Sonatype CLM Server - REST APIs

Contents

Chapter 1

Introduction

Using REST API calls, the Sonatype CLM Server provides functionality to create and update applications, as well as retrieve values for policy violations.

These APIs have been designed for system-to-system functionality; however, examples are provided using the HTTP client CURL. Following along, you can initiate the described API REST request via the command line tool.

Chapter 2

Sonatype CLM Application REST APIs

The primary function of the Sonatype CLM Application REST APIs is for the creation and update of applications. In addition to this, tags can be added and users /groups mapped to Sonatype CLM roles (permissions) for the application. The currently available APIs include:

- GET - used to retrieve information, such as a list of organizations, their internal IDs, and their tags. This is also used for retrieving roles and mapping those roles to an application.
- POST - used to create applications.
- PUT - used for mapping roles to the application, as well as making any necessary changes to modifiable application properties (i.e. name, tags, or contact).

As mentioned in the introduction we will provide both the API, as well as an example using the HTTP client CURL. Additionally, to help demonstrate use of the APIs, we've approached this in a step-by-step manner that will start with gathering the necessary information to create and set permissions for an application.

Tip
If you already have an application, and wish to make an update, you can jump to the final step now.

Step 1 - Get the Organization ID

In order to create an application, it must have a parent organization. This is an inherent requirement of the Sonatype CLM system. Thus, we must start with our GET API call...

```
GET  /api/v1/organizations
```

and find the Organization ID. Additionally, any tags available to be applied to the application will be provided. To follow along using CURL, enter the following command:

```
curl -u admin:admin123 -X GET http://localhost:8070/api/v1/organizations
```

This will produce a list of your organizations and associated information in a JSON format. Here is an example of what might be returned.

```
{
    "organizations": [
        {
            "id": "36d7e629462a4038b581488c347959bc",
            "name": "My Organization",
            "tags": [ ]
        },
        {
            "id": "f48b5344fa204a4c88df96b2d455d521",
            "name": "My Organization 2",
            "tags": [
                {
                    "id": "cd8fd2f4f289445b8975092e7d3045ba",
                    "name": "Distributed",
                    "description": "Applications that are provided for  ↩
                        consumption outside the company",
                    "color": "yellow"
                },
                {
                    "id": "004d789684834f7c889c8b186a5ff24b",
                    "name": "Hosted",
                    "description": "Applications that are hosted such as  ↩
                        services or software as a service.",
                    "color": "grey"
                },
                {
                    "id": "da9e09887c754157a2113831ae7e99ac",
                    "name": "Internal",
                    "description": "Applications that are used only by  ↩
                        your employees",
                    "color": "green"
```

```
                    }
                ]
            }
        ]
}
```

id
: This is the internal id for the organization.

name
: This is the name of the organization, and is visible in the Sonatype CLM GUI.

tags
: Tags represent identifying characteristic for an application. Tags are created at the organization level, and then applied to applications. Policies can then select which tags, and in turn applications, the policy will be evaluated against.

  id (tags)
  : the internal id for the tag. This will be used when creating the application.

  name (tags)
  : the name of the tag, which is visible in the Sonatype CLM GUI.

  description (tags)
  : each tag is required to have a description. This description provides information related to the type of application it should be applied to.

In many cases, you will have more than one organization, as does our example. However, this is not a requirement, and each implementation of Sonatype CLM is unique.

Step 2 - Create an Application

To create an application in Sonatype CLM, you need several pieces of information. As we've already mentioned, you will need the Organization ID, but you will also need:

"publicId" (Application ID)
: This is the Application ID for the application. In the Sonatype CLM Server GUI this is represented by the "Application" field. It must be unique.

"name" (Application Name)
: This is the name of the application. In CLM Server GUI this corresponds to the "Application Name" field. It must be unique.

"contactUserName" (Application Contact)
This is typically the person in charge of the application. In the Sonatype CLM Server GUI, it corresponds to the contact field for the application. The value entered here should represent the user name. If you are using LDAP and have configured it to work with Sonatype CLM, that user name can be used here.

"tagId" (Internal Tag ID)
As mentioned previously, a tag applied to an application will ensure it is evaluated by any policies that have selected the corresponding tag. The Tag ID is returned as part of the organization information.

Tip
The application contact and tags are optional, but it is generally recommended to include them.

We'll be using the POST API call...

```
POST  /api/v1/applications
```

and appending the information mentioned above into a JSON-formatted body. Here is an example of the body that's been formatted for easier review:

```
{
    "publicId": "MyApplicationID",
    "name": "MyFirstApplication",
    "organizationId": "f48b5344fa204a4c88df96b2d455d521",
    "contactUserName": "AppContact",
    "applicationTags": [
        {
            "tagId": "cd8fd2f4f289445b8975092e7d3045ba"
        }
    ]
}
```

Putting this all together, and using our CURL example, you should enter the following command:

```
curl -u admin:admin123 -X POST -H "Content-Type: application/json" -d
'{"publicId": "MyApplicationID","name": "MyFirstApplication"," ↩
    organizationId":
"f48b5344fa204a4c88df96b2d455d521","contactUserName":
"AppContact","applicationTags": [{"tagId":
"cd8fd2f4f289445b8975092e7d3045ba"}]}'
http://localhost:8070/api/v1/applications
```

If your application creation was successful, the system will respond with the following:

```
{
    "id": "4bb67dcfc86344e3a483832f8c496419",
    "publicId": "MyApplicationID",
    "name": "MyFirstApplication",
    "organizationId": "f48b5344fa204a4c88df96b2d455d521",
    "contactUserName": "AppContact",
    "applicationTags": [
        {
            "id": "9beee80c6fc148dfa51e8b0359ee4d4e",
            "tagId": "cd8fd2f4f289445b8975092e7d3045ba",
            "applicationId": "4bb67dcfc86344e3a483832f8c496419"
        }
    ]
}
```

Additionally, CLM will return the standard HTTP response codes.

Tip
Be sure to make note of the "applicationId". This is the internal Application ID that will be used in the next steps when finding roles and mapping a user to them.

Step 3 - Find the Currently Available Roles

Before mapping any roles, it's a good idea to take a look at all the available roles for applications. This can be done using a simple GET:

```
GET /api/v1/applications/roles
```

This returns all the available roles for all applications:

```
{
    "roles": [
        {
            "id": "1da70fae1fd54d6cb7999871ebdb9a36",
            "name": "Developer",
            "description": "Allows to evaluate policies."
        },
        {
            "id": "1cddabf7fdaa47d6833454af10e0a3ef",
```

```
            "name": "Owner",
            "description": "Allows to manage policies."
        }
    ]
}
```

id
This is the internal ID for the role. There is no corresponding field in the Sonatype CLM Server GUI.

name
This is the name of the role, which is exactly the same as the *Role Name* displayed in the Sonatype CLM Server GUI. As we mentioned, there are two role names in the example above `Developer` and `Owner`.

description
This provides a description of the level of access the role grants.

As you may notice, there are two roles available for applications at this time. Before proceeding to the next step, take note of the `roleId` (in this example we will be adding an owner). This will be needed for mapping a user to the specific role.

Step 4 - Map Users to Roles

To map users to roles, we'll need the `id` (Application ID) from our successful application creation, the `roleId` for the role returned above that you will be mapping a user to, and the following information:

type
The type of user, which at this time, can only be either `USER` or `GROUP`.

userOrGroupName
This is the username for the user you want to add to the role. If you are using LDAP and have configured it to work with Sonatype CLM, the LDAP user or group name can be used here.

For this step, we'll be using the following PUT API call...

```
PUT /api/v1/applications/"id"/roleMembers
```

with an appended JSON-formatted body. Here is an example of the body that's been formatted for easier review:

```
{
    "memberMappings": [
        {
            "roleId": "1cddabf7fdaa47d6833454af10e0a3ef",
            "members": [
                {
                    "type": "USER",
                    "userOrGroupName": "user-owner"
                }
            ]
        }
    ]
}
```

Putting this altogether, and using our CURL example, you should enter the following command:

```
curl -u admin:admin123 -X PUT -H "Content-Type: application/json" -d
'{"memberMappings": [{"roleId": "1da70fae1fd54d6cb7999871ebdb9a36"," ↩
    members": [{"type": "USER","userOrGroupName": "user-b"}]}]}'
http://localhost:8070/api/v1/applications/4bb67dcfc86344e3a483832f8c496419 ↩
    /roleMembers
```

As before, CLM will return the standard HTTP response codes.

Tip
You can map multiple roles in a single PUT.

Step 5 - Update Application Information

This final step isn't a requirement to complete an application, but it will be necessary for any application you want to update.

Before updating the application, you will need the `id`, or what is considered the unique, Sonatype CLM ID for the application. If you haven't recorded this, you can obtain it by using the `publicID`. To retrieve the information for the application use the following GET API call...

```
GET /api/v1/applications?publicId={YourPublicId}
```

Tip
All applications can always be returned by omitting the reference to the public id.

Using the CURL command, it would look like this:

```
curl -u admin:admin123 -X GET
http://localhost:8070/api/v1/applications?publicId=MyApplicationID
```

If you like, you can also return multiple applications by appending additional

information will be returned (unique to your application). This has been formatted for readability:

```
{
    "applications": [
        {
            "id": "4bb67dcfc86344e3a483832f8c496419",
            "publicId": "MyApplicationID",
            "name": "MySecondApplication",
            "organizationId": "bb41817bd3e2403a8a52fe8bcd8fe25a",
            "contactUserName": "NewAppContact",
            "applicationTags": [
                {
                    "id": "9beee80c6fc148dfa51e8b0359ee4d4e",
                    "tagId": "cfea8fa79df64283bd64e5b6b624ba48",
                    "applicationId": "4bb67dcfc86344e3a483832f8c496419"
                }
            ]
        }
    ]
}
```

From the information returned above, make note of the following:

- "id"
- "organizationId"

The information for an application that can be updated, follows the same rules as editing an application via the Sonatype CLM GUI. In addition, you will also need any information you will be editing, which can include any or all of the following:

- "name"
- "contactUserName"
- "applicationTags" (tagID)

Note
If you are looking to update the role mapping for the application, simply follow the instructions from step 4.

Now, you should be ready to apply your changes. Before we move on to the command, let's take a look at a sample body for the attached JSON file:

```
{
    "id": "4bb67dcfc86344e3a483832f8c496419",
    "publicId": "MyApplicationID",
    "name": "MySecondApplication",
    "organizationId": "bb41817bd3e2403a8a52fe8bcd8fe25a",
    "contactUserName": "NewAppContact",
    "applicationTags": [
        {
            "tagId": "cfea8fa79df64283bd64e5b6b624ba48"
        }
    ]
}
```

Using CURL enter the following command:

```
curl -u admin:admin123 -X PUT -H "Content-Type: application/json" -d
'{"id":"4bb67dcfc86344e3a483832f8c496419","publicId": "MyApplicationID"," ↩
    name": "MySecondApplication","organizationId":" ↩
    bb41817bd3e2403a8a52fe8bcd8fe25a",
"contactUserName": "NewAppContact","applicationTags":[{"tagId":
"cfea8fa79df64283bd64e5b6b624ba48"}]}'
http://localhost:8070/api/v1/applications/4bb67dcfc86344e3a483832f8c496419
```

If your update is successful, you will see something similar to the formatted JSON file below:

```
{
    "id": "4bb67dcfc86344e3a483832f8c496419",
    "publicId": "MyApplicationID",
    "name": "MySecondApplication",
    "organizationId": "bb41817bd3e2403a8a52fe8bcd8fe25a",
```

```
    "contactUserName": "NewAppContact",
    "applicationTags": [
        {
            "id": "9beee80c6fc148dfa51e8b0359ee4d4e",
            "tagId": "cfea8fa79df64283bd64e5b6b624ba48",
            "applicationId": "4bb67dcfc86344e3a483832f8c496419"
        }
    ]
}
```

Additionally, CLM will return the standard HTTP response codes.

Chapter 3

Sonatype CLM Policy Violation REST API

The Policy Violation REST APIs for Sonatype CLM allow you to access and extract policy violations gathered during the evaluation of applications by the Sonatype CLM Server. In most cases the desire for getting to this data is to integrate into other tools your company may have. For example you may have a specific dashboard or reporting application that should have this data.

Whatever the case, just as with other Sonatype CLM APIs, this is all done using Sonatype CLM REST API calls. For the accessing policy violation information the following API is used:

GET
: Used to retrieve policy information, such as a list of policy ids as well as a list of violations based on a specific Policy ID, or list of IDs.

As mentioned previously, we will provide both the API, as well as exampled using the HTTP client CURL. This is only for demonstration purposes and displaying the necessary input, and desired output.

Additionally, to help demonstrate this, we've approached this in a step-by-step manner that will start with gathering policy ids, and then requesting the violations.

3.1 Before You Get Started

As with other Sonatype CLM REST APIs, you will need a username and password to interface with Sonatype CLM. In addition, because access to this data is granted based on the roles (permissions) you have set up, you may wish to create one specifically for this process.

Other than this, the only piece you may need in order to follow along with our instructions is CURL, or a comparable HTTP client.

Step 1 - Get the Policy IDs

To access policy violation information you need the Policy ID(s). For this reason, we start with the GET API call...

```
GET  /api/v1/policies/
```

which will return a list of all Policy IDs. To follow along using CURL, enter the following command:

```
curl -u admin:admin123 -X GET http://localhost:8070/api/v1/policies
```

The action above will produce a list of your policies in a JSON format. Here is an example of what might be returned.

```
{
    "policies": [
        {
            "id": "6984017845c645b0ad0c95401ad4f17d",
            "name": "My Application Policy",
            "ownerId": "36d7e629462a4038b581488c347959bc",
            "ownerType": "APPLICATION",
            "threatLevel": 5,
            "policyType": "quality"
        },
    ]
}
```

Note
As you can see above, we've used the admin user which is shipped with Sonatype CLM, as well as the default server location. The user you use may differ depending on your configuration.

id
This is the internal id for the policy.

name
This is the name of the policy, and is visible in the Sonatype CLM GUI.

ownerId
This is the internal id for the organization or application that the policy resides in, and is not visible within the Sonatype CLM GUI.

ownerType
This indicates whether the policy is for an organization or application.

threatLevel
This is the threat level that is set for the policy.

policyType
This is the policy type, which is based on the conditions used in the policy.

Tip
In many cases, you will have many policies, especially if you are retrieving information for an account that has access to many applications and/or organization.

Step 2 - Get the Policy Violations

Now that you have the Policy IDs, they can be used to gather a list of policy violations. To do this, you will need the Policy IDs you retrieved from step one. For example:

```
id": "6984017845c645b0ad0c95401ad4f17d"
```

Note
Policy IDs are unique, and thus in the example above, specific to our installation of Sonatype CLM.

Slightly different from before, we will use the GET API call...

```
GET /api/v1/policyViolations?p=policyid1
```

which passes a simple query for Policy IDs. For each policy we want to retrieve violations for, we will include that ID. If desired we can retrieve violations for multiple Policy IDs. To do this, just make sure you add &p="The Policy ID" for each policy you want included. Here is an example of the API with the Policy ID we retrieved:

```
GET /api/v1/policyViolations?p=6984017845c645b0ad0c95401ad4f17d
```

Putting this all together, and using our CURL example, you should enter the following command:

```
curl -u admin:admin123 -X GET http://localhost:8070/api/v1/ ↩
    policyViolations?p=6984017845c645b0ad0c95401ad4f17d
```

If your query was successful, the system will respond with something like this:

```
{
    "applicationViolations": [
        {
            "application": {
                "id": "529b7f71bb714eca8955e5d66687ae2c",
                "publicId": "MyAppID1",
                "name": "MyApplications",
                "organizationId": "36d7e629462a4038b581488c347959bc",
                "contactUserName": null
            },
            "policyViolations": [
                {
                    "policyId": "6984017845c645b0ad0c95401ad4f17d",
                    "policyName": "Security-High",
                    "stageId": "build",
                    "reportUrl": "ui/links/application/MyAppID1/report/ ↩
                        c0ddefc4512f42d0bcbe29029e2be117",
                    "constraintViolations": [
                        {
                            "constraintId": "19011 ↩
                                de290b147a38c820ad7bd5c653d",
                            "constraintName": "CVSS >=7 and <10",
                            "reasons": [
                                {
                                    "reason": "Found 2 Security  ↩
                                        Vulnerabilities with Severity >= 7"
                                },
                                {
```

```
                                    "reason": "Found 4 Security  ↩
                                        Vulnerabilities with Severity < 10"
                                },
                                {
                                    "reason": "Found 4 Security  ↩
                                        Vulnerabilities with Status OPEN"
                                }
                            ]
                        }
                    ],
                    "mavenComponent": {
                        "hash": "384faa82e193d4e4b054",
                        "groupId": "commons-fileupload",
                        "artifactId": "commons-fileupload",
                        "version": "1.2.1"
                    }
                }
            ]
        }
    ]
}
```


And there you have it, you've just retrieved policy violations for two policies (or however many Policy IDs you submitted). Below, each of the categories of data that is returned, as well as each field, have been described.

application
This category contains specific information about the application.

id
This is the internal id.

publicId
This is the Application ID. In the Sonatype CLM Server GUI this is represented by the "Application" field.

name
This is the name of the application. In CLM Server GUI this corresponds to the "Application Name" field.

organizationId
This is the internal id for the organization that the application resides in, and is not visible within the Sonatype CLM GUI.

contactUserName
This is typically the person in charge of the application. In the Sonatype CLM Server GUI, it corresponds to the contact field for the application.

policyViolations
This is a subcategory of the application, and provides specific information about the policy and corresponding violations that were found.

policyId
This is the internal id for the policy.

policyName
This is the name of the policy, and is visible in the Sonatype CLM GUI.

stage
This is the stage in which the policy violation occurred in. It is displayed in various places within the Sonatype CLM GUI, including the associated Application Composition Report.

reportUrl
This is the URL to the Application Composition Report associated with the evaluation that found the listed policy violations

constraintViolations
This is a subcategory for Policy Violations, and includes all information related to specific constraint that was violated.

constraintId
This is the internal id for the constraint, and is not visible in the Sonatype CLM GUI, or in the associated Application Composition Report

constraintName
This is the name of the constraint and is visible in the policy area where the policy was created (i.e either the organization or application). It is also displayed in the Application Composition Report and various tools that connect to Sonatype CLM and display violation data.

reasons
This is a subcategory of Constraint Violations, and gives the reason why the violation occurred.

reason
The reason is formed by the value(s) for the condition(s) violated. Conditions are visible where the policy was created (i.e either the organization or application). It is also displayed in the Application Composition Report and various tools that connect to Sonatype CLM and display violation data.

mavenComponent
The Maven Component is a subcategory of Policy Violations, and includes information about the component(s) causing the violation to occur.

hash
This is the hash value for the component, and is matched to Central, or your list of proprietary components.

groupId, artifactId, version
This is the GAV (GROUP, Artifact ID, Version) for the component.