
}, {
    "aaaUserRole": {
        "attributes": {
            "name": "ops",
            "privType": "writePriv",
        },
        "children": []
    }
}, {
    "aaaUserRole": {
        "attributes": {
            "name": "read-all",
            "privType": "writePriv",
        },
        "children": []
    }
}, {
    "aaaUserRole": {
        "attributes": {
            "name": "tenant-admin",
            "privType": "writePriv",
        },
        "children": []
    }
}, {
    "aaaUserRole": {
        "attributes": {
            "name": "tenant-ext-admin",
            "privType": "writePriv",
        },
        "children": []
    }
}, {
    "aaaUserRole": {
        "attributes": {
            "name": "vmm-admin",
            "privType": "writePriv",
        },
        "children": []
    }
}]
```

```
                    }
                } ]
            }
        }
```